Microsoft® Silverlight™

Silverlight 2 Data and Communications

Concepts in building Connected Applications with Silverlight 2



Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Unless expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright © 2008 Microsoft Corporation. All rights reserved.

Microsoft are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

Version 1.0

|  |
| --- |
| Introduction |

Estimated Time to Completion: 01:30

In this hands-on lab, you will look at using a variety of remote data sources and communication techniques to retrieve data for a Silverlight application. You will consume WCF web services and use the WebClient, Downloader, and WebRequest classes to fetch data, as well as use the data binding and LINQ to XML features in Silverlight 2. Finally, you will also have the opportunity to work with the ADO.NET Data Services and feed syndication support provided by the ASP.NET Extensions Preview.

The lab is not designed to be worked through from start to finish. It is structured as a series of practical ‘how to’ exercises, illustrating certain techniques. You are encouraged to explore and experiment. Think of an example you’d like to build, and use the instructions in this lab as stepping stones to that goal.

The content for this lab is based on the **Beta 1** release of Silverlight 2. This is a preliminary release that does not represent the final feature set or functionality.

Basic Communications With WebClient and WebRequest

Silverlight 2 includes a number of classes you can use to communicate with a remote server. These include the WebClient and WebRequest classes. We will use the WebClient and WebRequest classes in this exercise to retrieve the current time on the web server.

Please note that all communication APIs in Silverlight are asynchronous by design. You generally ask a component, like a WebClient object, to download a resource from the server. The component will fire an event when it has finished retrieving the resource. Web service calls are also asynchronous.

# WebClient

The WebClient class offers one of the simplest APIs for remote communication.

1. Create a new Silverlight project in Visual Studio 2008 named BasicCommunication. Let Visual Studio create an associated web project.
2. In the Page.xaml file, replace the default <Grid> element with this Canvas and its TextBlock child element

<Canvas x:Name="LayoutRoot" Background="White">

<TextBlock x:Name="\_serverTimeText"

Canvas.Top="10" Canvas.Left="10"

Width="400" Height="300" Text="Hello"

FontSize="52" TextAlignment="Center"

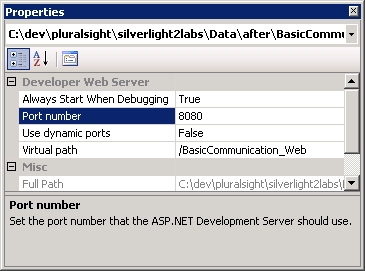
VerticalAlignment="Center" />

</Canvas>

1. Right-click the BasicCommunication\_Web project and select “Add New Item”. Select “Generic Handler” from the list of templates in the Add New Item dialog. Give the handler a name of ServerTime.ashx, and press OK to add the new handler to the web project.
2. Inside the ProcessRequest method of the new ServerTime class, replace the line of code writing “Hello, World” with the following line of code. This code will send the server’s current time as the response to the HTTP request.

context.Response.Write(DateTime.Now.ToLongTimeString());

1. The ASP.NET WebDev server used for development will run the BasicCommunication\_Web project on a random port. We need to change this behavior so we have a consistent port number. Click on the BasicCommunication\_Web project node in Solution Explorer and open the properties window (press F4). Change the “Use dynamic ports” property from True to False, then close the window and save the project. Click on the BasicCommunication\_Web project node in Solution Explorer and open the properties window a second time. Now we can edit the port number, so set the port number to 8080. (**Note, do not use port 80 as that is likely already in use by IIS)**



1. Inside Page.xaml.cs, after the constructor’s call to InitializeComponent, add the following code. This code will start an asynchronous download of the content from our HTTP handler. (You’ll also need to include a *using* statement for System.Net at the beginning of Page.xaml.cs).

WebClient client = new WebClient();

Uri endpoint = new Uri("http://localhost:8080/BasicCommunication\_Web/ServerTime.ashx");

client.DownloadStringCompleted += new DownloadStringCompletedEventHandler(

client\_DownloadStringCompleted);

client.DownloadStringAsync(endpoint);

1. Use the following code for the client\_DownloadStringCompleted event handler. We are simply assigning the result of the WebClient download to the Text property of our TextBlock.

