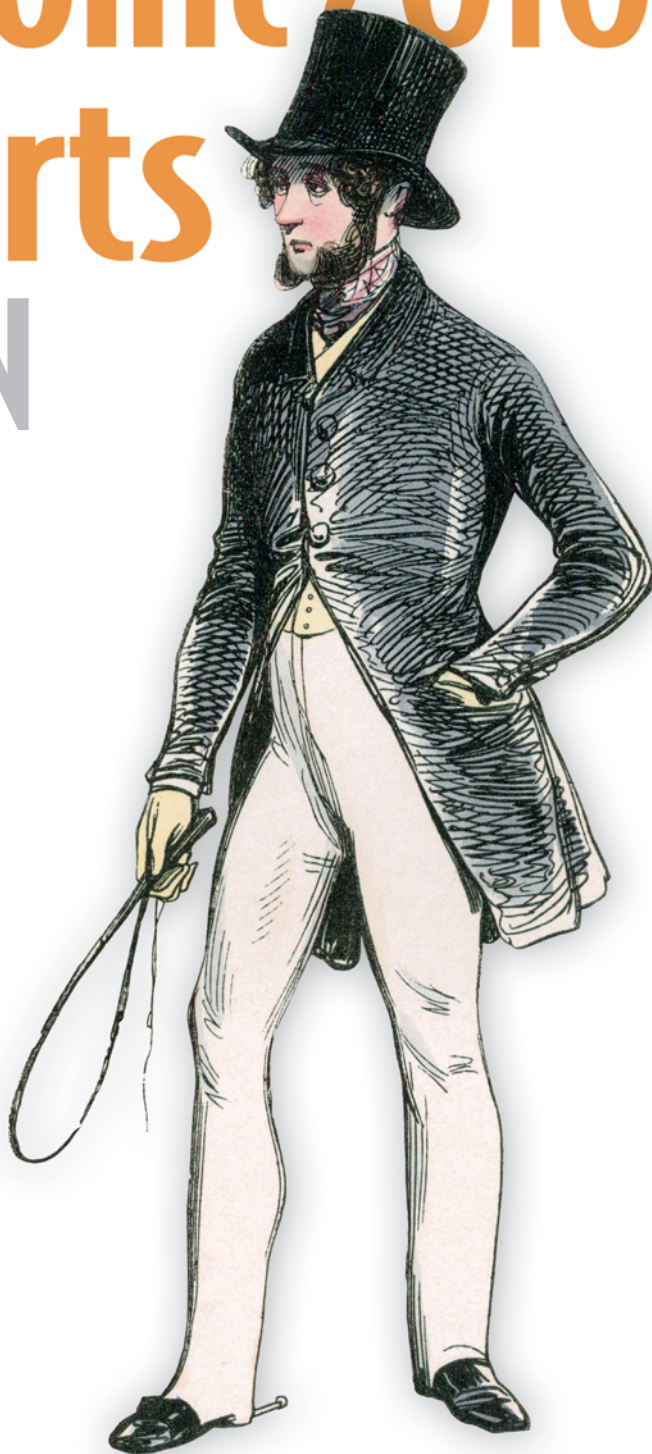


SharePoint 2010 Web Parts IN ACTION

Wictor Wilén

SAMPLE CHAPTER





***SharePoint 2010
Webparts in Action***

Wictor Wilén

Chapter 11

Copyright 2011 Manning Publications

brief contents

PART 1 INTRODUCING SHAREPOINT 2010 WEB PARTS.....1

- 1 ■ Introducing SharePoint 2010 Web Parts 3
- 2 ■ Using and configuring Web Parts in SharePoint 2010 24

PART 2 DEVELOPING SHAREPOINT 2010 WEB PARTS.....53

- 3 ■ Building Web Parts with Visual Studio 2010 55
- 4 ■ Building the user interface 79
- 5 ■ Making Web Parts customizable 110
- 6 ■ Web Part resources and localization 148
- 7 ■ Packaging, deployment, and security 171
- 8 ■ Tools for troubleshooting and logging 201
- 9 ■ Programming and caching for performance 221
- 10 ■ Dynamic interfaces in Web Parts 242
- 11 ■ The Client Object Model and Silverlight Web Parts 274
- 12 ■ Making Web Parts mobile 291
- 13 ■ Design patterns and testability 309

PART 3 DASHBOARDS AND CONNECTIONS333

- 14 ■ Connecting Web Parts 335
- 15 ■ Building pages and dashboards 359

11

The Client Object Model and Silverlight Web Parts

This chapter covers

- The Client Object Model and Web Parts
- The Silverlight Web Part
- Custom Silverlight Web Parts

Browsers have become increasingly more powerful, allowing developers to build rich user interfaces with JavaScript and plug-ins such as Silverlight and Flash. The combination of rich clients and web services enables applications to take some load off the servers and even perform tasks not possible on the servers. A benefit for power users is the ability to add custom functionality to applications without access to the server. JavaScript and Silverlight in combination with SharePoint sandboxed solutions is the way to go when you have your SharePoint hosted or if you can't install solutions directly onto the server in any other way.

In previous SharePoint versions, external applications had the option of interacting with SharePoint using web services only. These web services were poorly documented, loosely typed, and hard to work with. A new concept called the *Client Object Model* lets external applications work with SharePoint 2010 using a client

runtime and API. The Client Object Model doesn't replace the old web services (which still remain) but makes it easier and more efficient to work with the SharePoint object model remotely.

Silverlight is a technology that Microsoft is pushing hard, and SharePoint comes with a Silverlight Web Part out of the box. Silverlight applications can make the user interface more interactive and rich, and it allows developers to be more creative when building applications. This chapter shows you how to work with the Client Object Model and Silverlight to create a rich experience for your end users.

