Foundations of Atlas

Rapid Ajax Development with ASP.NET 2.0

Laurence Moroney

Apress®

Foundations of Atlas: Rapid Ajax Development with ASP.NET 2.0

Copyright © 2006 by Laurence Moroney

ISBN-13 (pbk): 978-1-59059-647-0

ISBN-10 (pbk): 1-59059-647-1

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Ewan Buckingham
Technical Reviewer: Keith Smith
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Kylie Johnston
Copy Edit Manager: Nicole LeClerc
Copy Editor: Kim Wimpsett
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Esterman
Compositor: Linda Weidemann, Wolf Creek Press
Proofreader: Nancy Sixsmith
Indexer: Michael Brinkman
Artist: Kinetic Publishing Services, LLC
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax 201-348-4505, e-mail `orders-ny@springer-sbm.com`, or visit `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

CHAPTER 6

Introducing Server Controls in Atlas

The first three chapters of this book gave you an overview of Ajax and Atlas and how you can use them to build web applications using this technology to restrict unnecessary postbacks and processing on your web pages, thus improving the performance and polish of your web applications. Chapters 4 and 5 introduced you to the client-side controls presented by Atlas and stepped you through many examples of how to use these controls in JavaScript and in a new XML-based script called Atlas Script.

You looked at some advanced aspects of the scripting framework, including *actions*, which are compound commands associated with an event or stimulus on a control; *behaviors*, which are automatic units of functionality that can be associated with a control, enabling things such as drag and drop; and *data binding*, which allows for controls to be wired up to each other or to themselves in order to pass data between them.

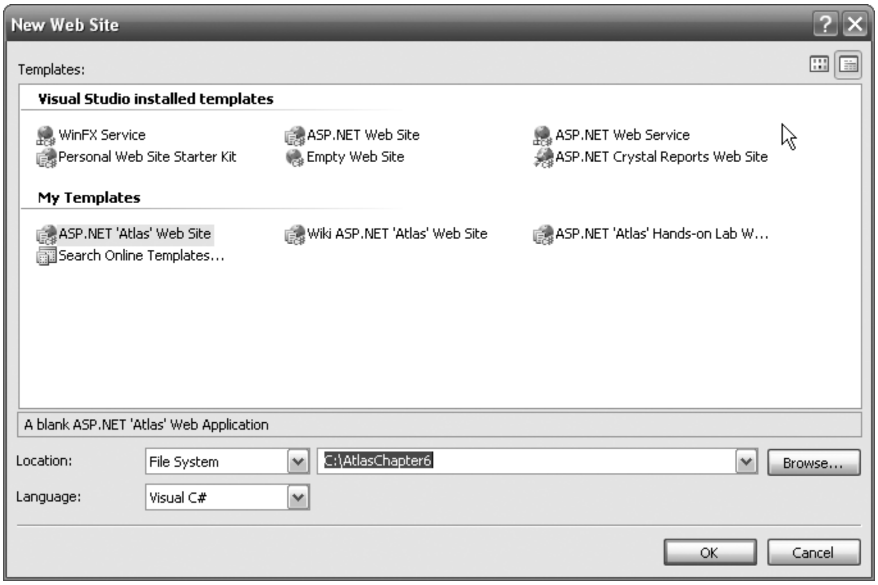
In this chapter, you will go to the other side of the action—the server—and begin exploring the various server-side controls available to you when building your Atlas applications. You have seen one of these controls, the ScriptManager control, already. In this chapter you will look at ScriptManager in more detail. In Chapter 7, you will start learning about how these controls work by navigating through some examples.

Adding the Atlas Server Controls to Visual Studio 2005

Visual Studio 2005 and ASP.NET offer some great design tools that allow you to visually construct pages. This fits in neatly with the concepts that Atlas introduces; developers can place controls on a page, and these controls generate the JavaScript that is necessary to implement the Ajax functionality. In the following sections, you'll look at how to use these controls within the integrated development environment (IDE).

Creating an Atlas Web Site

If you haven't done so already, now is a good time to take a look at Atlas projects in Visual Studio 2005. You can create a new web site by selecting File ➤ New Web Site in the Visual Studio 2005 IDE. This opens the dialog box shown in Figure 6-1. To create an Atlas-based web site, select the ASP.NET 'Atlas' Web Site template.


New Web Site
Templates:
Visual Studio installed templates
WinFX Service
ASP.NET Web Site
ASP.NET Web Service
Personal Web Site Starter Kit
Empty Web Site
ASP.NET Crystal Reports Web Site
My Templates
ASP.NET 'Atlas' Web Site
Wiki ASP.NET 'Atlas' Web Site
ASP.NET 'Atlas' Hands-on Lab W...
Search Online Templates...
Browse...
OK
Cancel

A blank ASP.NET 'Atlas' Web Application

Location: File System C:\AtlasChapter6

Language: Visual C#

Figure 6-1. *Creating a new ASP.NET Atlas web site*

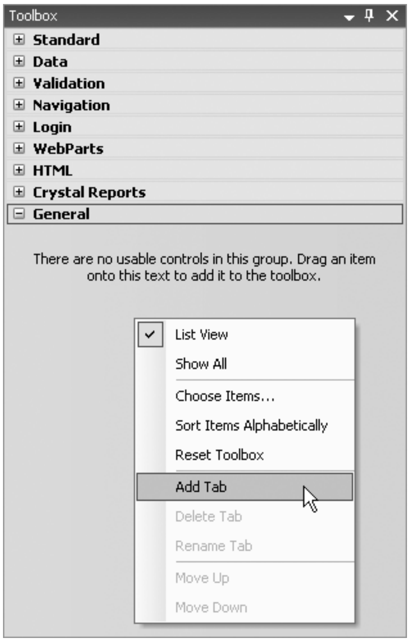
This creates a solution containing references to the Atlas binaries as well as the Atlas script libraries. For more on this, see Chapter 3.

Adding the Server Controls to the Toolbox

On the Toolbox, if you right-click, you will find an option called Add Tab (see Figure 6-2).

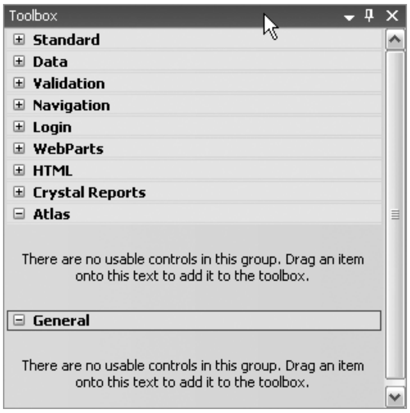
Selecting this adds a new tab to the Toolbox. This tab has a text box where you can type in a new title. Call the tab Atlas, and you will see a new, empty tab with no controls in it (see Figure 6-3).

To populate this tab with the suite of Atlas server controls, right-click it, and select Choose Items, which opens the dialog box shown in Figure 6-4. It may take a few moments if it is the first time you've done this, because the dialog box will be evaluating all the references and type libraries for the .NET and COM controls. Make sure the .NET Framework Components tab is selected.



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

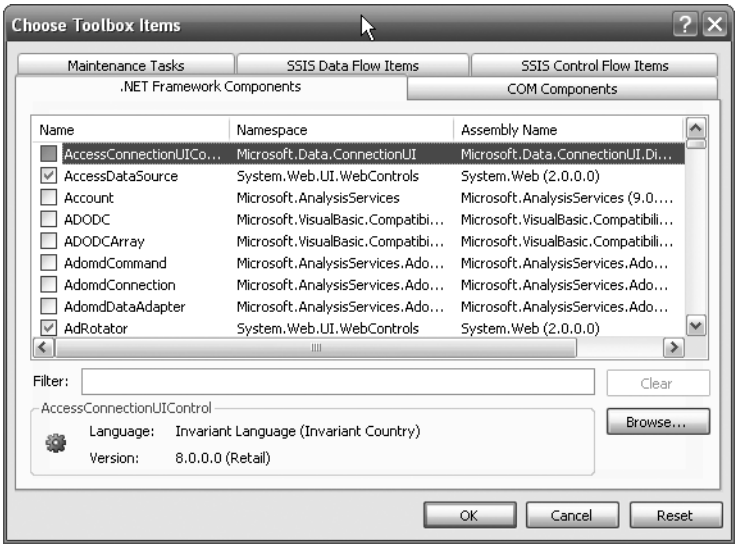
Figure 6-2. *Using the Toolbox Add Tab functionality*



There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.

