## Data Access – Client Side

**Lab Time:** 45 minutes

**Lab Folder**: C:\Student\Labs\DataAccessClientSide

**Lab Overview:** The Client Object Model extends the familiar server-side object model to the client. The Client Object Model comes in two flavors: .NET and JavaScript. The .NET client object model can be used with console applications, Windows applications, and Silverlight applications. The JavaScript client object model is used in client scripting. In this lab, you’ll make use of the Client Object Model to build some applications.

**Lab Setup** Requirements

* This lab requires the OpenXML 2.0 SDK. This should already be installed on your student VM but if it is not install from this link  
  **http://www.microsoft.com/downloads/details.aspx?FamilyID=c6e744e5-36e9-45f5-8d8c-331df206e0d0**.
* Before you begin this lab, you must run the batch file named **SetupLab.bat**. This batch file creates a new SharePoint site collection at the location **http://intranet.wingtip.com/sites/ClientData**.

### Exercise 1: Retrieving Lists with the .NET Client Object Model

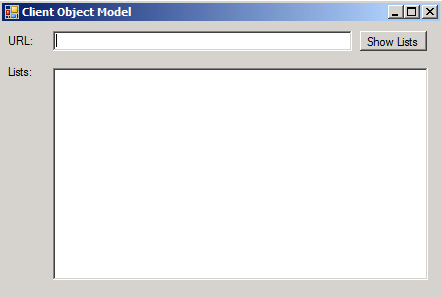
In this exercise you will create the first part of a Windows application for printing SharePoint lists as Word documents. In this exercise, you will retrieve the available lists from a SharePoint site.

1. Launch the Internet Explorer and navigate to the top-level site at **http://intranet.wingtip.com/sites/ClientData**. Take a moment to inspect the site and make sure it behaves as expected. Note that the setup script creates a new site collection with a Blank site as its top-level site.
2. Launch Visual Studio 2010 and create a new **Windows Forms Application** project under the **Visual C# » Windows** section using the **File » New » Project** menu. Give it a name of **ListPrinter**.
3. Add a reference to the Client Object Model by selecting **Project » Add Reference** from the Visual Studio main menu. In the **References** dialog, browse to C:\Program Files\Common Files\Microsoft Shared\web server extensions\14\ISAPI and select the following assemblies, then click OK to add the references to the project:

Microsoft.SharePoint.Client.dll

Microsoft.SharePoint.Client.Runtime.dll

1. Drag a TextBox control onto Form1 and give it a name of **UrlTextBox**. You’ll use this TextBox control to specify the URL for the site where you want to access list data.
2. Drag a ListBox control onto Form1 and give it a name of **ListsListBox**. You’ll use this to show the lists available from the site.
3. Drag a Button control onto Form1 and give it a name of **ShowButton**. You’ll use this button to connect with the site and fill the ListBox. Change the Text property of this button to **Show Lists**.
4. You can optionally drag additional labels onto the form to make it look like the following:



1. Double-click the Button control named **ShowButton** to open the code window and create an event handler for the button’s Click event. To avoid ambiguous references between the Client Object Model and the Windows Forms namespace, add the following statement to the top of the code module.

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 ClientOM = Microsoft.SharePoint.Client;

1. Enter the following code in the button1\_Click() event to retrieve the available lists from the target site.

private void ShowButton\_Click(object sender, EventArgs e) {

ListsListBox.Items.Clear();

using (ClientOM.ClientContext ctx =

new ClientOM.ClientContext(UrlTextBox.Text)) {

ClientOM.Web site = ctx.Web;

ctx.Load(site.Lists);

ctx.ExecuteQuery();

foreach (ClientOM.List list in site.Lists) {

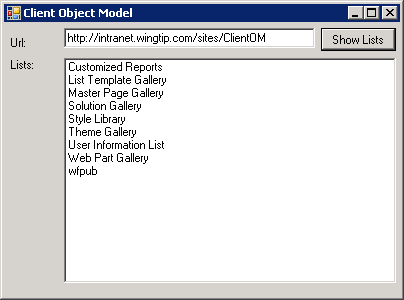
ListsListBox.Items.Add(list.Title);

}

}

}

1. Run your project, enter **http://intranet.wingtip.com/sites/ClientData** in the TextBox control and click the **Show Lists** button. Verify that you are returning all lists from the site including hidden lists and galleries such as the **Master Page Gallery** and the **User Information List** as shown below.