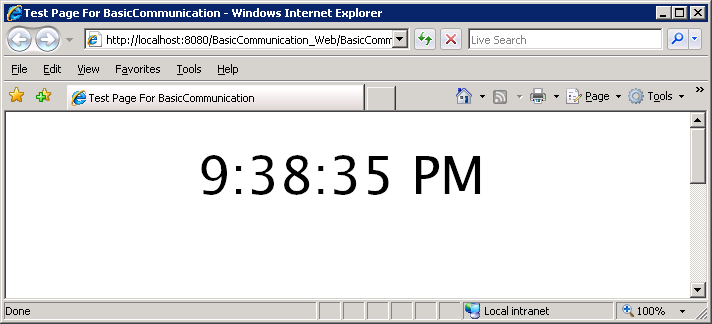
void client\_DownloadStringCompleted(object sender, DownloadStringCompletedEventArgs e)

{

\_serverTimeText.Text = e.Result;

}

1. Run the project and make sure the current time appears in the Silverlight application!



#### WebRequest

The WebRequest class offers a lower-level API for asynchronous download. Using this class we can process the response as a stream, and also POST form values to the server. This section will continue to build on the BasicCommunication and BasicCommunication\_Web projects we created earlier.

1. Right click the BasicCommunication project and select “Add Reference”. Double-click the System.Net assembly to reference the assembly from the Silverlight project.
2. We need to change the behavior of our HTTP handler in the BasicCommunication\_Web project. Inside ServerTime.ashx, replace the ProcessRequest method with the following code. We will look for an HTTP POST request and let the remote user *think* they changed the server’s current time if they POST a properly formatted NewTime value (of course we wouldn’t *really* want remote users changing the clock on the web server, so don’t try this code at home (or at work))!

public void ProcessRequest (HttpContext context)

{

context.Response.ContentType = "text/plain";

DateTime result = DateTime.Now;

if (context.Request.HttpMethod == "POST")

{

// let the user think they changed the server time...

result = DateTime.Parse(context.Request.Form["NewTime"]);

}

context.Response.Write(result.ToLongTimeString());

}

1. Back in the Silverlight project open Page.xaml.cs. We need to include the following using statement at the beginning of the file, if it isn’t already there:

using System.IO;

1. Inside the Page constructor and after the call to InitializeComponent, replace the WebClient code we added earlier with the following WebRequest code. The code will setup the WebRequest for a POST operation. We must process both the request stream and response stream asynchronously. Streams provide an abstract mechanism to read and write data from a backing store. The backing store could be a file on disk, or in the case of the WebRequest, a network connection. See <http://msdn2.microsoft.com/en-us/library/336wast5(VS.80).aspx> for a basic introduction to streaming and I/O in .NET.

Uri endpoint = new Uri("http://localhost:8080/BasicCommunication\_Web/ServerTime.ashx");

WebRequest request = WebRequest.Create(endpoint);

request.Method = "POST";

request.ContentType = "application/x-www-form-urlencoded";

request.BeginGetRequestStream(new AsyncCallback(RequestReady), request);

request.BeginGetResponse( new AsyncCallback(ResponseReady), request);

1. First, we will implement the RequestReady method. In this method we need to write data into the request stream. The request stream will transport data to the server. This code writes our form value into the request stream - we need to write just a single form value named “NewTime”.

void RequestReady(IAsyncResult asyncResult)

{

WebRequest request = asyncResult.AsyncState as WebRequest;

Stream requestStream = request.EndGetRequestStream(asyncResult);

StreamWriter writer = new StreamWriter(requestStream);

writer.Write("NewTime=11:58 AM");

writer.Flush();

}

1. Finally, we implement the ResponseReady method to process the reply from the server. Notice how we process the response as a stream instead of downloading the entire response as we did with the WebClient class. In this scenario the streaming is overkill, but with larger payloads, the streaming API offers the opportunity for optimization (like cancelling the download before the entire payload is received).

void ResponseReady(IAsyncResult asyncResult)