Figure 6-3. *Your new, empty tab*



	Name	Namespace	Assembly Name
☐	AccessConnectionUICo...	Microsoft.Data.ConnectionUI	Microsoft.Data.ConnectionUI.Di...
✓	AccessDataSource	System.Web.UI.WebControls	System.Web (2.0.0.0)
☐	Account	Microsoft.AnalysisServices	Microsoft.AnalysisServices (9.0....
☐	ADODC	Microsoft.VisualBasic.Compatibi...	Microsoft.VisualBasic.Compatibili...
☐	ADODCArray	Microsoft.VisualBasic.Compatibi...	Microsoft.VisualBasic.Compatibili...
☐	AdomdCommand	Microsoft.AnalysisServices.Ado...	Microsoft.AnalysisServices.Ado...
☐	AdomdConnection	Microsoft.AnalysisServices.Ado...	Microsoft.AnalysisServices.Ado...
☐	AdomdDataAdapter	Microsoft.AnalysisServices.Ado...	Microsoft.AnalysisServices.Ado...
✓	AdRotator	System.Web.UI.WebControls	System.Web (2.0.0.0)

Filter:

AccessConnectionUIControl

Language: Invariant Language (Invariant Country)

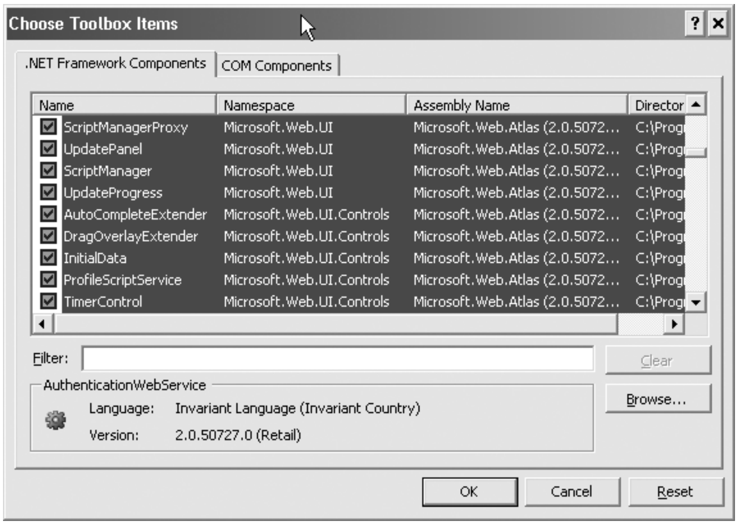
Version: 8.0.0.0 (Retail)

Figure 6-4. *Choosing Toolbox items*

By default, this dialog box lists the controls in alphabetical order of name. Click the Namespace column header to sort by namespace.

The Atlas controls appear in the Microsoft.Web.UI and Microsoft.Web.UI.Controls namespaces. If these namespaces are missing, click the Browse button, and find the Microsoft.Web.Atlas.DLL file in the \Bin directory of your web site.

You can see the Atlas controls from this assembly in Figure 6-5.



Name	Namespace	Assembly Name	Director
✓ ScriptManagerProxy	Microsoft.Web.UI	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ UpdatePanel	Microsoft.Web.UI	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ ScriptManager	Microsoft.Web.UI	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ UpdateProgress	Microsoft.Web.UI	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ AutoCompleteExtender	Microsoft.Web.UI.Controls	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ DragOverlayExtender	Microsoft.Web.UI.Controls	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ InitialData	Microsoft.Web.UI.Controls	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ ProfileScriptService	Microsoft.Web.UI.Controls	Microsoft.Web.Atlas (2.0.5072...	C:\Prog
✓ TimerControl	Microsoft.Web.UI.Controls	Microsoft.Web.Atlas (2.0.5072...	C:\Prog

AuthenticationWebService
Language: Invariant Language (Invariant Country)
Version: 2.0.50727.0 (Retail)

Figure 6-5. *The Atlas server controls*

Once you've added these controls, you'll see them on the tab you created earlier (see Figure 6-6).

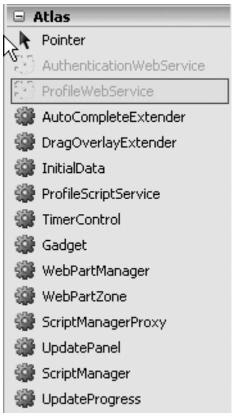


Figure 6-6. *Your Toolbox tab containing Atlas server controls*

Now that you have the controls in your Toolbox, you can drag and drop them onto your web forms. For the rest of this chapter, I'll discuss these controls and their object models, and in Chapter 7 you will start using these controls in hands-on examples.

Introducing the ScriptManager Control

The ScriptManager control is at the heart of Atlas. This control, as its name suggests, manages the deployment of the various JavaScript libraries that implement the client-side runtime functionality of Atlas.

Using the ScriptManager Designer Interface

You've used the ScriptManager control already to create references on the client side to the Atlas script libraries. Using the control is simple. When you drag and drop the control onto a page, you get a design-time user interface that allows you to set up some of the common elements of the ScriptManager control (see Figure 6-7).

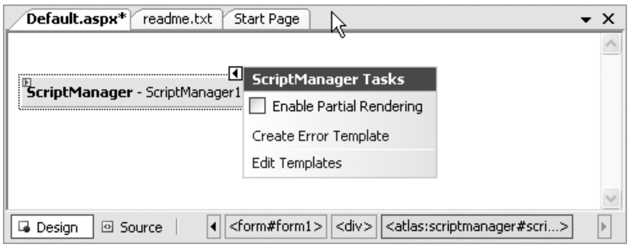


Figure 6-7. *ScriptManager design-time user interface*

This designer allows you to visually set up partial rendering, which is a tag that enables partial-page updates and is specified in UpdatePanel controls; this designer also allows you to create error templates and manage other ScriptManager templates. You'll be seeing more about these features throughout this chapter.

Now, if you take a look at the code behind this .aspx page, you will see that placing the ScriptManager control has led to the following script being added to the page:

```
<atlas:ScriptManager ID="ScriptManager1" runat="server">
</atlas:ScriptManager>
```

When you run the page, you'll see the following source code when you select the View ➤ Source command in the browser:

```
<script src="ScriptLibrary/Atlas/Debug/Atlas.js" type="text/javascript"></script>
<div>
</div>
<script type="text/xml-script">
```

```
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <components />
</page></script>
<script type="text/javascript">
</script>
```

In this case, at runtime, ASP.NET renders the server-side control as a number of statements. The first is a reference to the script library Atlas.js, which will get downloaded automatically. Next is a `<script>` tag containing any client-side Atlas Script for the page components. Finally, another script tag appears for any JavaScript you may want to use on the page, such as custom actions or transforms.

Programming with the Script Manager

The ScriptManager control is in many ways the core of an Atlas-enabled ASP.NET page. In addition to managing the download of the scripts to the client, it also orchestrates how the page updates and refreshes.

Performing Partial Rendering

The EnablePartialRendering property of this control sets how your page will behave insofar as updates are concerned. If this is false (the default), full-page refreshes will occur on round-trips to the server. If this is true, then postbacks and full-page refreshes are suppressed and replaced with targeted and partial updates. Instead of the application performing a full post-back, the application will simulate full postbacks using the XMLHttpRequest object when this is set to true (as you would expect from an Ajax application).

On the server side, the page will be processed in the normal way, responding to any controls that call _doPostBack(). Existing server-side postback events will continue to fire, and event handlers will continue to work as they always have. It is intended, by design, that Atlas-enabled applications change existing ASP.NET applications as little as possible.

The power of the ScriptManager control, when partial rendering is enabled, comes at render time. It determines, with the aid of the UpdatePanel control, which portions of the page have changed. The UpdatePanel, which you will see more of later in this chapter, defines regions in the page that get updated as a chunk. If, for example, you have a page containing a number of chat rooms and you want to update only a single chat room, you would surround that area of the page with an UpdatePanel control.

The ScriptManager control will override the rendering of the page and instead will send HTML down to the XMLHttpRequest object for each of the UpdatePanel controls on the page.

Enabling Script Components

When you browse to an Atlas-enabled application, two script libraries get downloaded to the client; these are Atlas.js for Internet Explorer and AtlasCompat.js for other browsers. These script libraries effectively bootstrap Atlas functionality on the client side. The ScriptManager control handles the rendering of these references on the page (as `<script>` tags), so you don't need to manually add them to your page.

If you add a ScriptManager control to a blank page and then execute the application, you'll see the following tag embedded in your page (in this case, it is running in debug mode) when running Internet Explorer:

```
<script src="ScriptLibrary/Atlas/Debug/Atlas.js"
        type="text/javascript">
</script>
```

And you'll see this one when running it in Firefox:

```
<script src="ScriptLibrary/Atlas/Debug/AtlasCompat.js"
        type="text/javascript">
</script>
<script src="ScriptLibrary/Atlas/Debug/Atlas.js"
        type="text/javascript">
</script>
```

Specifying Additional Script Components

The ScriptManager control has a `<Scripts>` child tag that can specify additional scripts to download to the browser. This should contain one or more `<atlas:ScriptReference>` tags that specify the path to the script and the browser the script targets.