1. Stop the project and return to Visual Studio. Now you are going to modify the code you have just written to only return lists that have a Hidden attribute value of false. Accomplish this by adding a lambda expression to your load statement as shown in the following code block.

private void ShowButton\_Click(object sender, EventArgs e) {

ListsListBox.Items.Clear();

using (ClientOM.ClientContext ctx =

new ClientOM.ClientContext(UrlTextBox.Text)) {

ClientOM.Web site = ctx.Web;

ctx.Load(site, s => s.Lists.Where( list => list.Hidden==false));

ctx.ExecuteQuery();

foreach (ClientOM.List list in site.Lists) {

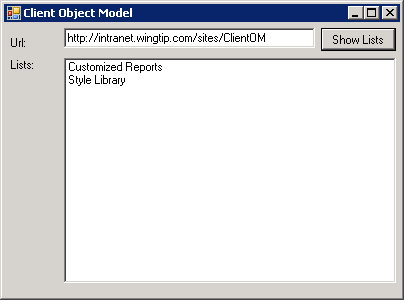
ListsListBox.Items.Add(list.Title);

}

}

}

1. After rerunning your project, you should be able to verify that your list box is now being filled with only those lists that are not hidden as seen below.



In this exercise you created a Windows Forms application that used the SharePoint .NET Client Object Model to query a SharePoint site for all the lists in the site and display them in a listbox.

### Exercise 2: Printing a List with the .NET Client Object Model

In this exercise you will finish the application by using a combination of the Client Object model and the Open XML 2.0 SDK for Office to create a Word document that contains list items from a selected list. This exercise requires that the OpenXML 2.0 SDK be installed.

1. Using the same Windows Form application from the previous exercise, drag a new Button control to Form1 and give it a name of PrintButton. This second button will be used to create a Word document from items in a chosen list.

**Note:** When creating the project, make sure you specify the location where the project is created to something other than the default c:\users\[username]\Documents\Visual Studio 2010…. The reason for this is that this deeply nested path will cause issues when adding references to the two assemblies used in the Silverlight Client Object Model because of the way the Visual Studio 2010 project files add relative references. The relative reference to the path of the assemblies exceeds the maximum number of characters for a path and Visual Studio 2010 will report it can’t find the two Silverlight assemblies. Therefore, when specifying the path it is a good idea to put it in the [[LAB FILES]] folder.

1. Change the Text property for the Button to **Print List**.
2. Add a reference to the **OpenXML SDK 2.0 API** by selecting **Project » Add Reference** from the Visual Studio main menu. When the **References** dialog appears, set a reference to the DocumentFormat.OpenXML.dll assembly by browsing to the C:\Program Files (x86)\OpenXML SDK\v2.0\Lib directory and double-clicking on the \*.DLL file.
3. Add another reference to **WindowsBase.dll** assembly by following the same steps, except browse to C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.0 and select to set a reference to WindowsBase.dll.
4. Witn Form1 open in **Design** mode, double click the **Print List** Button to open the code window and to generate an event handler for the button’s click event.
5. Add the following statements to the top of the code module to use the Open XML 2.0 API.

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 ClientOM = Microsoft.SharePoint.Client;

using DocumentFormat.OpenXml;

using DocumentFormat.OpenXml.Packaging;

using DocumentFormat.OpenXml.Wordprocessing;

1. Add the following code to the Click event of the **Print List** button to retrieve the columns and list items that you will need to build the document. Note that this code can be copy and pasted from the snip1.txt file in the **[[LAB FILES]]\StarterFiles** folder for this lab:

private void PrintButton\_Click(object sender, EventArgs e) {

if (ListsListBox.SelectedIndex > -1) {

using (ClientOM.ClientContext ctx =

new ClientOM.ClientContext(UrlTextBox.Text)) {

string listTitle = ListsListBox.SelectedItem.ToString();

ClientOM.Web site = ctx.Web;

ctx.Load(site, s => s.Lists.Where(l => l.Title == listTitle));

ctx.ExecuteQuery();