{

WebRequest request = asyncResult.AsyncState as WebRequest;

using (WebResponse response =

request.EndGetResponse(asyncResult))

using(Stream responseStream =

response.GetResponseStream())

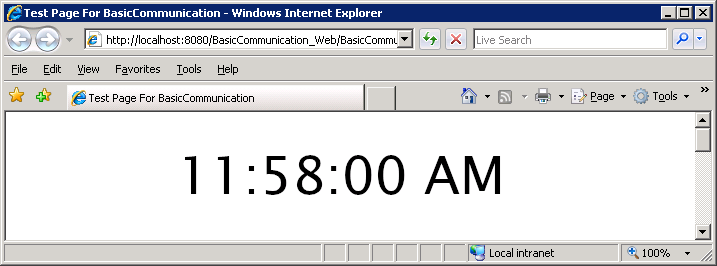
{

StreamReader reader = new StreamReader(responseStream);

\_serverTimeText.Text = reader.ReadToEnd();

}

}

1. Run the project to ensure the Silverlight application has posted a value to the server! 

# Fetching and Binding Remote Data

The Silverlight WebClient and WebRequest classes can be used for a variety of scenarios. One is to retrieve and bind to data. In this section we will retrieve a mashed up RSS feed of blog content.

1. Create a new Silverlight project in Visual Studio with a name of Mashup. Allow Visual Studio to create an associated web site project.
2. In Page.xaml, replace the default <Grid> with the following XAML. We will use some of the data binding features in Silverlight to bind the title of each blog post to a TextBlock object’s Text property. Later we will see how the ItemsControl widget will build a TextBlock for each blog post we have to the ItemsControl. Note: Visual Studio might say the XML inside the ItemTemplate element contains errors, but you can ignore these messages.

<Canvas x:Name="LayoutRoot" Width="400" Height="300" Background="White">

<ItemsControl x:Name="\_itemTitles">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Vertical">

<TextBlock FontWeight="Bold" Text="{Binding Title.Text}" />

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

</Canvas>

1. Right-click the Mashup\_Web project and click “Add New Item”. In the dialog that follows, select the Generic Handler template, and give the template the name of Mashup.ashx.
2. The ASP.NET WebDev server used for development will run the Mashup\_Web project on a random port. We need to change this behavior so we have a consistent port number. Click on the Mashup\_Web project node in Solution Explorer and open the properties window (press F4). Change the “Use dynamic ports” property from True to False, then close the window and save the project. Click on the Mashup\_Web project node in Solution Explorer and open the properties window *a second time*. Now we can edit the port number, so set the port number to 8081.
3. Add the following statements to the top of the Mashup.ashx file, after the @WebHandler directive.

using System.Xml;

using System.Linq;

using System.Collections.Generic;

using System.ServiceModel.Syndication;

1. Delete all the code inside the ProcessRequest method.
2. Inside ProcessRequest, create an array of strings containing the RSS URLs of some of your favorite blogs. For example:

string[] urls = {

"http://blogs.msdn.com/webnext/rss.xml",

"http://blogs.msdn.com/tims/rss.xml",

"http://feeds.feedburner.com/JesseLiberty-SilverlightGeek"

};

1. The next step is to load the contents of each blog into a SyndicationFeed. SyndicationFeed is part of System.ServiceModel in .NET 3.5, and allows us to easily consume and build feeds using RSS and ATOM. Place the following code after the array definition.

List<SyndicationFeed> feeds = new List<SyndicationFeed>();

foreach (string url in urls)

{

using (XmlReader feedReader = XmlReader.Create(url))

{

feeds.Add(SyndicationFeed.Load(feedReader));

}

}

1. We’ll use LINQ to Objects to mash the feeds into a single collection of items. Place the following code after the code from step 9.