11.1 The Client Object Model

One of the most innovative new features of SharePoint 2010 is the Client Object Model. It's a completely new approach to working with SharePoint for remote clients. Previously, remote clients used SharePoint web services, a limited and tedious task for developers. The Client Object Model offers a client-side API that comes in three flavors: one for managed .NET code, one for Silverlight applications, and a JavaScript variant.

You can use the Client Object Model to accomplish many SharePoint tasks, but for Web Parts, its use is limited. The client-side API for Web Parts lets you add, move, and delete Web Parts on a page and change some default properties, such as the title. There's no way to access the custom properties or methods on a Web Part. If you need to do this kind of customization with remote clients, you have to add your own remote API to SharePoint. Other parts of the Client Object Model allow you to access larger portions of SharePoint's features and APIs.

Web Parts built using the SharePoint Web Part implementation have had their own client-side JavaScript library called Web Part Page Services Components (WPSC), which enabled client-side Web Part interactions. This JavaScript library still exists in SharePoint for backward compatibility only with SharePoint 2003 and SharePoint 2007.

11.1.1 What is the ECMAScript Client Object Model?

The Client Object Model is Microsoft's response to user requests for more and better web services in SharePoint. Instead of building new web services, Microsoft built a whole new API using a subset of the standard SharePoint API. The Client Object Model is an API that's supposed to run where access to the server API isn't possible. The client API is similar to the server API, but it's limited to using objects from the site collection (SPSite) level and down—which means that it's easier to learn and understand the model than to learn a set of new web services.

The Client Object Model consists of three APIs that are very similar:

- A managed client API for Microsoft .NET Framework code
- A Silverlight 2.0+ client API for Silverlight applications
- An ECMAScript client API for web browser-based JavaScript applications

NOTE ECMAScript (ECMA-262) is the standardization (ECMA International) of JavaScript.

The Client Object Model uses a client runtime proxy that communicates with a Windows Communication Foundation (WCF) service. The service is located at `/_vti_bin/client.svc`. To communicate, the client proxy sends batched XML commands and the WCF services respond with a result formatted using JavaScript Object Notation (JSON). Because all commands are batched together and the JSON notation is a compact form, the Client Object Model bandwidth use is kept to a minimum.

Creating Client Object Model applications using a managed .NET client requires references to `Microsoft.SharePoint.Client.dll` and the `Microsoft.SharePoint.Client.Runtime.dll`. These files are located in the `{SharePoint Root}\ISAPI` folder. The Silverlight API uses two other assemblies as reference: `Microsoft.SharePoint.Client.Silverlight.dll` and `Microsoft.SharePoint.Client.Silverlight.Runtime.dll`. These files are located in `{SharePoint Root}\TEMPLATE\LAYOUTS\ClientBin`. To use the ECMAScript Client Object Model, you need to copy or reference the following JavaScript files that are located in the `{SharePoint Root}\TEMPLATE\LAYOUTS` folder:

- `SP.js`
- `SP.Core.js`
- `SP.Ribbon.js`
- `SP.Runtime.js`

Each of those JavaScript files has a debug version that includes IntelliSense support. Use the debug files when you've enabled debugging in the web application (using the `web.config` file as described earlier in this book). The debug and release versions differ in such a way that the release versions are minified JavaScript files and the debug files contain readable and debuggable code. The debug versions have the following names:

- `SP.debug.js`
- `SP.Core.debug.js`
- `SP.Ribbon.debug.js`
- `SP.Runtime.debug.js`

The JavaScript files are normally an out-of-the-box feature added to all pages using the SharePoint master pages. If you intend to create a custom web page that uses a custom master page, use the SharePoint `ScriptLink` control, described in chapter 4, to add these JavaScript files to your pages.

11.1.2 Working with the Client Object Model and Web Parts

The Client Object Model is fairly limited when it comes to working with Web Parts. Basic operations such as adding and removing Web Parts can be done as well as changing some default properties of the Web Part. There's no access to custom Web Part properties.

You can use the Client Object Model to remotely add Web Parts to pages or inspect pages for closed or hidden Web Parts. The Client Object Model API differs a bit from the server object model. All Web Part functionality is in the `Microsoft.SharePoint.Client.WebParts` namespace. Three classes are of interest to us for this discussion:

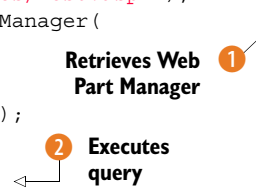
- `LimitedWebPartManager`—Imports and adds Web Parts
- `WebPartDefinition`—Provides move, remove, and change state operations
- `WebPart`—Provides information about the Web Part

ADDING A WEB PART USING THE CLIENT OBJECT MODEL

The `LimitedWebPartManager` object is used to add new Web Parts to a page by first importing a Web Parts control description XML file and then adding it to a page and zone. To create an application that adds a Web Part to a page, you create a console application using Visual Studio 2010. Be sure to set the .NET Framework to version 3.5. Then add `Microsoft.SharePoint.Client.dll` and the `Microsoft.SharePoint.Client.Runtime.dll`, found in the {SharePoint Root}\ISAPI folder, as references to the project. Listing 11.1 shows how a Web Part is loaded from a Web Parts description file and then added to a page using the Client Object Model.