ClientOM.List list = site.Lists[0];

ctx.Load(list, l => l.Fields.Where(f => (f.Hidden == false) &&

(f.CanBeDeleted == true ||

f.InternalName == "Title")));

ctx.ExecuteQuery();

ClientOM.ListItemCollection listItems =

list.GetItems(ClientOM.CamlQuery.CreateAllItemsQuery());

ctx.Load(listItems);

ctx.ExecuteQuery();

// DOCUMENT CREATION CODE GOES HERE

}

MessageBox.Show("Document Created!");

}

}

The Open XML formats use a set of XML documents to represent all of the components of an Office 2007 document. These components are referred to as “parts”. Parts are designated through content type Uris (not to be confused with SharePoint content types!). So, the process of creating a Word document involves the proper creation and packaging of the required “parts”. The OpenXML 2.0 SDK makes it easier to create these parts because it provides strongly-typed objects from which you can build a document. The document you will create will contain a table with columns for each field in the list.

1. Add the following code into your project to build the document. Note that this code can be copy and pasted from the snip2.txt file in the **[[LAB FILES]]\StarterFiles** folder for this lab.

private void button2\_Click(object sender, EventArgs e)

{

...

ctx.Load(list.Items);

ctx.ExecuteQuery();

// DOCUMENT CREATION CODE GOES HERE

using (WordprocessingDocument package =

WordprocessingDocument.Create(@"c:\Student\" +

ListsListBox.SelectedItem.ToString() + ".docx",

WordprocessingDocumentType.Document)) {

Body body = new Body();

Table table = new Table();

//Columns

TableRow colRow = new TableRow();

foreach (ClientOM.Field field in list.Fields) {

TableCell colCell = new TableCell();

colCell.Append(new Paragraph(new Run(new Text(field.Title))));

colRow.Append(colCell);

}

table.Append(colRow);

//Rows

foreach (ClientOM.ListItem item in listItems) {

TableRow dataRow = new TableRow();

foreach (ClientOM.Field field in list.Fields) {

TableCell dataCell = new TableCell();

string dataVal = string.Empty;

try { dataVal = item[field.InternalName].ToString(); }

catch { dataVal = "-"; }

dataCell.Append(new Paragraph(new Run(new Text(dataVal))));

dataRow.Append(dataCell);

}

table.Append(dataRow);

}

body.Append(table);

//Build document package

package.AddMainDocumentPart();

package.MainDocumentPart.Document = new Document(body);

package.MainDocumentPart.Document.Save();

package.Close();

}

}

MessageBox.Show("Document Created!");

}

}

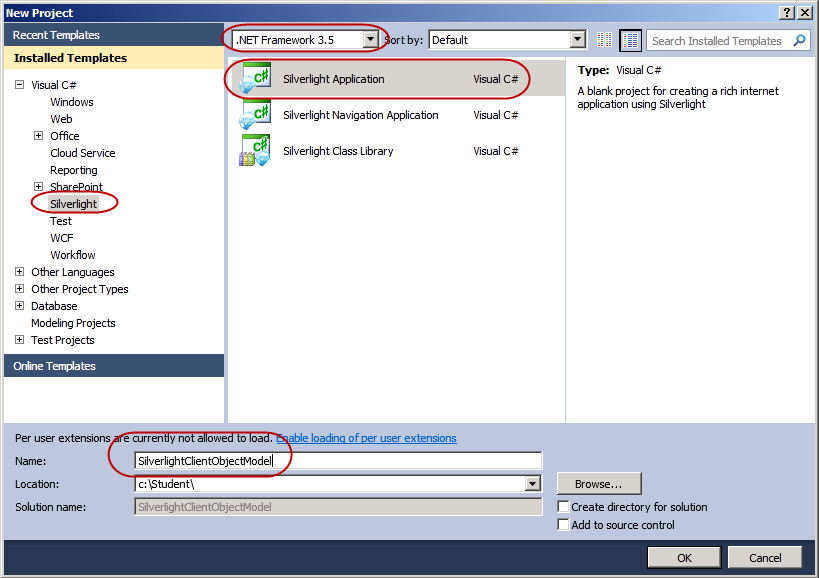
1. Once you have finished coding, verify that the project compiles and start it in Visual Studio by building it. You should be able to enter a URL for a site and retrieve the lists for the site. Then select a list from the site and print it. A new Word document will appear in the root of the C:\Student folder having a name that is the title of the selected list.