DateTime minDate = DateTime.Now.AddDays(-30);

var mashedItems =

from feed in feeds

from item in feed.Items

where item.PublishDate > minDate

orderby item.PublishDate descending

select item;

1. Our next step is to write respond with our mashed up blog feed in an RSS 2 format.

Add the following code at the end of the ProcessRequest method (after the code in step 10).

int maxResults = 15;

SyndicationFeed mashedFeed = new

SyndicationFeed(mashedItems.Take(maxResults));

using (XmlWriter writer =

XmlWriter.Create(context.Response.OutputStream))

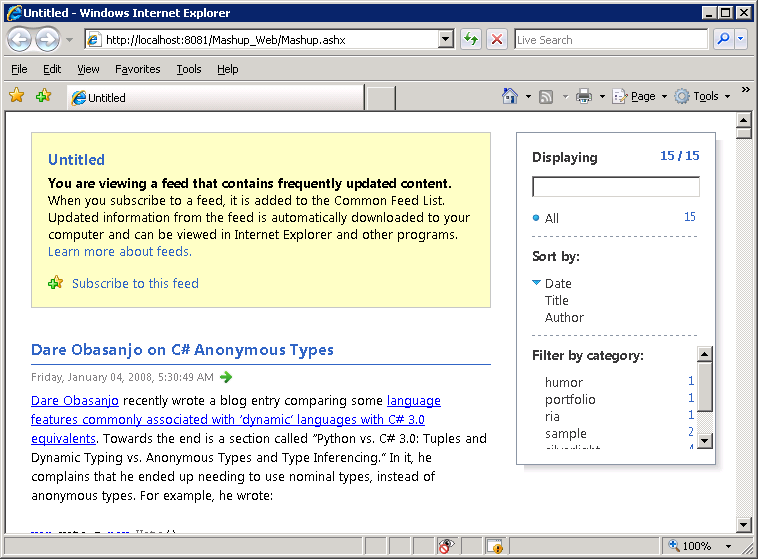
{

context.Response.ContentType = "text/xml";

mashedFeed.SaveAsRss20(writer);

}

1. We should be able to right-click the Mashup.ashx file and select “View In Browser” to preview our new RSS feed.



1. Back in the Silverlight project Mashup, we can also utilize the SyndicationFeed class to process RSS and Atom feeds. First, though, we’ll need an assembly reference. Right-click the Mashup project and select “Add Reference”. Click the .NET Tab and find the reference to ‘System.ServiceModel.Syndication.dll’ and add it. If it isn’t present on the list, then click the Browse tab. The Syndication features for Silverlight are included in an assembly provided with the Silverlight 2 SDK. The SDK assemblies are typically installed into Program Files\Microsoft SDKs\Silverlight\v2\Libraries. Browse to this location and double-click System.ServiceModel.Syndication.dll.
2. Next, move to the Page.xaml.cs file. In the Page constructor, after the call to InitializeComponent, add the following code. This code will setup a WebClient object to fetch our newly built RSS feed.

WebClient client = new WebClient();

Uri endpoint = new Uri("http://localhost:8081/Mashup\_Web/Mashup.ashx");

client.OpenReadCompleted += new

OpenReadCompletedEventHandler(client\_OpenReadCompleted);

client.OpenReadAsync(endpoint);

1. Make sure to include the following statements at the top of Page.xaml.cs.

using System.Xml;

using System.IO;

using System.ServiceModel.Syndication;

1. Our last step is to process the results retrieved from the WebClient. We’ll use the following code for its OpenReadCompleted event handler.

void client\_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)

{

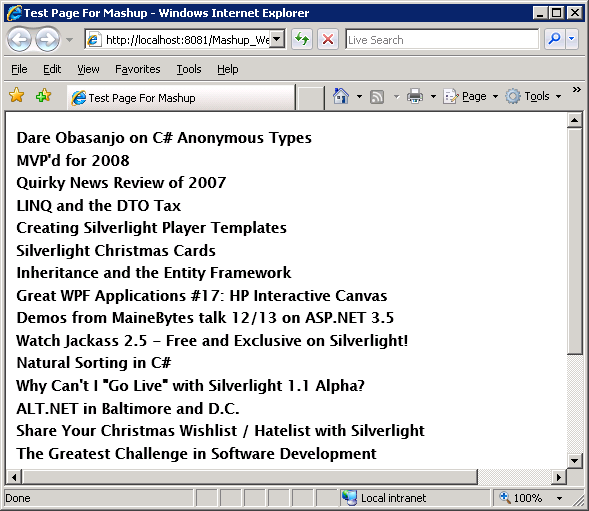
XmlReader r = XmlReader.Create(e.Result);

SyndicationFeed feed = SyndicationFeed.Load(r);

\_itemTitles.ItemsSource = feed.Items;

}

1. We should now see our Silverlight application display the recent titles from our favorite blog posts.



# WCF and Silverlight

Silverlight 2 can use System.ServiceModel to consume web services from a variety of sources. In this excersise we will build and consume a web service using WCF.