Listing 11.1 Adding a Web Part to a page using the Client Object Model

```
using (ClientContext context = new ClientContext("http://server")) {
    Web web = context.Web;
    File file = web.GetFileByServerRelativeUrl("/SitePages/Test.aspx");
    LimitedWebPartManager wpMgr = file.GetLimitedWebPartManager(
        PersonalizationScope.Shared);
    WebPartDefinition webPartDef = wpMgr.ImportWebPart(
        System.IO.File.ReadAllText("Content_Editor.dwp"));
    wpMgr.AddWebPart(webPartDef.WebPart, "Left", 1);
    context.ExecuteQuery();
}
```



When working with the Client Object Model, you always start by acquiring a client context (`ClientContext`) to the site. Using this context, the `Web` object is retrieved and from that a specific page. The Client Object Model `Web` object corresponds to the server API `SPWeb` object. The `LimitedWebPartManager` object is then retrieved from the file object ❶. The Web Part is created by first importing a Web Parts control description file, exported from a page, for instance. This results in a `WebPartDefinition` object that's then added (using the `LimitedWebPartManager`) to a specific zone on the page. So far, everything is done on the client side and the server isn't yet updated. Not until it calls the `ExecuteQuery` ❷ method of the client context object is the command sent to the server. All commands are batched together and executed on the server only when the `ExecuteQuery` method is invoked. This allows you to make several additions and changes before making the changes permanent. Note that the batch jobs aren't transactional.

MODIFYING A WEB PART USING THE CLIENT OBJECT MODEL

In the Client Object Model, the only Web Part properties that you can access are the following:

- | | |
|-------------------------|--------------------------|
| ■ <code>Title</code> | ■ <code>TitleUrl</code> |
| ■ <code>Subtitle</code> | ■ <code>IsClosed</code> |
| ■ <code>Hidden</code> | ■ <code>ZoneIndex</code> |

This means that you can't do more than add a Web Part and then change its title or inspect for closed or hidden Web Parts in the pages. To change the title of a Web Part, the approach is similar to our previous example. Listing 11.2 shows how to retrieve all Web Parts on a page and then make all titles uppercase.

Listing 11.2 Modifying Web Parts using the Client Object Model

```
using (ClientContext context = new ClientContext("http://server/")) {
    Web web = context.Web;
    File file = web.GetFileByServerRelativeUrl("/SitePages/Home.aspx");
    LimitedWebPartManager wpMgr =
        file.GetLimitedWebPartManager(PersonalizationScope.Shared);

    context.Load(wpMgr.WebParts,
        webParts => webParts.Include(
            webPart => webPart.WebPart.Title));
    context.ExecuteQuery();

    foreach (WebPartDefinition webPartDef in wpMgr.WebParts) {
        WebPart webPart = webPartDef.WebPart;
        webPart.Title = webPart.Title.ToUpper();
        webPartDef.SaveWebPartChanges();
    }
    context.ExecuteQuery();
}
```

The diagram consists of four numbered callouts with arrows pointing to specific lines of code:

- 1 Loads all Web Parts**: Points to the `context.Load(wpMgr.WebParts, ...)` line.
- 2 Loads only Title property**: Points to the `webPart => webPart.WebPart.Title` line inside the `Include` method.
- 3 Saves changes**: Points to the `webPartDef.SaveWebPartChanges();` line inside the `foreach` loop.
- 4 Persists changes**: Points to the `context.ExecuteQuery();` line at the end of the `foreach` loop.

Just as in the previous sample, the client context is created first and then the file and Limited Web Part Manager are retrieved. In this case you're only interested in the titles of the Web Parts, so you tell the Client Object Model to load all Web Parts on the page **1** and then only include the `Title` properties of those objects **2**. Then the `ExecuteQuery` method is called so that it fetches the information from the server of the Web Parts to work with. All Web Parts are iterated over and their title is changed to uppercase and saved **3** before calling `ExecuteQuery` **4** once again to persist the changes.

This sample required two calls to the `ExecuteQuery`: one to load the Web Parts and one to save the titles. Requesting the titles of the Web Parts minimizes the network traffic and potentially speeds up the application.

What about Web Part Page Services Components in SharePoint 2010?

The Web Part Page Services Component (WPSC) Object model is a JavaScript component that delivers client-side functionality to Web Parts and Web Part pages. It was used in previous versions of SharePoint and still exists for backward compatibility. The WPSC could make changes to and listen to events from Web Parts using JavaScript.

The WPSC is only available for SharePoint Web Parts derived from the old Web Part base class (`Microsoft.SharePoint.WebPartPages.WebPart`). Because Microsoft discourages the use of the old Web Part implementation, they don't recommend you use WPSC either. Instead, use the Client Object Model, ASP.NET Ajax, or other dynamic JavaScript libraries such as jQuery.

11.2 Silverlight Web Parts

Silverlight is a rich client technology that allows you to do even more than is possible with just HTML, JavaScript, and the Client Object Model. SharePoint 2010 supports Silverlight out of the box. It's used sparsely in the default user interface but can be enhanced with custom Silverlight applications using the Silverlight Web Parts. SharePoint Foundation has a generic Silverlight Web Part that you can use to add any Silverlight application to a Web Part page. Silverlight in combination with the Client Object Model is a great way to develop SharePoint solutions. It allows developers to build their applications and package them as sandboxed solutions, which means that they can install them without access to the SharePoint servers. This is great both for custom on-premise solutions as well as for hosted SharePoint solutions such as Office 365 and SharePoint Online 2010. Silverlight allows developers to be more creative with the user interface and introduce new and richer interfaces.

In this section, I'll show you how to create a Silverlight application using the Silverlight SDK that uses the Client Object Model to read information from SharePoint. The Silverlight application will first be deployed using the out-of-the-box Silverlight Web Part and then using a custom Web Part. I'll show you how to customize the default Silverlight Web Part and use various packaging options in Visual Studio. Custom Silverlight Web Parts allow developers to lock down the ability for users to change which Silverlight application they want to use and make it easier to add preconfigured Silverlight Web Parts to the Web Part Gallery.