In this exercise you added code to the existing Windows Forms application that prints the contents of a SharePoint list to a Word document using the OpenXML 2.0 SDK.

### Exercise 3: Creating Lists with the Silverlight Client Object Model

In this exercise you will create a Silverlight application that leverages the Silverlight Client Object Model to create a new SharePoint list. This Silverlight application will be hosted in the OOTB Silverlight Web Part.

1. In Visual Studio 2010, create a new **Silverlight Application** project found in the **Visual C# » Silverlight** category start a new project by selecting **File » New » Project**. Make sure you have targeted the **.NET Framework 3.5** version and give it a name of **SilverlightClientObjectModel**.



1. Visual Studio will next prompt you for some additional information. When creating a Silverlight application, Visual Studio can create an associated Web project which can be used to test the Silverlight application. In this case, that Web application won’t be of much use because the Silverlight application will make a remote call to a SharePoint site. By default these cross-domain queries will be blocked by SharePoint. It is possible to get around this (using a crossdomain.xml file… as documented on MSDN here: **http://msdn2.microsoft.com/en-us/library/cc197955(VS.95).aspx)**, however this exercise will go for the more straightforward approach. Use the following to set the values in the next dialog:

Host the Silverlight application in a new Web site – **UNCHECKED**

Silverlight Version – **Silverlight 3.0**

1. The first step is to create the UI needed in the Silverlight application. Open the MainPage.xaml file and using the Toolbox, add three (3) TextBlocks, two (2) TextBoxes and one (1) Button to the designer. You could also type the markup in manually using the following snippet:

<UserControl x:Class="SilverlightClientObjectModel.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">

<Grid x:Name="LayoutRoot" Background="White" Height="217" Width="310">

<TextBlock Height="23" HorizontalAlignment="Left" Margin="12,12,0,0" Name="textBlock1" Text="SharePoint Site URL:" VerticalAlignment="Top" Width="120" />

<TextBlock Height="23" HorizontalAlignment="Left" Margin="12,74,0,0" Name="textBlock2" Text="New Announcements List Name:" VerticalAlignment="Top" Width="202" />

<TextBox Height="23" HorizontalAlignment="Left" Margin="12,41,0,0" Name="SharePointSiteTextBox" VerticalAlignment="Top" Width="278" />

<TextBox Height="23" HorizontalAlignment="Left" Margin="12,103,0,0" Name="NewListNameTextBox" VerticalAlignment="Top" Width="278" />

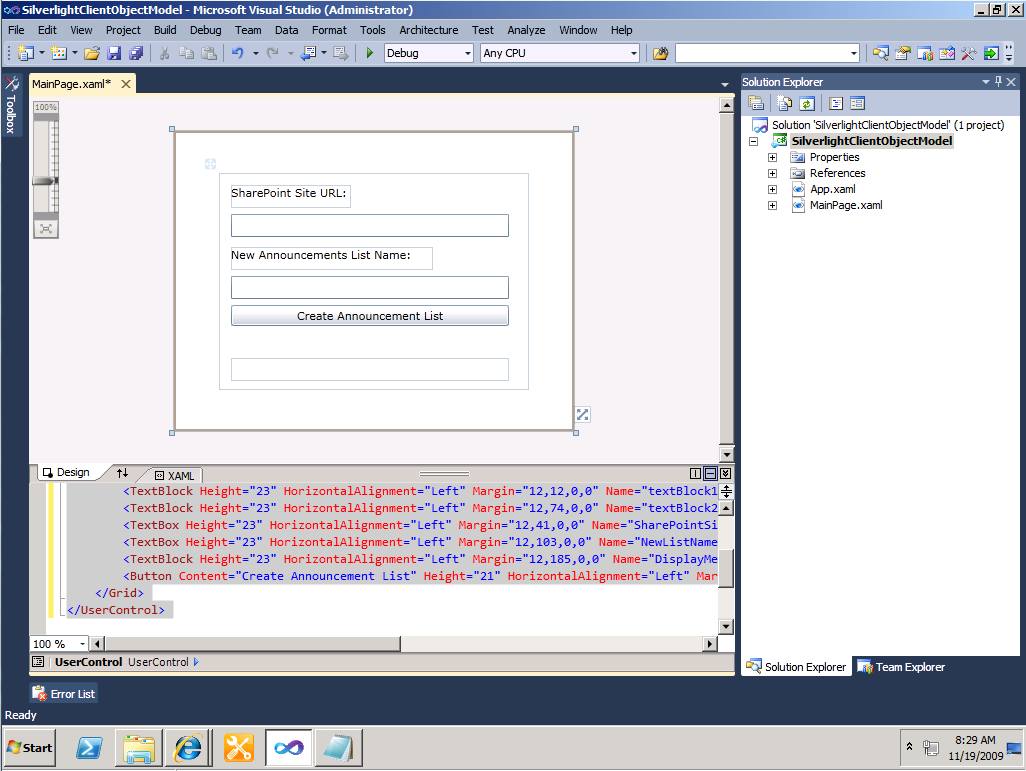
<TextBlock Height="23" HorizontalAlignment="Left" Margin="12,185,0,0" Name="DisplayMessageTextBlock" Text="" VerticalAlignment="Top" Width="278" />