1. Create a new Silverlight project in Visual Studio with a name of ServerDiagnostics. Allow Visual Studio to create an associated web site project.
2. The ASP.NET WebDev server used for development will run the ServerDiagnostics \_Web project on a random port. We need to change this behavior so we have a consistent port number. Click on the ServerDiagnostics \_Web project node in Solution Explorer and open the properties window (press F4). Change the “Use dynamic ports” property from True to False, then close the window and save the project. Click on the ServerDiagnostics \_Web project node in Solution Explorer and open the properties window a second time. Now we can edit the port number, so set the port number to 8082.
3. Right-click the ServerDiagnostics\_Web project and select “Add New Item”. Inside the resulting dialog, select the Class template and give the new file a name of ModuleSummary.cs. Allow Visual Studio to place the file into the App\_Code directory.
4. Add the following statement to the top of ModuleSummary.cs

using System.Runtime.Serialization;

1. Modify the ModuleSummary class so the class looks like the following.

[DataContract]

public class ModuleSummary

{

[DataMember]

public String Name { get; set; }

[DataMember]

public String Version { get; set; }

}

1. Right-click the ServerDiagnostics\_Web project and select “Add New Item”. This time, select the WCF Service template. Give the service a name of Modules.svc.
2. Inside the IModules.cs file (from the App\_Code directory), modify the IModules interface to look like the following code.

[ServiceContract]

public interface IModules

{

[OperationContract]

ModuleSummary[] GetModules();

}

1. Next, open the Modules.cs file in App\_Code. Add the following statement to the top of the file.

using System.Diagnostics;

1. Still in the Modules.cs file, replace the Modules class definition with the following code.

public class Modules : IModules

{

public ModuleSummary[] GetModules()

{

ProcessModuleCollection modules =

Process.GetCurrentProcess().Modules;

return (

from m in modules.OfType<ProcessModule>()

select new ModuleSummary

{

Name = m.ModuleName,

Version = m.FileVersionInfo.FileVersion

}

).ToArray();

}

}

1. We need to make a small change to our WCF service binding. Open the web.config file in the web project and find the <system.serviceModel> configuration section. Find the <endpoint> node underneath the <service> node. Notice the binding is current wsHttpBinding. We need to change the binding to basicHttpBinding, as shown in the highlighted section below.

<services>

<service behaviorConfiguration="ModulesBehavior" name="Modules">

<endpoint address="" **binding="basicHttpBinding"**

contract="IModules">

<identity>

<dns value="localhost" />

</identity>

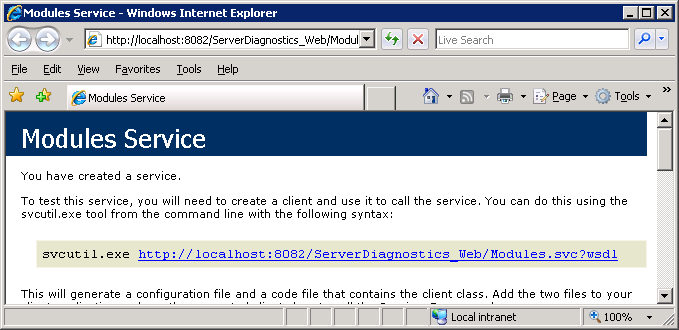
</endpoint>

<endpoint address="mex" binding="mexHttpBinding"