This tag has three parameters:

Browser: This attribute allows you to target the script at a specific browser. Here's an example that instructs the ScriptManager control to download the script at the path myff.js to Firefox-based callers:

```
<atlas:ScriptReference Browser="FireFox" Path="myff.js" />
```

Path: This specifies the path where the ScriptManager control can find the script file to download. In the previous example, myff.js was in the same directory as the page containing the `<ScriptManager>` tag.

ScriptName: If you want to target one of the built-in Atlas scripts, instead of a custom one, you can reference them by name here. This means you don't have to specify the directory of the script, so you can have consistent `<atlas:ScriptReference>` tags on your pages regardless of their location within the directory structure on the page. Here's an example that instructs the ScriptManager control to download the script AtlasUIGlitz.js to Firefox-based browsers:

```
<atlas:ScriptReference Browser="FireFox" ScriptName="AtlasUIGlitz" />
```

So when using the optional script components on a page, your ScriptManager control will look something like this:

```
<atlas:ScriptManager ID="ScriptManager1" runat="server">
<Scripts>
  <atlas:ScriptReference Browser="Firefox"
       Path="hello.js" ScriptName="Custom" />
  <atlas:ScriptReference Browser="Firefox"
       ScriptName="AtlasUIGlitz" />
</Scripts>
</atlas:ScriptManager>
```

When you run the page containing this script on Internet Explorer, you will see this:

```
<script src="ScriptLibrary/Atlas/Debug/Atlas.js"
        type="text/javascript">
</script>
```

When you run the page containing this script on Firefox, you will see this:

```
<script src="ScriptLibrary/Atlas/Debug/AtlasCompat.js"
        type="text/javascript">
</script>
<script src="ScriptLibrary/Atlas/Debug/Atlas.js"
        type="text/javascript"></script>
<div>&nbsp;</div>

<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <references>
    <add src="hello.js" />
    <add src="ScriptLibrary/Atlas/Debug/AtlasUIGlitz.js" />
  </references>
  <components />
</page></script>
```

Specifying Services

In Chapter 2 you saw how a service can be directly consumed in a client application through a script-based proxy to it. You can use the ScriptManager control to reference this using the `<Services>` child tag. This tag should contain one or more `<atlas:ServiceReference>` tags that specify the service you want to reference.

This tag has two attributes:

Path: This specifies the path to the service. You saw in Chapter 2 that JavaScript proxies to web services on Atlas web sites can be automatically generated by postfixing "/js" at the end of its URI. So, for example, the web service at test.asmx would return a JavaScript proxy that could be used to call it at test.asmx/js. When using the `<atlas:ServiceReference>` tag to specify the service, this would automatically be generated for you on the client side when the ScriptManager control is rendered. Here's an example:

```
<atlas:ServiceReference Path="wstest.asmx"/>
```

GenerateProxy: This is a Boolean value (true or false) that specifies whether the service reference should generate a proxy class or not. The default is true.

When running a page that contains this within a ScriptManager control, you will get the following code on the client:

```
<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <references>
    <add src="wstest.asmx/js" />
  </references>
  <components />
</page>
</script>
```

As you can see, a `<reference>` tag has been added to the Atlas Script, and this references the JavaScript proxy for the wstest.asmx web service. You'll see some examples on how to use web services and their proxies in Chapter 7.

Using Error Templates

The ScriptManager control provides an error handling mechanism whereby you can specify the HTML to render when an error condition is met. This is particularly useful for the client experience because you can then help your users gracefully handle errors. Within the Script-Manager control, you can use the `<ErrorTemplate>` tag to define your error message in HTML. So, for example, the following script will produce a user-friendly response to an error:

```
<atlas:ScriptManager ID="ScriptManager1" runat="server">
<ErrorTemplate>
    There was an error processing your action.<br />
    <span id="errorMessageLabel"></span>
    <hr />
    <button type="button" id="okButton">OK</button>
</ErrorTemplate>
</atlas:ScriptManager>
```

The HTML defined within the `<ErrorTemplate>` element will render in a simulation of a modal dialog box. This is achieved by the rest of the page being partially obscured using a semitransparent overlay. The HTML within the `<ErrorTemplate>` element is enabled and active. This allows you to design a pretty rich error display according to your preferences.

■ **Note** The `<ErrorTemplate>` element is meaningful only when used in conjunction with an UpdatePanel control because it triggers the asynchronous communication used by the UpdatePanel control. So, although the `<ErrorTemplate>` is defined on the ScriptManager control, it will do nothing until an error is received during an asynchronous update.

It's important to remember the ID values used in this example. You must always use these values (errorMessageLabel and okButton). The ScriptManager control generates XML script that uses these controls to bind controls to the error message property and to clear error methods on the PageRequestManager control.

You can also design this HTML using the Visual Studio 2005 IDE. When you mouse over the ScriptManager control in the page designer, you'll see a small right-pointing arrow on its top-right corner (see Figure 6-8).

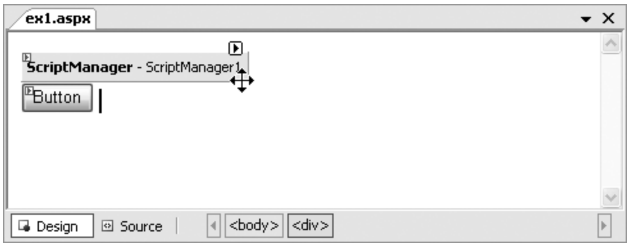


Figure 6-8. *The ScriptManager control*

If you click this arrow, the ScriptManager Tasks Assistant will appear. From this assistant, you can configure or remove the error template and enable partial rendering (see Figure 6-9).

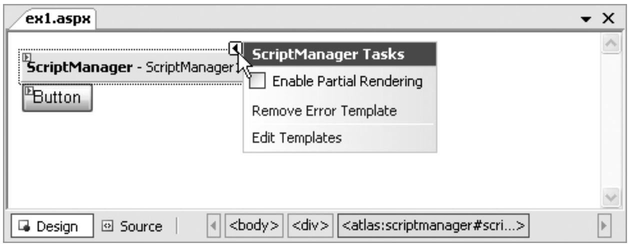


Figure 6-9. *The ScriptManager Tasks Assistant when an* `<ErrorTemplate>` *element is already configured*

As in this case, the ScriptManager control already has a template associated with it, and the option Remove Error Template is present. Figure 6-10 shows how the ScriptManager Tasks Assistant will appear when no template is present.

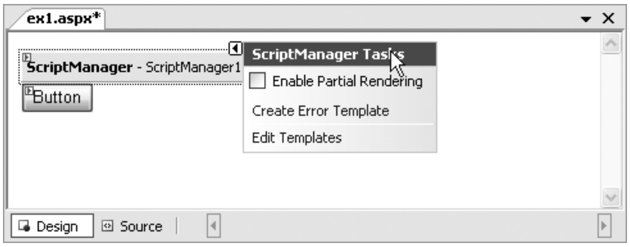
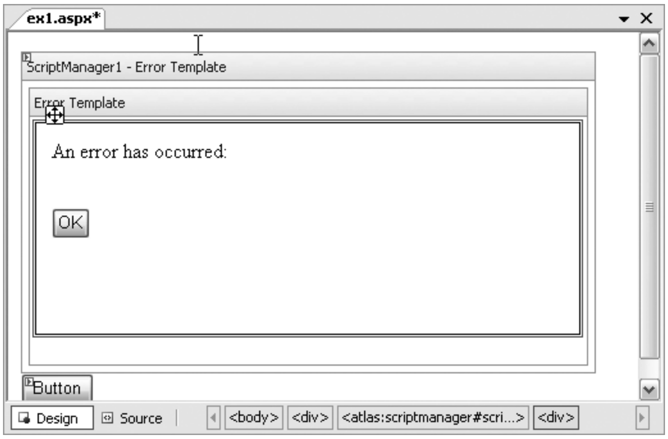


Figure 6-10. *The ScriptManager Tasks Assistant when there is no* `<ErrorTemplate>` *element present*

To create an error template, click the Create Error Template link. This will not only create a new Error Template for you, but it will also populate it with the standard error message HTML you saw earlier.

If you select Edit Templates, the ScriptManager control surface will open a basic HTML editor that you can use to customize your error template (see Figure 6-11).