<Button Content="Create Announcement List" Height="21" HorizontalAlignment="Left" Margin="12,132,0,0" Name="button1" VerticalAlignment="Top" Width="278" />

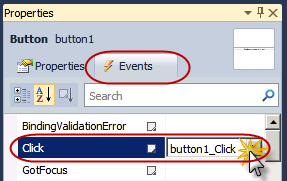
</Grid>

</UserControl>

1. Notice in the previous code snippet some of the object Name and Content attributes have been changed… make sure you reflect this within your markup as well. The three important ones are the two TextBoxes & the TextBlock at the bottom of the application. These changes can be made in the XAML window or in the Properties tool window (which you can bring up by pressing **[F4]**). Your application should look similar to the following figure:



1. In the designer select the button and look at the Properties tool window (if it isn’t visible, press **[F4]**). Select **Events** and double-click the box next to the **Click** event. This will create an event handler in the code file of the page.



1. Before adding the necessary code, you need to add two references to the Silverlight client object model. Right-click the References node within the project in the Solution Explorer tool window and select **Add Reference…**. From the **Browse** tab, navigate to the c:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\LAYOUTS\ClientBin folder and select the two assemblies in that folder:

Microsoft.SharePoint.Client.Silverlight.dll

Microsoft.SharePoint.Client.Silverlight.Runtime.dll

1. Now it’s time to add the business logic to the Silverlight application. If it isn’t already open, open the MainPage.xaml.cs file.
2. Add a few extra using statements to the top of this file:

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 Microsoft.SharePoint.Client;

using System.Threading;

1. Before building out the logic for the button’s Click event, take a second to setup some Silverlight specific plumbing. All Silverlight communication is done asynchronously and also on a separate thread from the user interface. Setting this up requires a bit of extra work. First create a delegate and method that will be used to update the UI:

public partial class MainPage : UserControl

{

public MainPage()

{

InitializeComponent();

}

private delegate void UpdateUI();

private void DisplaySuccessResults()

{

DisplayMessageTextBlock.Text = "List created!";

}

}

1. Next, create the success & fail callbacks when the query is sent to SharePoint. Add the following two methods to the MainPage class. The OnSuccess() method creates an instance of the delegate and then fires it off asynchronously. The OnFail() method simply reports an error:

private void OnSuccess(object sender, ClientRequestSucceededEventArgs args)

{

UpdateUI updater = DisplaySuccessResults;

this.Dispatcher.BeginInvoke(updater);

}

private void OnFail(object sender, ClientRequestFailedEventArgs args)

{

MessageBox.Show("Error occurred: " + args.Message);