NOTE Silverlight isn't available for all platforms, so you have to carefully plan whether to use Silverlight. Silverlight is currently available for all major browsers on Windows and Mac OS X.

11.2.1 SharePoint Silverlight Web Part

SharePoint Foundation contains a Web Part for Silverlight applications out of the box. This Web Part is found in the *Media and Content* Web Part category, as you can see in figure 11.1. The Silverlight Web Part can be used to show custom Silverlight applications and only requires you to specify the source of the Silverlight XAP file.

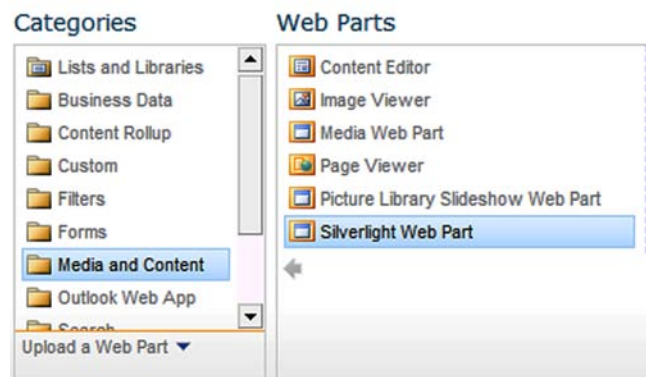


Figure 11.1 The SharePoint Silverlight Web Part is available in all SharePoint editions and can be used to add Silverlight applications to your pages. You'll find the Silverlight Web Part in the Media and Content category in the Web Part Gallery.

Once you've added the Silverlight Web Part to the page, the Web Part will ask you to enter the URL where the Silverlight XAP file is located. The URL can be a local URL in your SharePoint farm or a remote URL where the Silverlight application is hosted by someone else. You can easily edit the export mode settings of a configured Silverlight Web Part and then export it to use in your applications.

Using Silverlight in combination with SharePoint allows you to build a richer user interface that you can't achieve using standard HTML, CSS, and JavaScript. Silverlight has great support for rich media and streaming, and SharePoint Server has a configurable Silverlight-based Media Web Part out of the box.

11.2.2 Building a Silverlight SharePoint application

To illustrate the Silverlight Web Part, I'll show you how to build a simple Silverlight application that uses the Client Object Model. The application retrieves the number of tasks in a Tasks list and displays the number on the screen, as shown in figure 11.2.

Silverlight applications are built using Visual Studio 2010 with the Silverlight SDK, which is a separate download. After you've downloaded Silverlight SDK 4 and installed it on your development machine, create a new project using the Silverlight Application template. After you enter the name of your project, the New Silverlight Application wizard asks you to optionally create a website that hosts the Silverlight application and the Silverlight version to use. Because this application will be hosted in SharePoint, you don't need a website and you should select Silverlight Version 4. Click OK to create your project.

NOTE To create Silverlight applications you need the Microsoft Silverlight 3/4 Tools for Visual Studio. This is a free download at <http://msdn.microsoft.com/Silverlight/>.

Visual Studio will create a solution containing two XAML (Extensible Application Markup Language) files: one named App.xaml, which is the actual application, and a second file named MainPage.xaml, which is the Silverlight user interface/control. Double-click on the MainPage.xaml file to open the Silverlight designer. Then use the Toolbox in Visual Studio to drag and drop a new Label control onto the Silverlight control. Next, use the Properties window to set the value of the Content property to Task Count. Add another label and change the Content property to ??, and then set its text size to 24.

When this is done, resize the whole control so that the two labels fit nicely. The Silverlight application should look something like that shown in figure 11.3.



Figure 11.2 The Silverlight Web Part configured to use the Task Count Silverlight application, which shows the number of tasks in the Tasks list of the current site.

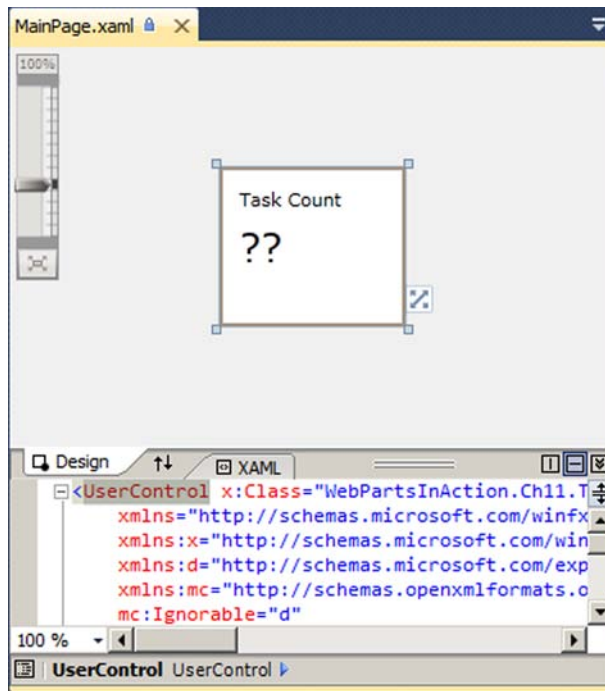


Figure 11.3 The Silverlight XAML designer in Visual Studio 2010 makes the design of the Silverlight application easy.

The Silverlight designer in Visual Studio creates the XAML code, which you can see by switching to the XAML view in Visual Studio. You can edit the XAML code directly instead of using the designer, if you prefer.

Now it's time to write the code that retrieves the number of tasks from the current Tasks list in the site. You'll use the Client Object Model for Silverlight, and for that you need to add the Silverlight Client Object Model reference assemblies. These assemblies are located in {SharePoint Root}\TEMPLATES\LAYOUTS\ClientBin\ and are called `Microsoft.SharePoint.Client.Silverlight.dll` and `Microsoft.SharePoint.Client.Silverlight.Runtime.dll`, respectively. Add these two files as references in the Silverlight project.