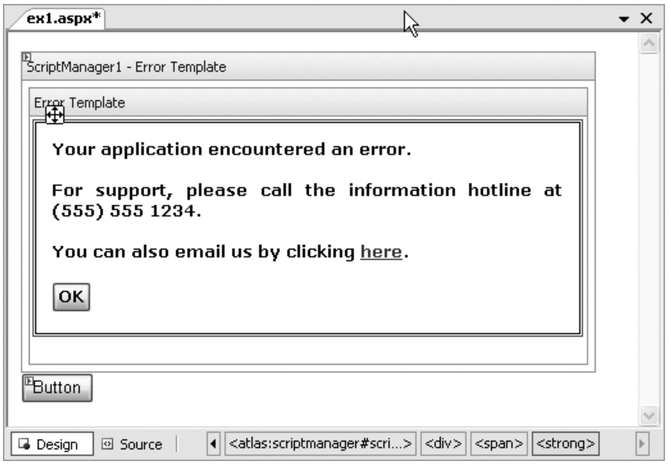

ex1.aspx*
ScriptManager1 - Error Template
Error Template
Button
Design
Source
<body>
<div>
<atlas:scriptmanager#scri...>
<div>

An error has occurred:

OK

Figure 6-11. *Editing the HTML for your* `<ErrorLayout>` *tag on your ScriptManager control*

As mentioned earlier, you can use the HTML editor to specify how you want this error box to appear. Figure 6-12 shows a customized version of the error template.


ex1.aspx*
ScriptManager1 - Error Template
Error Template
Button
Design
Source
<atlas:scriptmanager#scri...>
<div>
<span>
<strong>

Your application encountered an error.

For support, please call the information hotline at (555) 555 1234.

You can also email us by clicking here.

OK

Figure 6-12. *A custom error template*

The designer then generates the `<ErrorTemplate>` code for you. Here is the code associated with the custom error template shown in Figure 6-12:

```
<atlas:ScriptManager ID="ScriptManager1" runat="server">
<ErrorTemplate>
<div style="padding: 12px; width: 400px; height: 140px;
        border: #000000 1px solid;background-color: white; text-align: justify">
<span style="font-size: 10pt; font-family: Verdana">
<strong>
Your application encountered an error.
<br />
<br />
For support, please call the information hotline at (555) 555 1234.
<br />
<span id="errorMessageLabel">
</span>
<br />
You can also email us by clicking
<a href="mailto:support@supporthotline.com">here</a>.
<br />
<br />
</strong>
</span>
<input id="okButton" type="button" value="OK"
style="font-weight: bold; font-size: 10pt; font-family: Verdana" />
<spanstyle="font-size: 10pt; font-family: Verdana">
</span>
</div>
</ErrorTemplate>

</atlas:ScriptManager>
```

To run an application that creates an error, you can add a button to the page. This button, when clicked, will trigger the error.

First add a button:

```
<asp:Button runat="server"
            ID="MyButton"
            Text="Click Me!"
            OnClick="MyButton_Click" />
```

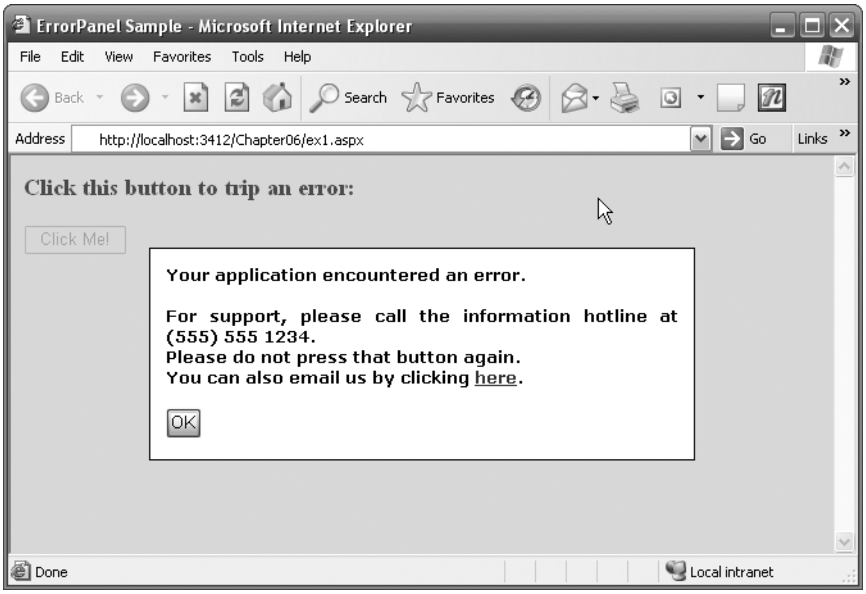
When you click this button, you will call the MyButton_Click event. You define this using a server-side script:

```
<script runat="server">
  private void OnScriptManagerPageError(object sender,
                                  PageErrorEventArgs e)
  {
    e.ErrorMessage = "Please do not press that button again.";
  }
  void MyButton_Click(object sender, EventArgs e)
  {
    throw new ArgumentException("Please do not press
        that button again.", "MyButton");

  }
</script>
```

This script throws an exception when you click the button. The ScriptManager control is also configured to call OnScriptManagerPageError when a page error occurs:

```
<atlas:ScriptManager runat="server"
     ID="scriptManager"
     EnablePartialRendering="true"
     OnPageError="OnScriptManagerPageError">
```

Now when you click the button, the error will fire, and your page will display the HTML defined in the error template (see Figure 6-13).


ErrorPanel Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:3412/Chapter06/ex1.aspx Go Links
Click Me!
OK
Done
Local intranet

Click this button to trip an error:

Your application encountered an error.

For support, please call the information hotline at (555) 555 1234.
Please do not press that button again.
You can also email us by clicking here.

Figure 6-13. *Atlas showing an error*

This concludes the tour of the ScriptManager control. In the rest of this chapter, you will look at the other server-side controls offered by the Atlas framework, and in the next chapter you will go through several examples that use these controls.

Introducing the ScriptManagerProxy Control

The ScriptManagerProxy control is available as an additional script manager for a page. Only one ScriptManager control is allowed, and if you, for example, place a ScriptManager control on a master page but need to add script references to your content page, then you can use the ScriptManagerProxy control.

So, if you have the following master page:

```
<%@ Master Language="C#" AutoEventWireup="true"
        CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <atlas:ScriptManager ID="ScriptManager1" runat="server">
        </atlas:ScriptManager>
            This is the Master Page.<br />
            It contains this ScriptManager control:<br />
        <br />
        <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
            <br />
        </asp:contentplaceholder>
        &nbsp;
    </div>
    </form>
</body>
</html>
```

and you create a new content page based on this master page, your new page will look like this at design time:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master" AutoEventWireup="true"
        CodeFile="Default3.aspx.cs" Inherits="Default3" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
</asp:Content>
```

When you run this page and look at the source code that is generated by ASP.NET from this design-time definition, you will see this:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
        Untitled Page
</title></head>
<body>
    <form name="aspnetForm" method="post" action="Default3.aspx" id="aspnetForm">
<div>
<input type="hidden" name="__VIEWSTATE"
        id="__VIEWSTATE"
        value="/wEPDwULLTEwMDUyNjYzMjhkZERQ3hQz51DsahcItBSVyAcTtqP7" />
</div>


<script src="ScriptLibrary/Atlas/Debug/Atlas.js" type="text/javascript"></script>
    <div>

            This is the Master Page.<br />
            It contains this ScriptManager control:<br />
        <br />



        &nbsp;
    </div>

<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <components />
</page></script>
<script type="text/javascript">
</script>
</form>
</body>
</html>
```

You can see here where the `<script>` tag is generated, referencing the Atlas.js core libraries. However, if this page requires using the AtlasUIGlitz library and you don't want to add a reference to that library to the template (because this page is the only one on your site that needs it), then you can use a ScriptManagerProxy control.

So, if you return to the content page that was generated by the master page, you can add a ScriptManagerProxy control to it in the designer (see Figure 6-14).