}

1. The last step is to go ahead and add the logic to the button’s Click event handler. This method first checks to make sure the user has entered the URL in for the target SharePoint site and has provided a name for the new SharePoint list. It then gets a reference to the current ClientContext which is common in all SharePoint 2010 client object model code. Once it has a context, it then queues up a reference to the current site & list collection for the target SharePoint site. With the references established, but keeping in mind these objects are not yet hydrated (the client object model has not gone back to the SharePoint site yet… it’s waiting for the ExecuteQueryAsync() method to be called), the event handler then creates an instance of the ListCreationInformation object which is used to specify the values of a list to be created. Finally, it then sends all the queued commands to the SharePoint site and wires up the two callbacks that were defined earlier:

private void button1\_Click(object sender, RoutedEventArgs e)

{

// make sure a site & list name were entered

if ( string.IsNullOrEmpty(SharePointSiteTextBox.Text.Trim()) ||

string.IsNullOrEmpty(NewListNameTextBox.Text.Trim()) ) {

MessageBox.Show("SharePoint site URL & list name are required.");

return;

}

// get a context, reference to site & list collection

ClientContext context = new ClientContext(SharePointSiteTextBox.Text);

Web site = context.Web;

ListCollection lists = site.Lists;

// create a new list

ListCreationInformation newAnnouncementList = new ListCreationInformation();

newAnnouncementList.Title = NewListNameTextBox.Text;

newAnnouncementList.TemplateType = (int)ListTemplateType.Announcements;

// add the list to the collection of lists

List newList = lists.Add(newAnnouncementList);

// load the list and execute the query, registring the success & fail handlers

context.Load(newList);

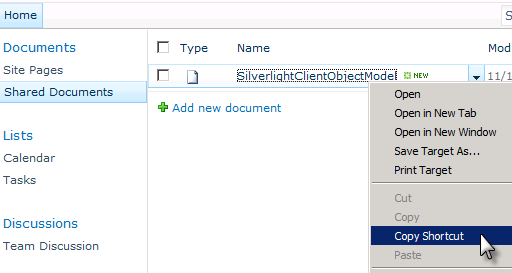
context.ExecuteQueryAsync(OnSuccess, OnFail);

}

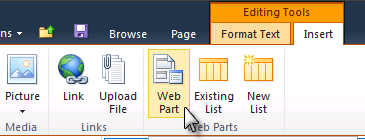
1. Test everything by right-clicking the project name in the **Solution Explorer** tool window and selecting **Build**. If there are code errors, address those.
2. Now it’s time to test the Silverlight application! Open a browser and navigate to the **http://intranet.wingtip.com/sites/ClientData** site.

**Note:** Silverlight applications can be deployed in many ways… they can be uploaded to libraries, placed in the /\_layouts folder or even embedded as a resource within an assembly. This exercise simply walks you through one of these options.

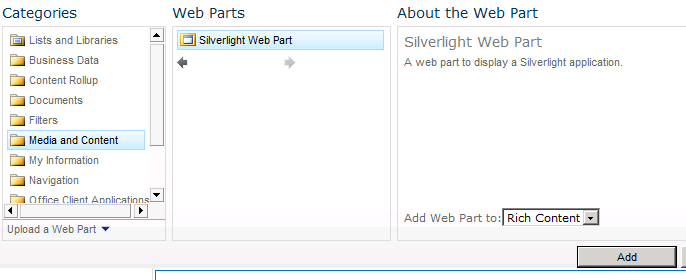
1. First you need to upload your Silverlight application. Click on the **Shared Documents** library and upload the XAP file that you created with Visual Studio. This will be found in the \bin\Debug folder of the project.
2. Now get the URL of the XAP file by right clicking it and selecting **Copy Shortcut**:



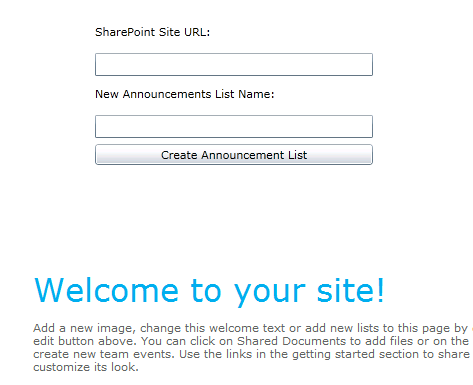
1. Using the ribbon, select **Page** **»** **Edit**.
2. Using the ribbon, select **Insert** **»** **Web Part**:



1. From the category **Media and Content**, select the OOTB **Silverlight Web Part** and add it to the **Rich Content** area:



1. The Silverlight Web Part will ask for the location of the XAP file. Paste in the URL of the XAP file you acquired from a previous step. The rich content area on a page may display an error while still in edit mode… this is not unexpected. Simply save the page by selecting the **Page** tab and then the **Save** button on the ribbon. When the page refreshes, it should render the Silverlight application you created.