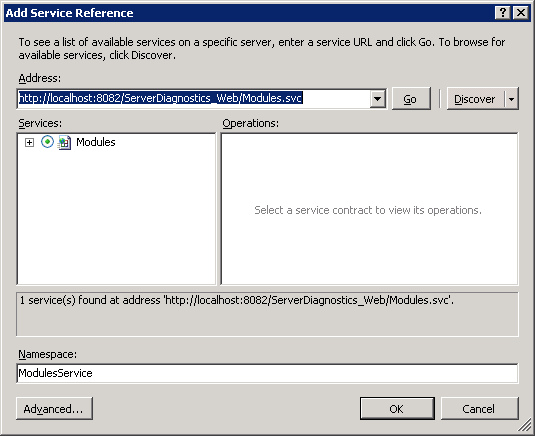
contract="IMetadataExchange" />

</service>

1. Right-click the Modules.svc file, and select ‘View in Browser’ to ensure the project runs without any errors.



1. Close the browser and then go to the Silverlight project. Right-click the project and select “Add Service Reference”. In the “Add Service Reference” dialog box, enter the URL to the Modules.svc file (http://localhost:8082/ServerDiagnostics\_Web/Modules.svc) and press Go. In the Namespace textbox, enter ModulesService, then press OK.



1. In Page.xaml, we’ll prepare a UI by placing the following XAML inside the root layout canvas.

<ItemsControl x:Name="\_modulesItems">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<TextBlock FontSize="10" Text="{Binding Name}" />

<TextBlock Text=" "></TextBlock>

<TextBlock FontSize="10" Text="{Binding Version}" />

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

1. In Page.xaml.cs, we’ll need to add the following statements to the beginning of the file.

using System.ServiceModel;

using System.ServiceModel.Channels;

using ServerDiagnostics.ModulesService;

1. Place the following code in the Page constructor, just after the call to InitializeComponents. The code will invoke the WCF service asynchronously.

Binding binding = new BasicHttpBinding();

EndpointAddress endPoint = new EndpointAddress(

"http://localhost:8082/ServerDiagnostics\_Web/Modules.svc");

ModulesClient client = new ModulesClient(binding, endPoint);

client.GetModulesCompleted +=

new EventHandler<GetModulesCompletedEventArgs>

(client\_GetModulesCompleted);

client.GetModulesAsync();

1. To harvest the WCF service results, we need the following event handler.

void client\_GetModulesCompleted(object sender,

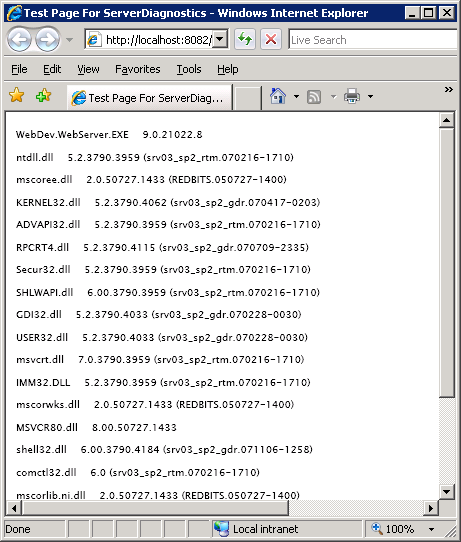
GetModulesCompletedEventArgs e)

{

\_modulesItems.ItemsSource = e.Result;

}

1. Finally, run the Silverlight application and view the results!



# Consuming ADO.NET Data Services

The ASP.NET Extensions Preview includes the ADO.NET Data Services framework. This framework allows you to expose data entities as URI addressable resources. The framework is commonly used in combination with the ADO.NET Entity Framework to serve relational data over the web, however, any object implementing IQueryable<T> can provide data. In this exercise we will use the framework to expose a simple collection of objects.