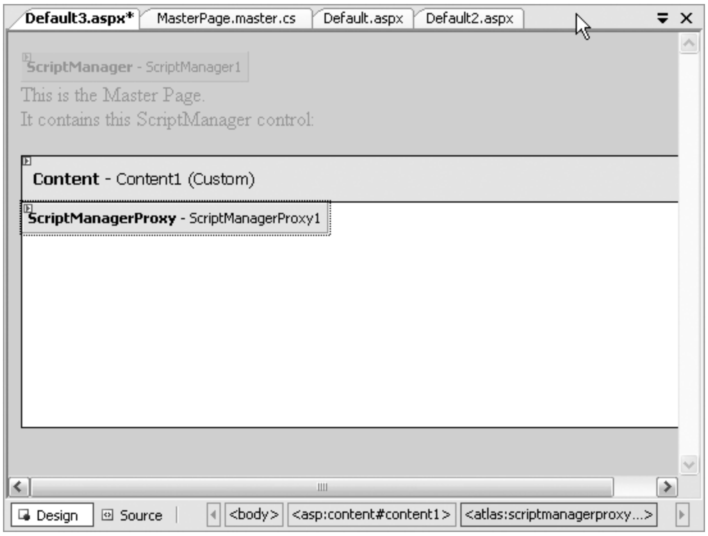

Default3.aspx*
MasterPage.master.cs
Default.aspx
Default2.aspx
ScriptManager - ScriptManager1
This is the Master Page.
It contains this ScriptManager control:
Content - Content1 (Custom)
ScriptManagerProxy - ScriptManagerProxy1
Design
Source
<body>
<asp:content#content1>
<atlas:scriptmanagerproxy...>

Figure 6-14. *Adding a ScriptManagerProxy control to a content page*

By adding the ScriptManagerProxy control to the content page, you added the following code to the page; you can see this in the Source view:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
        Runat="Server">
    <atlas:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    </atlas:ScriptManagerProxy>
</asp:Content>
```

Now you can simply add the scripts that you want to this page using the `<Scripts>` child tag in the same manner as you did with the ScriptManager control.

Here's an example:

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
        Runat="Server">
    <atlas:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
        <Scripts>
            <atlas:ScriptReference ScriptName="AtlasUIDragDrop" />
        </Scripts>
    </atlas:ScriptManagerProxy>
</asp:Content>
```

So now when you run the page, you will see that AtlasUIDragDrop has been included in this page:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
        Untitled Page
</title></head>
<body>
    <form name="aspnetForm" method="post" action="Default3.aspx" id="aspnetForm">
<div>
<input type="hidden" name="__VIEWSTATE"
        id="__VIEWSTATE"
        value="/wEPDwUKLTc4NzcwODI3NmRk9YXV49GLEVNvGzvlq/8dm4TNJOg=" />
</div>
<script src="ScriptLibrary/Atlas/Debug/Atlas.js" type="text/javascript"></script>
    <div>
            This is the Master Page.<br />
            It contains this ScriptManager control:<br />
        <br />
        &nbsp;
    </div>

<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
  <references>
    <add src="ScriptLibrary/Atlas/Debug/AtlasUIDragDrop.js" />
  </references>
  <components />
</page></script>
<script type="text/javascript">
</script>
</form>
</body>
</html>
```

Introducing the UpdatePanel Control

In typical ASP.NET 2.0 applications, if you do a postback on the web page, the entire page will be rerendered. This causes a “blink” or a “flash” in the client or browser. On the server, the postback is detected, which triggers the page life cycle. This ends up raising the specific postback event handler code for the control that caused the postback, and this calls upon the page’s event handler.

When you use UpdatePanel controls along with a ScriptManager control, you eliminate the need for a full-page refresh. The UpdatePanel control is similar to a ContentPanel control in that it marks out a region on the web page that will automatically be updated when the postback occurs (but without the aforementioned postback behavior on the client).

It will instead communicate through the XMLHttpRequest channel—in true Ajax style. The page on the server still handles the postback as expected and executes, raising event handlers, and so on, but the final rendering of the page means that only the regions specified in the UpdatePanel control’s regions will be created.

Using the UpdatePanel Designer

Visual Studio 2005 provides a designer for the UpdatePanel control, including a Tasks pane that helps you set up the control. To use an UpdatePanel control, you simply drag and drop it onto the design surface of your web form (see Figure 6-15).

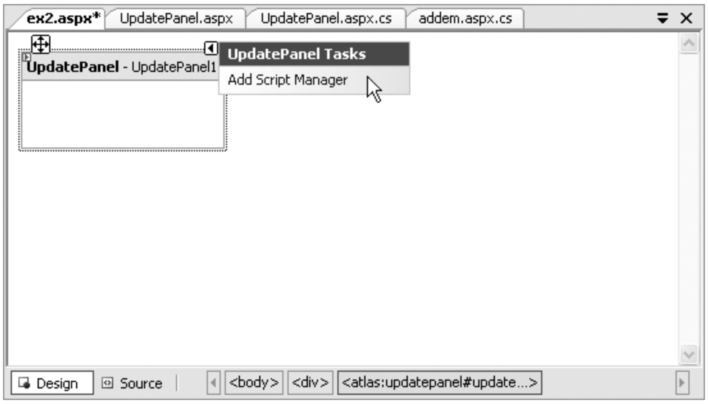


Figure 6-15. *Using the UpdatePanel designer*

As you can see from Figure 6-15, the only task associated with the UpdatePanel control is to add a ScriptManager control. The UpdatePanel control cannot function without a Script-Manager control on the page. Additionally, the ScriptManager control must be located before any UpdatePanel controls on your page. In other words, as you read your source code from top to bottom, the ScriptManager reference should appear before the UpdatePanel ones. Using the Tasks Assistant will ensure that it is placed correctly. If your ScriptManager control is not present or is incorrectly placed, you will get an error (see Figure 6-16).

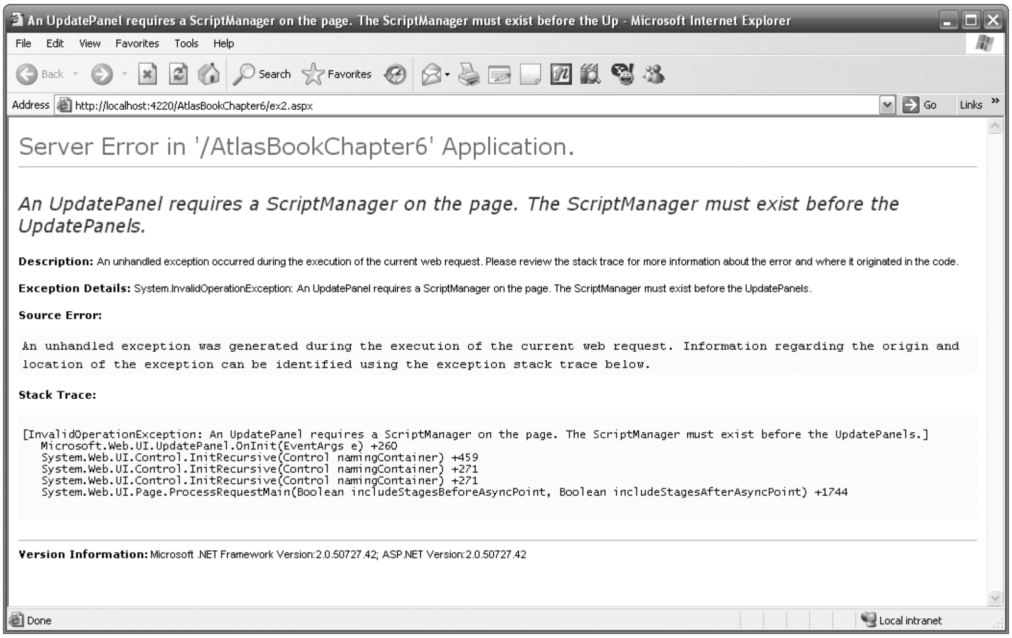
The UpdatePanel control contains a designer surface where you can place HTML. This code will be the only code updated upon a postback if the ScriptManager control is enabled for partial updates.

Consider Figure 6-17, where several text boxes and a button appear on the screen.

This application has two text boxes, two labels, and a button *outside* the UpdatePanel control, and it has a label *inside* the UpdatePanel designer. The label on the inside is called lblResult. The code behind the button reads as follows:

```
int x = Convert.ToInt16(txt1.Text.ToString());
int y = Convert.ToInt16(txt2.Text.ToString());
int z = x+y;
lblResult.Text = z.ToString();
```

As you can see, the label for the result will get updated to the value of the sum of the values of the text in the text boxes. Because lblResult is in the UpdatePanel control and the ScriptManager control is set to enable partial rendering, clicking the button will update only the text within the UpdatePanel control. You will see and dissect more examples of this in Chapter 7.


An UpdatePanel requires a ScriptManager on the page. The ScriptManager must exist before the Up - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4220/AtlasBookChapter6/ex2.aspx
Go Links
Done
Local intranet

Server Error in '/AtlasBookChapter6' Application.

An UpdatePanel requires a ScriptManager on the page. The ScriptManager must exist before the UpdatePanels.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.InvalidOperationException: An UpdatePanel requires a ScriptManager on the page. The ScriptManager must exist before the UpdatePanels.

Source Error:

```
An unhandled exception was generated during the execution of the current web request. Information regarding the origin and
location of the exception can be identified using the exception stack trace below.
```

Stack Trace:

```
[InvalidOperationException: An UpdatePanel requires a ScriptManager on the page. The ScriptManager must exist before the UpdatePanels.]
   Microsoft.Web.UI.UpdatePanel.OnInit(EventArgs e) +260
   System.Web.UI.Control.InitRecursive(Control namingContainer) +459
   System.Web.UI.Control.InitRecursive(Control namingContainer) +271
   System.Web.UI.Control.InitRecursive(Control namingContainer) +271
   System.Web.UI.Page.ProcessRequestMain(Boolean includeStagesBeforeAsyncPoint, Boolean includeStagesAfterAsyncPoint) +1744
```

Version Information: Microsoft .NET Framework Version:2.0.50727.42; ASP.NET Version:2.0.50727.42

Figure 6-16. *Error page when the UpdatePanel and ScriptManager controls aren't properly configured*

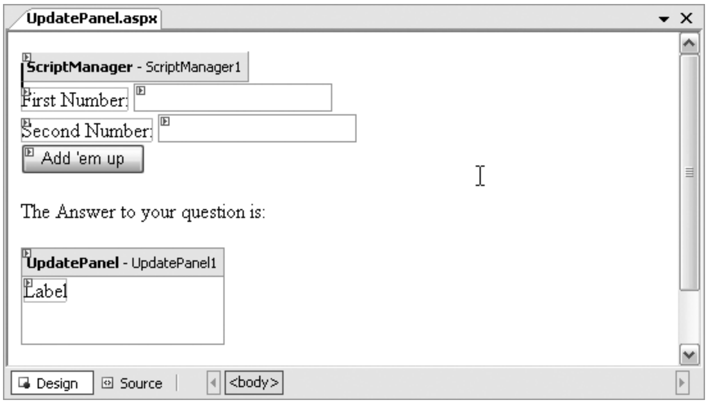

UpdatePanel.aspx
ScriptManager - ScriptManager1
First Number:
Second Number:
Add 'em up
The Answer to your question is:
UpdatePanel - UpdatePanel1
Label
Design
Source
<body>

Figure 6-17. *Simple application that uses the UpdatePanel control*

Programming with the UpdatePanel

If you look at the code behind the designer for the UpdatePanel control from the previous example, you'll see the following:

```
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
  <asp:Label ID="lblResult" runat="server" Text="Label"></asp:Label>
</ContentTemplate>
</atlas:UpdatePanel>
```

The `<atlas:UpdatePanel>` tag supports two child tags: the `<ContentTemplate>` tag and the `<Triggers>` tag.

Using the ContentTemplate Tag

The `<ContentTemplate>` tag simply defines the HTML or ASP.NET that will get updated by the UpdatePanel control. You can use the designer to generate this HTML. If, for example, you drag and drop a Calendar control onto the UpdatePanel control's content template area (see Figure 6-18), it will be defined within the `<ContentTemplate>` tag area.

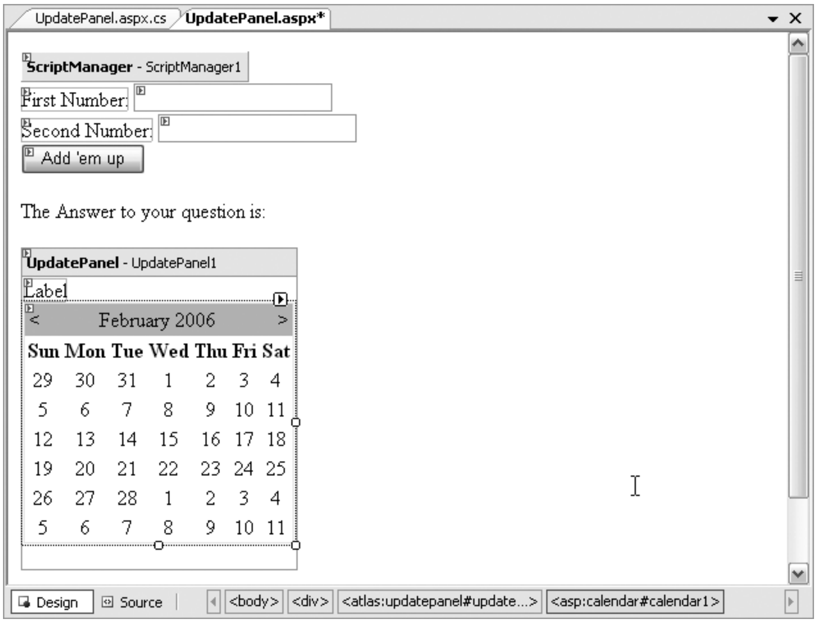


Figure 6-18. *Adding controls to the UpdatePanel control's content template*

You can see the code that is produced by adding the calendar as follows:

```
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
  <asp:Label ID="lblResult" runat="server" Text="Label"></asp:Label>
  <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
</ContentTemplate>
</atlas:UpdatePanel>
```

Using Triggers

The other child tag for the UpdatePanel control is `<Triggers>`. This allows you to define triggers for the update. The UpdatePanel control has a mode property attribute, and if you set this to Conditional (the other option is Always), then updates to the rendering of the markup will occur only when a trigger is hit. This tag contains the collection of trigger definitions. Two types of trigger exist, defined next.

Using ControlEventTrigger

When using a ControlEventTrigger trigger type, you define a trigger that has an associated control (specified by ControlID) and an event name (specified by the EventName) on that control. If the event is raised on that control, then the trigger fires, and the UpdatePanel control will be rendered.

You specify a ControlEventTrigger with the `<atlas:ControlEventTrigger>` tag. Here's an example:

```
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <asp:Label ID="lblResult" runat="server"
                   Text="Label">
        </asp:Label>
        <asp:Calendar ID="Calendar1" runat="server">
        </asp:Calendar>
      </ContentTemplate>
      <Triggers>
        <atlas:ControlEventTrigger ControlID="btnAdd"
                EventName="Click" />
      </Triggers>
</atlas:UpdatePanel>
```

Here the ControlEventTrigger specifies that the source for the trigger is the button called btnAdd, and the event on which to trigger is the Click event. Therefore, when the button is clicked, the ControlEventTrigger will fire, and the partial update will occur.

Using ControlValueTrigger

ControlValueTrigger defines a trigger that fires when the value of a property (specified by PropertyName) for the associated control (specified by ControlID) changes. Upon the change happening, the trigger fires, and the UpdatePanel control will be rendered during the partial rendering of the page on the server. You specify a ControlValueTrigger trigger type with the `<atlas:ControlValueTrigger>` tag.

Here's an example:

```
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
        <asp:Label ID="lblResult" runat="server"
                   Text="Label">
        </asp:Label>
        <asp:Calendar ID="Calendar1" runat="server">
        </asp:Calendar>

      </ContentTemplate>
      <Triggers>
        <atlas:ControlValueTrigger ControlID="txt1"
               PropertyName="Text" />
      </Triggers>
</atlas:UpdatePanel>
```

Here the ControlValueTrigger specifies that the source for the trigger is the txt1 control and that the trigger should fire whenever its text property changes.

Introducing the UpdateProgress Control

Another control that Atlas provides is the UpdateProgress control. This indicates the progress of an asynchronous transaction taking place. Typically the browser's status bar serves as an indicator of activity. With the partial-rendering model, this is no longer applicable, but to make Ajax activity simple and user-friendly, the UpdateProgress control is provided.

■**Note** As with the `<ErrorTemplate>` element you saw earlier, the UpdateProgress control is meaningful only when there is at least one UpdatePanel control on the page. This is because the control is triggered off the asynchronous communication of the underlying XMLHttpRequest object.

To use an UpdateProgress control, you drag and drop it onto your page. This will create an `<atlas:UpdateProgress>` tag on your page.

The HTML to display when the call is taking place is then defined using the `<ProgressTemplate>` tag.

When your application executes calls to the server, the HTML defined in the `<ProgressTemplate>` tag is then displayed.

Here's an example that specifies showing the image of a smiley face while the server is being called:

```
<atlas:UpdateProgress runat="server" ID="updateProgress1">
  <ProgressTemplate>
  <img src="Images/smiling.gif" />
  Contacting Server...
  </ProgressTemplate>
</atlas:UpdateProgress>
```

Introducing Control Extenders

A number of control extenders are available as server-side controls in Atlas. These provide a way to attach rich client-side functionality to the server controls. The available extenders are AutoCompleteExtender and DragOverlayExtender.

Introducing the AutoCompleteExtender

The AutoCompleteExtender control works in conjunction with an AutoCompleteProperties control. It provides for autocomplete functionality on client-side controls, so if you want a text box to provide autocomplete functionality, for example, you would create an AutoCompleteExtender control and an AutoCompleteProperties control. The former would define the extender; the latter would define the target of the autocomplete (in this case the text box) as well as the service and service method that provide the autocomplete values.

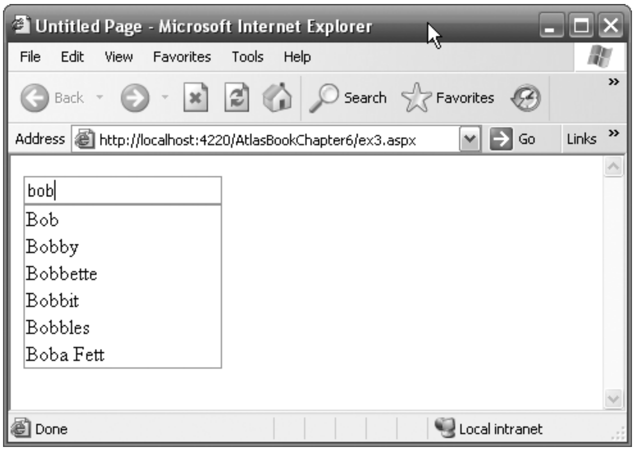
This is best demonstrated with a simple example. Here is the HTML for a web form containing a single text box, along with ScriptManager, AutoCompleteExtender, and AutoCompleteProperties controls:

```
<form id="form1" runat="server">
<div>
  <atlas:ScriptManager ID="ScriptManager1"
       runat="server"
       EnablePartialRendering="True">
  </atlas:ScriptManager>
  <asp:TextBox ID="TextBox1" runat="server">
  </asp:TextBox>
  <atlas:AutoCompleteExtender ID="AutoCompleteExtender1"
       runat="server">
    <atlas:AutoCompleteProperties Enabled="True"
       ServiceMethod="GetWordList"
       ServicePath="wordlst.asmx"
       TargetControlID="TextBox1" />
  </atlas:AutoCompleteExtender>
</div>
</form>
```

You can see that the AutoCompleteProperties control points at a web service called wordlst.asmx and a method on this service called GetWordList. You will need to create this service in your project. The code for the GetWordList web method is as follows:

```
[WebMethod]
    public string[] GetWordList()
    {
        String[] theWordList = new String[6];
        theWordList[0] = "Bob";
        theWordList[1] = "Bobby";
        theWordList[2] = "Bobbette";
        theWordList[3] = "Bobbit";
        theWordList[4] = "Bobbles";
        theWordList[5] = "Boba Fett";
        return theWordList;

    }
```

This is pretty straightforward, returning a hard-coded list of words. The autocomplete behavior kicks in once you've typed three characters into the text box. So, at runtime, if you type the letters *bob* into the text box (in memory of Microsoft Bob), you will see the autocomplete list appear (see Figure 6-19).


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4220/AtlasBookChapter6/ex3.aspx Go Links
Done
Local intranet

bob
Bob
Bobby
Bobbette
Bobbit
Bobbles
Boba Fett

Figure 6-19. *Demonstrating the AutoCompleteExtender control*

Using the DragOverlayExtender

The DragOverlayExtender control allows you to add drag-and-drop functionality to any control in a similar manner to the drag-and-drop behaviors discussed in Chapters 4 and 5. What is nice about this one is that you can declare it on the server side to enhance your existing applications. This demonstrates one of the design tenets of Atlas—it can enhance your existing ASP.NET applications without significantly changing them, so if you have an ASP.NET Label control that you want to add drag and drop to, you can do so, without touching its code. You simply add a DragOverlayExtender control and its associated DragOverlayProperties control and configure them to extend that control.

Consider the following page:

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeFile="ex4.aspx.cs" Inherits="ex4" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
<style type="text/css">
  body{font-family:Verdana;}
  .label{font-weight:bold;}
  .dropArea{height:500px;border:solid 2px Black;background:#ccc;}

</style>
</head>
<body>
  <form id="form1" runat="server">
   <div class="dropArea">

    <asp:Label ID="lbl" runat="server" CssClass="label">
  Drag Me with your mouse!
    </asp:Label>

   </div>
  </form>
</body>
</html>
```

This will provide a simple ASP.NET page containing a gray `<div>` element with a single label on it (see Figure 6-20).

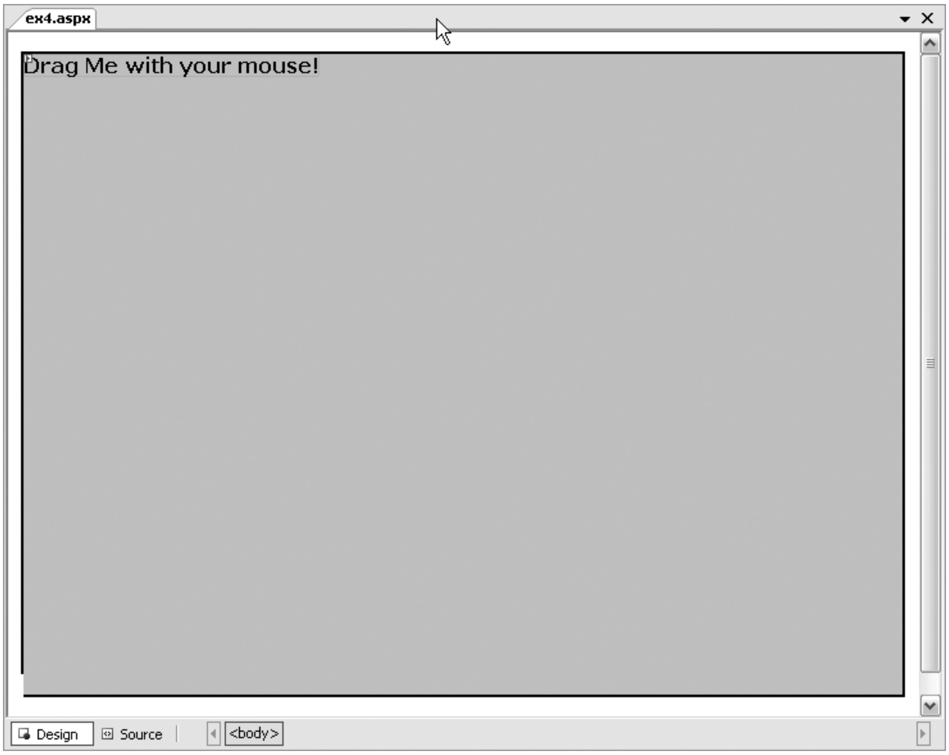


Figure 6-20. *Simple ASP.NET page with a* `<div>` *element and a* `<label>` *element*

You can now make the label draggable and droppable using the DragControlExtender control. You achieve this by first adding a ScriptManager control to the page so that the Atlas runtime components will be downloaded to the client.

Once you've done that, you can add a DragOverlayExtender control to the application (see Figure 6-21).

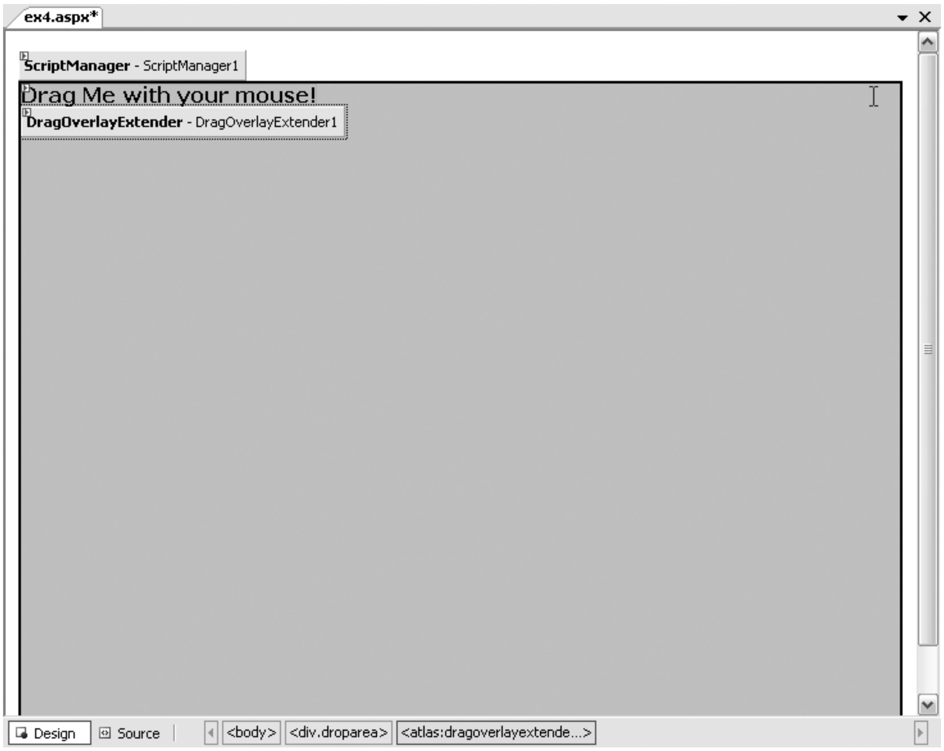

ex4.aspx*
ScriptManager - ScriptManager1
Drag Me with your mouse!
DragOverlayExtender - DragOverlayExtender1
Design
Source
<body>
<div.droparea>
<atlas:dragoverlayextende...>