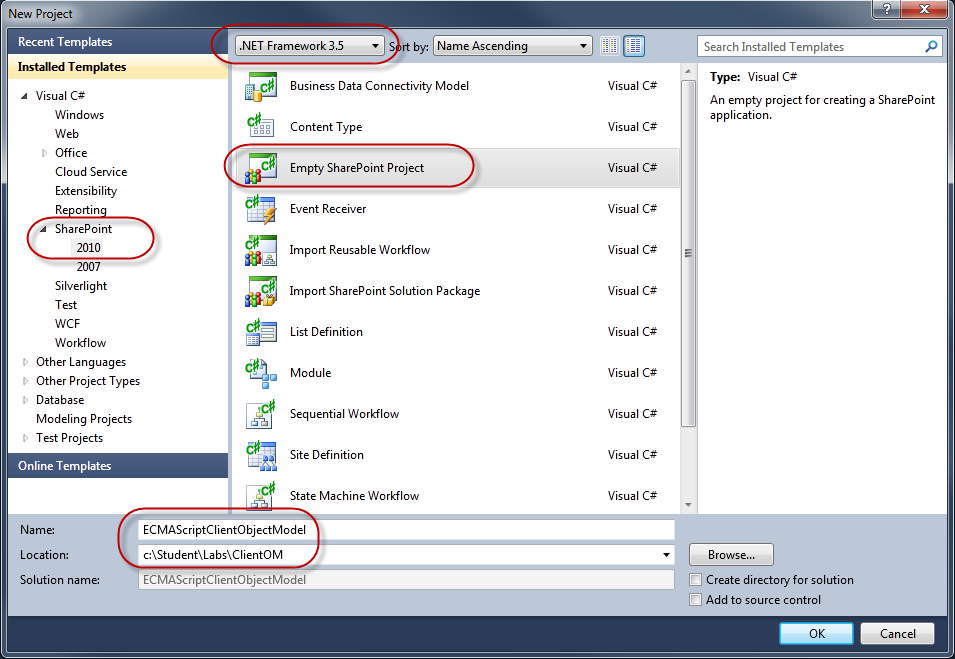
1. Enter the URL of your site (**http://intranet.wingtip.com/sites/ClientData**) and the desired name of the new list. Then click the **Create Announcement List** button. After a few seconds you should see a message stating the list was successfully created! To be sure, click the **View** **All Site Content** link in the **Quick Launch** navigation to see a list of all content on the current site, especially your new list.

In this exercise you created a Silverlight application that leverages the Silverlight Client Object Model to create a new SharePoint list. This Silverlight application is hosted in the OOTB Silverlight Web Part.

### Exercise 4: Creating Lists with the ECMAScript Client Object Model

In this exercise you will create a new SharePoint 2010 application page that uses the ECMAScript Client Object Model to create a new SharePoint announcement list.

1. Launch Visual Studio 2010 and create a new **Empty SharePoint Project** project using the **File » New » Project** menu. Make sure you have targeted the **.NET Framework 3.5** version and give it a name of **ECMAScriptClientObjectModel**.

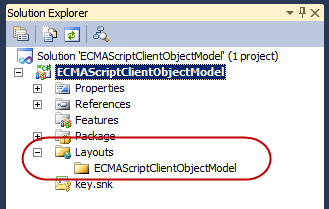


1. Complete the wizard that appears using the following information.

**Debugging site:** http://intranet.wingtip.com/sites/ClientData

**Deploy as a farm solution**: selected

1. The first step is to create the mapped folder that points to the site’s **/\_layouts** directory. Right-click the project and select **Add » SharePoint “Layouts” Mapped Folder**. Visual Studio 2010 will add a new folder structure to the project consisting of a mapped folder (“Layouts”) and a subfolder that corresponds to the project name as shown in the following figure.