This application should load the tasks as soon as the control is loaded, so you need to add a code-behind file to the Silverlight control. On the Designer surface, click on the gray area surrounding the control and then press F4, or open the Properties window. The Properties window has two tabs: Properties and Events. Select the Events tab and locate the Loaded event. Double-click on the event name to create the stub code for that event in the code-behind file. Visual Studio will automatically give it the name `UserControl_Loaded`. Once that is done, the code-behind file will open and the cursor will be positioned in the new event method. You'll use this method to load the tasks from the Tasks list using the Client Object Model. Listing 11.3 shows how to implement the retrieval of the tasks.

Listing 11.3 Silverlight code using the Client Object Model to retrieve tasks

```

List tasks;

private void UserControl_Loaded(object sender, RoutedEventArgs e) {
    using (ClientContext context = ClientContext.Current) {
        tasks = context.Web.Lists.GetByTitle("Tasks");
        context.Load(tasks);

        context.ExecuteQueryAsync(
            requestSucceeded,
            requestFailed);
    }
}

private void requestSucceeded(object sender,
    ClientRequestSucceededEventArgs e) {
    Dispatcher.BeginInvoke( () => {
        label2.Content = tasks.ItemCount.ToString();
    });
}

private void requestFailed(object sender,
    ClientRequestFailedEventArgs e) {
    Dispatcher.BeginInvoke( () => {
        label2.Content = "??";
    });
}

```

1 Defines shared variable

2 Executes asynchronous method

3 Updates user interface

This code requires that the namespace `Microsoft.SharePoint.Client` be added as a using statement so you can access the Client Object Model API. All queries to the Client Object Model are done asynchronously in Silverlight, in contrast to the .NET sample you looked at earlier. This forces you to declare the `tasks` variable, representing the Tasks list, to be declared as a class variable **1**. In the `UserControl_Loaded` method, the current client context is retrieved and then the Tasks list is fetched using the title of the list. When the list has been loaded into the context, it queries asynchronously for the information. The `ExecuteQueryAsync` **2** method takes two event handlers as parameters. The first one points to a method that's going to be executed if the query is successful, and the second if the query fails. The succeeded event handler, `requestSucceeded`, will update the content of the label in the control with the count of tasks in the Tasks list **3**. The failure event handler, `requestFailed`, just writes two question marks to indicate that it couldn't retrieve the number of tasks.

The two event handler methods use `Dispatcher.BeginInvoke` to execute delegates that update the Label controls. The delegates are executed asynchronously on the UI thread. Trying to update the user interface controls directly from the succeeded or failed event handlers would result in an exception because these methods execute on a separate thread.

NOTE Silverlight is a multithreaded application and the only thread that can update the user interface is the UI thread.

The resulting Silverlight Application Package has the .xap extension (pronounced ZAP). You'll reference this file in the Silverlight Web Part. Because you didn't create a web project to host the Silverlight application, upload the XAP file to SharePoint, preferably to a document library. Open up a SharePoint site and browse to Shared Documents and choose to upload a new document. Select the XAP file from your project (located in the bin\debug folder if you're using a debug build) and then upload the file. Right-click the newly uploaded file and select Copy Shortcut to copy the URL to the Silverlight application.

Create a new Web Part page or Wiki page and add the Silverlight Web Part to the page. When the Web Part asks for the URL to the Silverlight application package, paste the URL you just copied from the document library and click OK. The Silverlight Web Part will load your Silverlight application and display it, as shown in figure 11.2.

TIP If you want to debug a Silverlight application using Visual Studio while the application is running in SharePoint, you must enable Silverlight debugging. To do that, open the project's Properties window and click the SharePoint tab (on the left side). Make sure that the Enable Silverlight Debugging check box is checked. When that option is enabled, you won't be able to debug JavaScript using Visual Studio.

11.2.3 Input parameters to the Silverlight Web Part

There are often situations where you need to configure the Silverlight applications with different input parameters. Properties are passed to the Silverlight application through a text property called Custom Initialization Parameters, which can be found under the Other Settings Web Part property category, as shown in figure 11.4.

The Custom Initialization Parameters property consists of a comma-separated list of keys and values that will be sent to the Silverlight application. You can retrieve these parameters from the Silverlight application using a dictionary object. This dictionary object is accessible in the Application_Startup event of the Silverlight applications. To use the parameters or send them to the Silverlight control, you need to modify that event. The following code demonstrates how to extract the dictionary from the startup event arguments and then retrieve the properties:

```
private void Application_Startup(object sender, StartupEventArgs e) {
    IDictionary<string, string> parms = e.InitParams;
    string listId = parms.ContainsKey("ListId")
        ? parms["ListId"].ToString() : null;
}
```

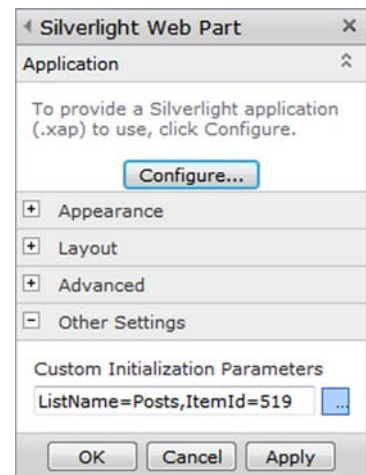


Figure 11.4 The editor pane of the Silverlight Web Part allows you to set the Silverlight application using the Configure button and supply custom initialization parameters to the application using the text field.

```

int itemId = parms.ContainsKey("ItemId")
    ? int.Parse(parms["ItemId"]) : 0;

this.RootVisual = new MainPage() {
    ListId = listId,
    ItemId = itemId
};
}

```

In this sample, the application expects that two parameters will be passed with the Custom Initialization Properties, as shown in figure 11.4: ListId and ItemId. The properties dictionary is retrieved from the StartupEventArgs object passed to the event. Each initialization property is stored in this dictionary and is retrieved and stored in local variables. These variables are then sent to the MainPage object. This object represents the Silverlight application control, which has been extended with two public properties that accept the values from the initialization properties.