1. Create a new Silverlight project in Visual Studio 2008 named StudentDataServices. Allow Visual Studio to create an associated web project.
2. The ASP.NET WebDev server used for development will run the StudentDataServices \_Web project on a random port. We need to change this behavior so we have a consistent port number. Click on the StudentDataServices \_Web project node in Solution Explorer and open the properties window (press F4). Change the “Use dynamic ports” property from True to False, then close the window and save the project. Click on the StudentDataServices \_Web project node in Solution Explorer and open the properties window a second time. Now we can edit the port number, so set the port number to 8083.
3. In the StudentDataService\_Web project, add a reference to the Microsoft.Data.Web.dll assembly. You can find the assembly in “Program Files\Reference Assemblies\Microsoft\Framework\ASP.NET 3.5 Extensions”. Note that you will need the Microsoft ASP.NET Extensions Preview installed. These are available to download at: http://www.microsoft.com/downloads/details.aspx?FamilyId=A9C6BC06-B894-4B11-8300-35BD2F8FC908&displaylang=en
4. Add a new class to the StudentDataServices\_Web project (Visual Studio will place the class file into App\_Code). Name the file Student.cs.
5. Inside Student.cs, give the Student class ID (int), Name, Age, and Address properties.
6. When working with a relational database, the ADO.NET Data Services framework can use metadata from the database (like a table’s primary key) to identify the property (or properties) that make an entity unique. When working with a collection of objects like in this exercise, we need to provide a hint. Apply the [DataWebKey] attribute to the ID property in the Student class. The class should now look like the following.

using System;

using Microsoft.Data.Web;

public class Student

{

[DataWebKey]

public int ID { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Address { get; set; }

}

1. Add another class to the StudentDataServices\_Web project. This class will have a name of StudentDataContainer. Our StudentDataContainer will be responsible for providing an IQueryable<Student> source of data with the following code. ADO.NET Data Services will use reflection to find the data source.

using System;

using System.Linq;

using System.Collections.Generic;

public class StudentDataContainer

{

public StudentDataContainer()

{

// create some fake student data

\_list.Add(new Student

{ ID=1, Name="Alex", Age=38, Address="WA" });

\_list.Add(new Student

{ ID=2, Name="Chris", Age=26, Address="CA" });

\_list.Add(new Student

{ ID = 3, Name = "Vicky", Age = 22, Address = "PA" });

}

public IQueryable<Student> Students

{

get { return \_list.AsQueryable(); }

}

List<Student> \_list = new List<Student>();

}

1. Now we are ready to build a data service. Right-click StudentDataServices\_Web and select “Add New Item”. Locate the ADO.NET Data Services template, and enter StudentDataService.svc as the name of the new item. Press OK.
2. In the App\_Code directory of the web project you should now have a StudentDataService.cs file. We need to edit the file a bit to tell the ADO.NET Data Services where to look for data and what operations are allowed. These changes are detailed in the TODO comments of the C# file. The final version of the file will look like the following and point the ADO.NET Data Services to our StudentDataContainer.

using System;

using Microsoft.Data.Web;

public class StudentDataService :

WebDataService<StudentDataContainer>

{

public static void InitializeService(

IWebDataServiceConfiguration config)

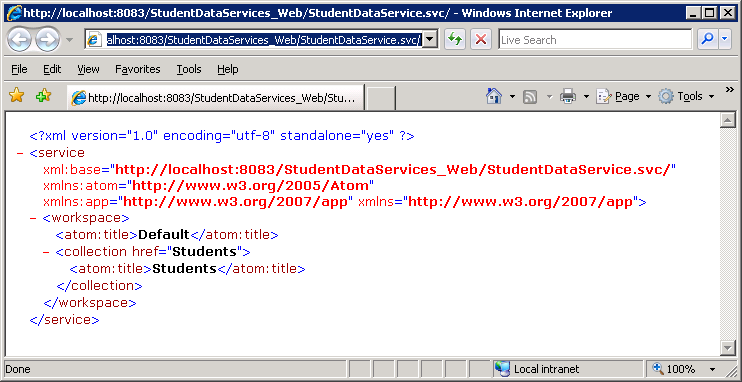
{

config.SetResourceContainerAccessRule(

"\*", ResourceContainerRights.AllRead);

}

}

1. At this point, our data service should be able to run. Right-click the StudentDataService.svc file in the web project and then select “View in Browser”. We should see the following XML appear. 
2. Experiment with querying the new data service. By requesting <http://localhost:8083/StudentDataServices_Web/StudentDataService.svc/Students> in the browser you should see a list of all three students. **You might need to use View Source in Internet Explorer to see the underlying XML, as IE tries to interpret the XML as a blog feed.** Requesting <http://localhost:8083/StudentDataServices_Web/StudentDataService.svc/Students(2)> will retrieve the Student with an ID of 2 and display the result in XML. JSON formatting is also available from ADO.NET Data Services.
3. Over in the Silverlight project, open Page.xaml and add the following code inside the layout canvas. We are setting up an ItemsControl to display data we fetch from the Students data service.