1. Next, add the application page by right-clicking the mapped subfolder **ECMAScriptClientObjectModel** and select **Add » New Item**. Pick **Application Page** from the list of available templates under the **SharePoint » 2010** category. Set the name to **ListCreator.aspx** and click **Add**.
2. With the page created, create a button that will be used to trigger the ECMAScript Client Object Model code. Within the ASP.NET content placeholder **Main**, add the following markup to create an HTML button that will call an ECMAScript method when clicked:

<input type=”button” value=”Create List” onclick=”CreateList()” />

1. Before implementing the client side script, a reference to the SharePoint ECMAScript client object model needs to be added. Add the following server control to the ASP.NET content placeholder **PlaceHolderAdditionalPageHead** to load the ECMAScript Client Object Model library.

<SharePoint:ScriptLink Name=”sp.js” LoadAfterUI=”true” Localizeable=”false” runat=”server” />

1. Now you can add the code to create the list. Add the following client side script declaration and method signatures just after the server control added in the previous step:

<script language="ecmascript" type="text/ecmascript">

function CreateList() {

}

function OnSuccess() {

}

function OnFailure() {

}

</script>

1. Within the CreateList() method, the first two things to do are to (a) get an instance of the client context object and (b) get a reference to the current site the page is loaded in. Add the following script to the CreateList() method:

function CreateList() {

// client context

var clientContext = new SP.ClientContext.get\_current();

// load current web

var currentSite = clientContext.get\_web();

}

1. Next, create a list creator object and set the new list’s title & template type:

function CreateList() {

// client context

var clientContext = new SP.ClientContext.get\_current();

// load current web

var currentSite = clientContext.get\_web();

// create list creator object

var listCreatorInfo = new SP.ListCreationInformation();

listCreatorInfo.set\_title("ECMAScript Created Announcements");

listCreatorInfo.set\_templateType(SP.ListTemplateType.announcements);

}

1. With the list creator object created, the next step is to tell SharePoint to create the list when the command it sent across. After adding the list to the site’s list collection, hydrate the list’s default properties as upon success, the script will alert the user to the list’s title:

function CreateList() {

// client context

var clientContext = new SP.ClientContext.get\_current();

// load current web

var currentSite = clientContext.get\_web();

// create list creator object

var listCreatorInfo = new SP.ListCreationInformation();

listCreatorInfo.set\_title("ECMAScript Created Announcements");

listCreatorInfo.set\_templateType(SP.ListTemplateType.announcements);

// get all the lists in the web

this.newList = currentSite.get\_lists().add(listCreatorInfo);

clientContext.load(newList);

}

1. The last step is to send all the queued commands to SharePoint. Doing this in the ECMAScript Client Object Model requires passing in two callbacks, one for success and one for failure, as all server calls are made asynchronously:

function CreateList() {

// client context

var clientContext = new SP.ClientContext.get\_current();

// load current web

var currentSite = clientContext.get\_web();

// create list creator object

var listCreatorInfo = new SP.ListCreationInformation();

listCreatorInfo.set\_title("ECMAScript Created Announcements");

listCreatorInfo.set\_templateType(SP.ListTemplateType.announcements);

// get all the lists in the web

this.newList = currentSite.get\_lists().add(listCreatorInfo);

clientContext.load(newList);

// run the queued commands

clientContext.executeQueryAsync(

Function.createDelegate(this, this.onSuccess),

Function.createDelegate(this, this.onFailure));

}

1. Finally, implement both callbacks by adding simple script alert boxes to show if the script succeeded or failed:

function onSuccess() {

alert(this.newList.get\_title() + " created.");

}

function onFailure() {

alert("Request failed.");

}

1. With everything complete, right-click the application page **ListCreator.aspx** and select **Set as Startup Item**. This tells Visual Studio 2010 to launch a browser and navigate to the page when launching the debugger. Now press **[F5]** to build, package and deploy the project. This will automatically launch a browser window.
2. When the browser loads, click the button **Create List** on the page. After a few seconds an alert box should display a message that the list was created. In addition, click **Site Actions » View All Site Content** and scroll down to see the new **ECMAScript Created Announcements** list has been created!

In this exercise you created a new SharePoint 2010 application page that used the ECMAScript Client Object Model to create a new SharePoint announcement list.