11.2.4 Packaging the Silverlight Web Part

To improve the experience for the end user, you can bundle the Silverlight application into a SharePoint solution package and preconfigure the Silverlight Web Part. First, add a new Empty SharePoint project to your current solution. Choose to deploy this solution as a sandboxed solution.

In the SharePoint project, add a new Module SharePoint project item. Once you create the project item, remove the sample.txt file that's automatically added. Right-click the project item, select to add an existing item, and then browse to your XAP file and select it to add it to the project. Open the elements.xml file and edit it so that it looks like the following listing.

Listing 11.4 Elements.xml file provisioning the Silverlight application package

```

<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module
    Name="TaskCountApp"
    Url="Shared Documents">
    <File
      Path="TaskCountApp\WebPartsInAction.Ch11.TaskCount.xap"
      Url="TaskCountApp/WebPartsInAction.Ch11.TaskCount.xap"
      Type="GhostableInLibrary" />
    </Module>
  </Elements>

```

The Url attribute of the Module element must be manually edited in the elements.xml file to correspond to the name of the library where you want to deploy your XAP file. Doing so deploys the files to the Shared Documents library instead of a custom folder. The File element is automatically added when you add a new or an existing item to this project item. Make sure that you add and set the Type attribute to GhostableInLibrary,

because you're deploying a file to a document library. `GhostableInLibrary` allows SharePoint to keep a reference to the file in the file system instead of in the content database, which improves performance.

When you build and deploy this solution, it will add the Silverlight XAP file. That file can be used in any Silverlight Web Part by referencing the XAP file. The problem is that whenever the Silverlight application is changed in your Visual Studio solution or when the build target is changed, you need to read the XAP file to the module in the SharePoint project from the output of the Silverlight project.

A better approach is to use a *project output reference* on the SharePoint Project item. First remove the XAP file from the SharePoint project; you won't use this manually added file anymore. Then, right-click on the Module item (or select it and press F4) and click the ellipsis icon on the Project Output References property. The Project Output References dialog box opens. Click the Add button to add a new reference. In the Project Name property of the new reference item, select the project name for the Silverlight project and as Deployment Type select `ElementFile`, as you can see in figure 11.5.

After adding the Project Output Reference, the `elements.xml` file is updated with a `File` element looking exactly the same as you wrote. However, this time the XAP file isn't added to the project; instead it's added during the packaging processes of the SharePoint solution package. In addition, the XAP file will always be the latest version and thus will use the correct build version. Try it by building and deploying the solution, then navigate back to the page where you added the Silverlight Web Part. Edit the Web Part and its properties so that the path to the Silverlight application points to the newly deployed XAP file.

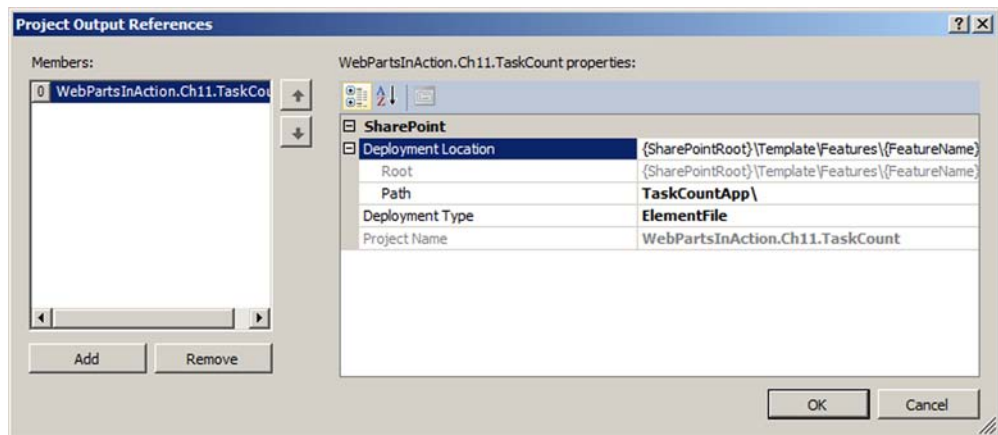


Figure 11.5 To include output files from other Visual Studio projects, use the Project Output References dialog box. Here you can add the output from a Silverlight project into a SharePoint project and add the XAP file as an `ElementFile`.

11.2.5 Preconfiguring the Silverlight Web Part

Not all users are comfortable selecting XAP files and editing the parameters in the Silverlight Web Part. To make it easier for end users to use the Silverlight Web Part, preconfigure the Silverlight Web Part and deploy it to the Web Part Gallery.

You add a preconfigured Web Part to the Visual Studio solution by adding a second Module item to the SharePoint project; don't forget to remove the sample.txt file from the new project item. To add the Web Parts control description file, either write one or even better, export the Silverlight Web Part. To export the .webpart file, you first need to enable export on the Silverlight Web Part. By default, the export mode of the Silverlight Web Part is set to Do Not Allow. Edit the Web Part and change the Export Mode setting in the Advanced category to Export All Data, and then save the Web Part. Use the Web Part options menu to export the Web Part to your local drive. The exported Web Part will contain all settings you configured prior to exporting it.