<ItemsControl x:Name="\_studentList" ItemsSource="{Binding}">

<ItemsControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding Name}" />

<TextBlock Text="Age: " />

<TextBlock Text="{Binding Age}" />

</StackPanel>

</DataTemplate>

</ItemsControl.ItemTemplate>

</ItemsControl>

1. Inside Page.xaml.cs, we will fetch a list of all Students in the data service. We’ll use a Downloader object to retrieve the XML. Place the following code inside the Page constructor (but after the call to InitializeComponents).

Uri allStudentsUri = new Uri("http://localhost:8083/StudentDataServices\_Web/StudentDataService.svc/Students");

WebClient client = new WebClient();

client.OpenReadCompleted += new

OpenReadCompletedEventHandler(client\_OpenReadCompleted);

client.OpenReadAsync(allStudentsUri);

1. There are a number of techniques we could use to process the XML we retrieve from the data service. In this exercise we also want to demonstrate the use of LINQ to XML in Silverlight 2. First, we’ll need an assembly reference. Right-click the StudentDataServices Silverlight project, select ‘Add Reference’ and browse to the System.Xml.Linq assembly. If it isn’t on the .NET tab, you can browse to it in the Program Files\Microsoft SDKs\Silverlight\v2\Libraries directory.
2. We are going to use a proxy class to store Student information after we’ve queried the XML. Add a new class file to the Silverlight project with the name of Student.cs and add the following code.

using System;

public class Student

{

public int ID { get; set; }

public string Name { get; set; }

public int Age { get; set; }

public string Address { get; set; }

}

1. Inside the client\_OpenReadCompleted event handler, we’ll need to place the XML coming back from the data service into a LINQ to XML XDocument instance for querying.

First make sure that these lines are present at the top of your code:

using System.Xml;

using System.Xml.Linq;

Then add these lines to the client\_OpenReadCompleted event handler

XmlReader r = XmlReader.Create(e.Result);

XDocument students = XDocument.Load(r);

1. If you’ve been experimenting with the data service, you might have noticed all the XML namespaces in play. We’ll need to properly utilize XML namespaces for our LINQ to XML queries to work. Add the following code next:

XNamespace xmlns = "http://www.w3.org/2005/Atom";

XNamespace ads = "http://schemas.microsoft.com/ado/2007/08/dataweb";

1. Finally, we need to add our LINQ to XML query. The query will extract Student information from the data service’s XML and create Student objects. Add the following code next:

\_studentList.DataContext =

from x in students.Descendants(xmlns + "entry")

select new Student

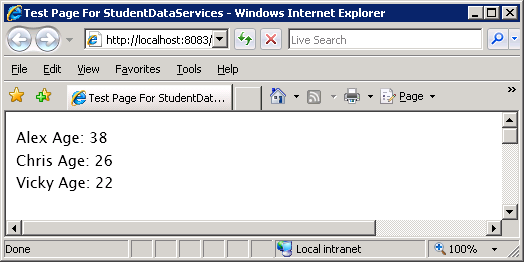
{

Name = x.Descendants(ads + "Name").First().Value,

Age = int.Parse(

x.Descendants(ads + "Age").First().Value)

};

1. Now run the Silverlight project and watch the application consume a data service. 

## Summary

There are a variety of communication mechanisms available in Silverlight 2, you just need to pick the best tool for the job. Combining WCF and the lower level communication APIs with technologies like LINQ, syndication, and ADO.NET Data Services will allow you to create compelling Silverlight applications full of dynamic data pulled over the network.