Go back to Visual Studio and select to add an existing item to the Web Part Module you just created. Add the exported .webpart file and give it a meaningful name, such as TaskCountWebPart.webpart, instead of using the default filename, as shown in figure 11.6.

The File element is automatically added to the elements.xml file. Open the .webpart file, locate the Title property, and change the default value; then change the ExportMode property so that it can't be exported:

```
<property
  name="Title"
  type="string">
  Task Count Web Part
</property>
<property
  name="ExportMode"
  type="exportmode">
  None
</property>
```

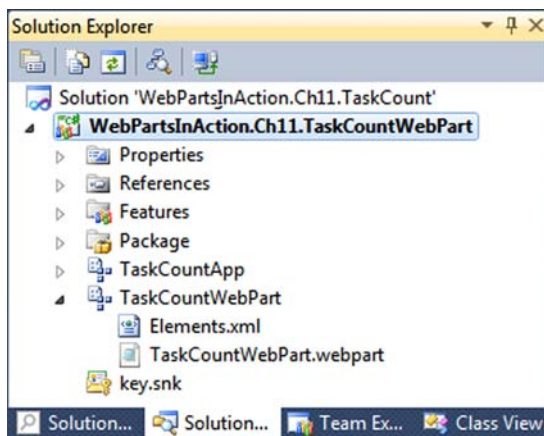


Figure 11.6 Preconfigured Web Parts can be added to a Visual Studio solution using an Empty SharePoint Project Item and then be deployed to the farm.

Finally, make sure that the .webpart file is deployed to the Web Part Gallery. Edit the elements.xml file so that it looks like listing 11.5.

Listing 11.5 The element manifest that provisions the Silverlight Web Part

```
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module
    Name="TaskCountWebPart "
    List="113"
    Url="_catalogs/wp">
    <File
      Path="TaskCountWebPart\Silverlight_Web_Part.webpart"
      Url="Silverlight_Web_Part.webpart"
      Type="GhostableInLibrary">
      <Property
        Name="Group"
        Value="Web Parts in Action"/>
    </File>
  </Module>
</Elements>
```

The first thing to edit is the Module element; the List and Url attributes have to be configured so that the Module is targeting the Web Part Gallery. Second, add the Type attribute to the File element and remove the folder name from the Url attribute. Finally, add a Property element to the File element and configure the element to set the Group property of the Web Part.

Before deploying the solution, make sure that the Feature of the SharePoint project has its Scope set to Site. Because this Feature is deploying the Web Part to the Web Part Gallery, you can't use the default Scope value (which is Web). When the solution is deployed, the Task Count Web Part can be found in the Web Part Gallery under the category you specified in the element manifest. The Web Part is now preconfigured with the Silverlight application that's deployed with the Feature.

11.2.6 Custom Silverlight Web Part

The previous example allowed you to bundle the Silverlight application with a preconfigured Silverlight Web Part, which made it easier for end users to use the Silverlight application. Because you're using the out-of-the-box Silverlight Web Part, users can still modify the Web Part's properties, such as the initialization parameters, and modify which Silverlight application to use. There are situations where you don't want your users to configure the Silverlight application and therefore you'd like to remove the ability to configure the source to the Silverlight XAP file or the initialization parameters. Unfortunately, the Silverlight Web Part is sealed so you can't subclass it and remove the custom Editor Part that provides the ability to configure the XAP URL. So, you have to create your own Silverlight Web Part from scratch. This also gives you benefits such as creating a better experience for the administrators using Web Part properties, instead of using the Custom Initialization Parameters.

Before building the new Silverlight Web Part, add an error check in the Silverlight application, which checks whether the `ClientContext` is null before proceeding. The check is implemented in the `UserControl_Loaded` method so that it looks like this:

```
private void UserControl_Loaded(object sender, RoutedEventArgs e) {
    using (ClientContext context = ClientContext.Current) {
        if (context == null) {
            Dispatcher.BeginInvoke(() => { label2.Content = "--"; });
            return;
        }
        tasks = context.Web.Lists.GetByTitle("Tasks");
        context.Load(tasks);
        context.ExecuteQueryAsync(requestSucceeded, requestFailed);
    }
}
```

Checks if `ClientContext` is available 1

When `ClientContext` is equal to null, you write two dashes in the Label control and exit the method. ❶ `ClientContext` will be set to null if the Silverlight application can't acquire the client context—for instance, when you run the application outside of SharePoint.

You create the new Silverlight Web Part by adding a new Web Part project item to the solution's SharePoint project. This Web Part will add the necessary HTML code and reference the XAP file without allowing users to change the Silverlight application. Listing 11.6 shows the `CreateChildControls` method of the new Silverlight Web Part.

Listing 11.6 A custom Silverlight Web Part

```
protected override void CreateChildControls() {
    Panel ctrlHost = new Panel() {
        ID = "silverlightControlHost"
    };

    var objectControl = new HtmlGenericControl("object");
    objectControl.Attributes.Add("data",
        "data:application/x-silverlight-2");
    objectControl.Attributes.Add("type", "application/x-silverlight-2");
    objectControl.Attributes.Add("width", "100%");
    objectControl.Attributes.Add("height", "100%");

    objectControl.Controls.Add(createParam("source",
        SPContext.Current.Site.Url +
        "/Shared%20Documents/TaskCountApp/" +
        "WebPartsInAction.Ch11.TaskCount.xap"));

    objectControl.Controls.Add(createParam("background",
        "white"));

    objectControl.Controls.Add(createParam("minRuntimeVersion",
        "4.0.50401.0"));

    objectControl.Controls.Add(new LiteralControl("Install Silverlight!"));
    ctrlHost.Controls.Add(objectControl);
}
```

```

        this.Controls.Add(ctrlHost);
    }

    private static HtmlGenericControl createParam(string name, string value) {
        var param = new HtmlGenericControl("param");
        param.Attributes.Add("name", name);
        param.Attributes.Add("value", value);
        return param;
    }

```

The `CreateChildControls` method creates a new `Panel` object that will host the Silverlight application. Then an object element is created, using the `HtmlGenericControl` that represents the Silverlight application. This object needs a set of parameters that are defined with `param` elements. The `param` elements are created using the `createParam` helper method. The source parameter references the Silverlight XAP file and in this case is hardcoded.

When you build and deploy the project into SharePoint, you can add the new Silverlight Web Part to a page. There's no need to specify the XAP file, and it can't be modified. The Web Part will look like the one shown in figure 11.7. As you can see, the Silverlight application writes two dashes in the control. That indicates that it can't acquire the client context from within the Silverlight application. In a moment you'll learn how to overcome this issue.

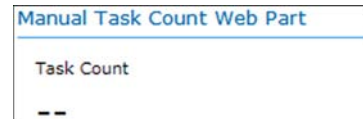


Figure 11.7 By default Silverlight can't use the Client Object Model in custom Web Parts—they need to be initialized with the proper parameters.

Using this method, you can add custom Silverlight applications as long as they aren't using the Client Object Model. To be able to use the Silverlight Client Object Model, the client runtime has to be initialized and this isn't done in our custom Silverlight Web Part. In the next section you'll learn how to initialize the client context so that you can build custom Silverlight Web Parts that can interact with SharePoint.

11.2.7 Enabling custom Silverlight Web Part interaction with SharePoint

So that your custom Silverlight Web Part can use the Client Object Model and get the current client context, you need to initialize the Client Object Model client runtime when instantiating the Silverlight application. The SharePoint Silverlight Web Part derives from an abstract class called `ClientApplicationWebPartBase` that exists in the `Microsoft.SharePoint.WebPartPages` namespace. This abstract class contains the logic necessary to send initialization information to the Silverlight application.

The `ClientApplicationWebPartBase` class can't be used in sandboxed solutions, so the first thing you need to do is change your SharePoint project to be deployed as a full-trust solution. Do so by setting the `Sandboxed Solution` property to `False` in the project's Properties window. Then modify the declaration of the custom Silverlight Web Part class so it looks like this:

```
public class ManualTaskCountWebPart : ClientApplicationWebPartBase
```

Next, add a new param element to the Silverlight object element in the `CreateChildControls` method. This param element is called `initParams` and is a string containing the necessary parameters that the Client Object Model needs to initialize. These parameters may be the current URL or the Form Digest token and timeout. The parameters are generated by the `GetInitParams` method of the `ClientApplicationWebPartBase`; here's the code you need to add to your Web Part:

```
objectControl.Controls.Add(  
    createParam(  
        "initParams",  
        GetInitParams()));
```

Now when you deploy this Web Part into SharePoint, you can use it as expected. The Silverlight application can now get the client context and the number of tasks in the Tasks list because the Client Object Model runtime is correctly initialized. There's also no way for the end user to change the Silverlight application that the Web Part uses.

The Silverlight Application XAP file is deployed to a document library, and users with necessary permissions on that library can change the Silverlight application. Deploy the XAP file to the Layouts folder so users can't change it.

TIP If you want to learn more about the Form Digest token and security validation in SharePoint, read the following article at MSDN: <http://msdn.microsoft.com/library/ms472879.aspx>.

11.3 Summary

The Client Object Model and Silverlight allow you to create rich SharePoint applications. You can think out of the box when it comes to the user interface. Using Silverlight to create graphs or animations can greatly enhance the user experience for any SharePoint site.

The SharePoint 2010 sandbox has its limitations; you can't, for instance, call web services from a sandboxed Web Part. Silverlight, on the other hand, is executed on the client, can be used in a sandboxed Web Part and used to call web services to overcome the sandbox limitations.

The Client Object Model can be used in Silverlight, JavaScript, or .NET applications. The JavaScript Client Object Model is something that you'll likely use when building Ribbon extensions or when you need to build non-Silverlight-based sandboxed Web Parts. For administrative remote operations or when you have other ASP.NET-based applications, you can use the .NET Client Object Model. For example, if you build solutions in Windows Azure, use the .NET Client Object Model to communicate with your SharePoint farms.

SharePoint 2010 Web Parts IN ACTION

Wictor Wilén



If you look at a SharePoint application you'll find that most of its active components are Web Parts. SharePoint 2010 includes dozens of prebuilt Web Parts that you can use. It also provides an API that lets you build custom Web Parts using C# or VB.NET.

SharePoint 2010 Web Parts in Action is a comprehensive guide to deploying, customizing, and creating Web Parts. Countless examples walk you through everything from design, to development, deployment, troubleshooting, and upgrading. Because Web Parts are ASP.NET controls, you'll learn to use Visual Studio 2010 to extend existing Web Parts and to build custom components from scratch.

What's Inside

- Using and configuring Web Parts
- Web Part and portal best practices
- Custom use cases, like mobile and international apps
- Web Part design patterns

This book is written for application developers working with SharePoint 2010. Knowing Visual Studio 2010 is helpful but not required.

Wictor Wilén is a Microsoft SharePoint MVP with a decade of SharePoint experience. He lives in Sweden.

For access to the book's forum and a free ebook for owners of this book, go to manning.com/SharePoint2010WebPartsinAction

"Easy to read, informative, and detailed."

—Kunal Mittal
Sony Pictures Entertainment

"Everything you've ever wanted to know about building Web Parts."

—Waldek Mastykarz
Mavention

"One of the best reads on the topic."

—Tobias Zimmergren
TOZIT

"I thought I knew my Web Parts until I read this book!"

—Anders Rask
ProActive A/S