Figure 6-21. *Adding the ScriptManager control and the DragOverlayExtender control to the web form*

This will add the ScriptManager code and the following snippet for the DragOverlayExtender control:

```
<atlas:DragOverlayExtender ID="DragOverlayExtender1" runat="server">
</atlas:DragOverlayExtender>
```

You then use the DragOverlayProperties control to configure this to make the label draggable and droppable. There is no visual designer for this, so you need to add it to the page code like this:

```
<atlas:DragOverlayExtender ID="DragOverlayExtender1" runat="server">
  <atlas:DragOverlayProperties TargetControlID="lbl" Enabled="true" />
</atlas:DragOverlayExtender>
```

Now if you view the page in your browser, you can drag and drop the label anywhere within the gray area (see Figure 6-22).

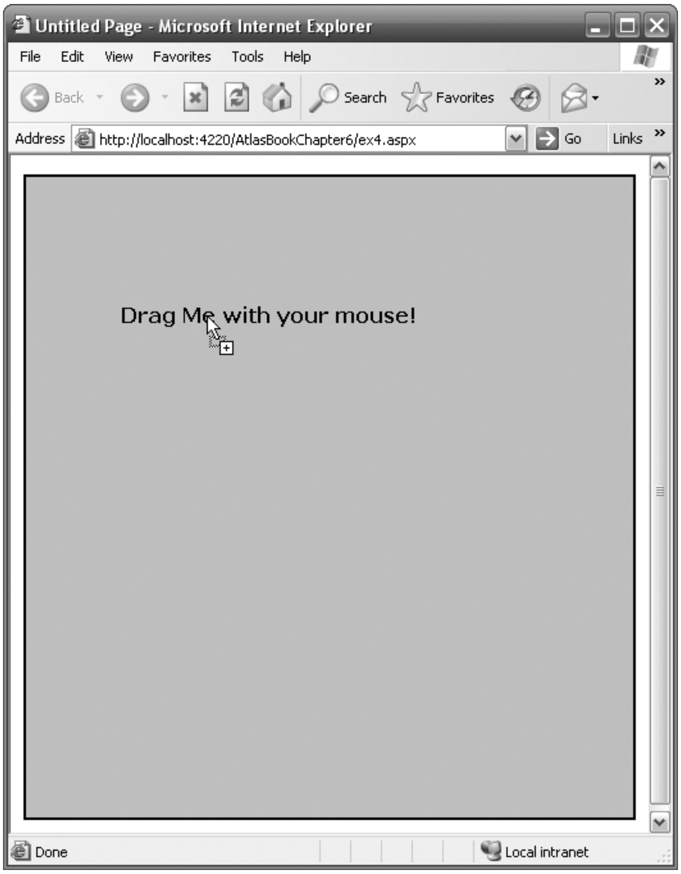

Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4220/AtlasBookChapter6/ex4.aspx Go Links
Drag Me with your mouse!
Done
Local intranet

Figure 6-22. *Dragging a label using the DragControlExtender control*

Introducing the ProfileScriptService Control

ASP.NET 2.0 provides a profile application service for persisting custom, per-user data across browser sessions. You add a profile property variable to the Web.config file like this:

```
<profile defaultProvider="AspNetSqlProfileProvider">
  <properties>
    <add name="labelLocation" />
  </properties>
</profile>
```

In this manner, the profile variable will be persisted for you in a SQL Server database. You must have a SQL Server or SQL Server Express Edition already set up to be able to use profiles. For more information on profiles, you can read *Pro ASP.NET 2.0 in C# 2005* (Apress, 2005) or *Pro ASP.NET 2.0 in VB .NET 2005* (Apress, 2006) or refer to Chapter 24 of *Pro ASP.NET 2.0 in VB .NET 2005*.

Once you are using profiles, you can then integrate them with Atlas using the ProfileScriptService control.

To do this, you can drag and drop a ProfileScriptService control onto the design surface for your web form (see Figure 6-23).

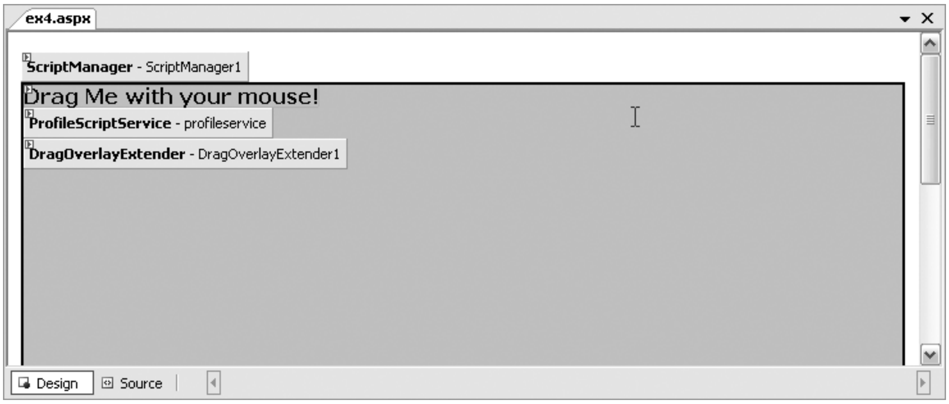

ex4.aspx
ScriptManager - ScriptManager1
Drag Me with your mouse!
ProfileScriptService - profileservice
DragOverlayExtender - DragOverlayExtender1
Design
Source

Figure 6-23. *Adding a ProfileScriptService control*

This will add the following tag to your page:

```
<atlas:ProfileScriptService ID="ProfileScriptService1" runat="server">
</atlas:ProfileScriptService>
```

Your page is now able to access profile variables through Atlas.

For an example of this, the DragOverlayProperties control allows you to specify a profile variable that can be used to persist the location of the control that you are dragging and dropping between sessions. So, if you amend the previous drag-and-drop example to specify the profile property that you want to use to persist the location like this:

```
<atlas:DragOverlayExtender ID="DragOverlayExtender1" runat="server">
  <atlas:DragOverlayProperties TargetControlID="lbl"
      Enabled="true" ProfileProperty="labelLocation" />
</atlas:DragOverlayExtender>
```

then you will be able to persist the location of the control between page sessions.

Introducing the Timer Control

Atlas provides a simple-to-use Timer control that can be configured to perform operations repeatedly based on the time elapsed.

You can add a Timer control to a page by dragging and dropping it onto the control surface. To use a Timer control, you will of course need a ScriptManager control on the page. A good use for timers is to update the contents of an UpdatePanel control when the timer ticks.

To see the Timer control in action, you can add an UpdatePanel control to a blank page and use its Tasks pane to add a ScriptManager control to the page. Once you've done this, you can drag and drop a Timer control onto the page. You can see what this looks like in the designer in Figure 6-24.

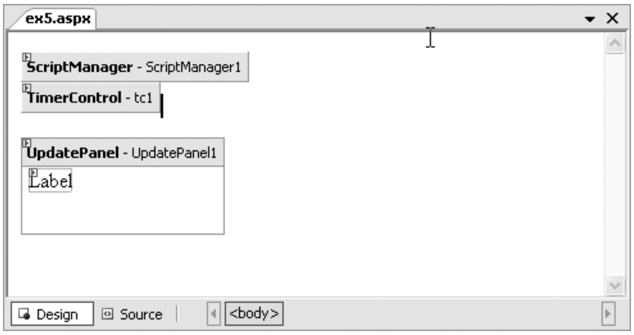


Figure 6-24. *Using a Timer control in the designer*

The code behind your page will now have the `<Timer>` tag already defined.

Here's an example of a timer that has been customized with a 5,000-millisecond interval (5 seconds in other words), with the name tc1 and the event handler tc1_tick:

```
<atlas:TimerControl ID="tc1" runat="server"
                    Interval="5000" OnTick="tc1_Tick">
</atlas:TimerControl>
```

Now, within the tc1_Tick function, you can write a handler for the timer. Additionally, a useful case for this is to use it to feed a ControlEventTrigger trigger type on an UpdatePanel control. Here's an example:

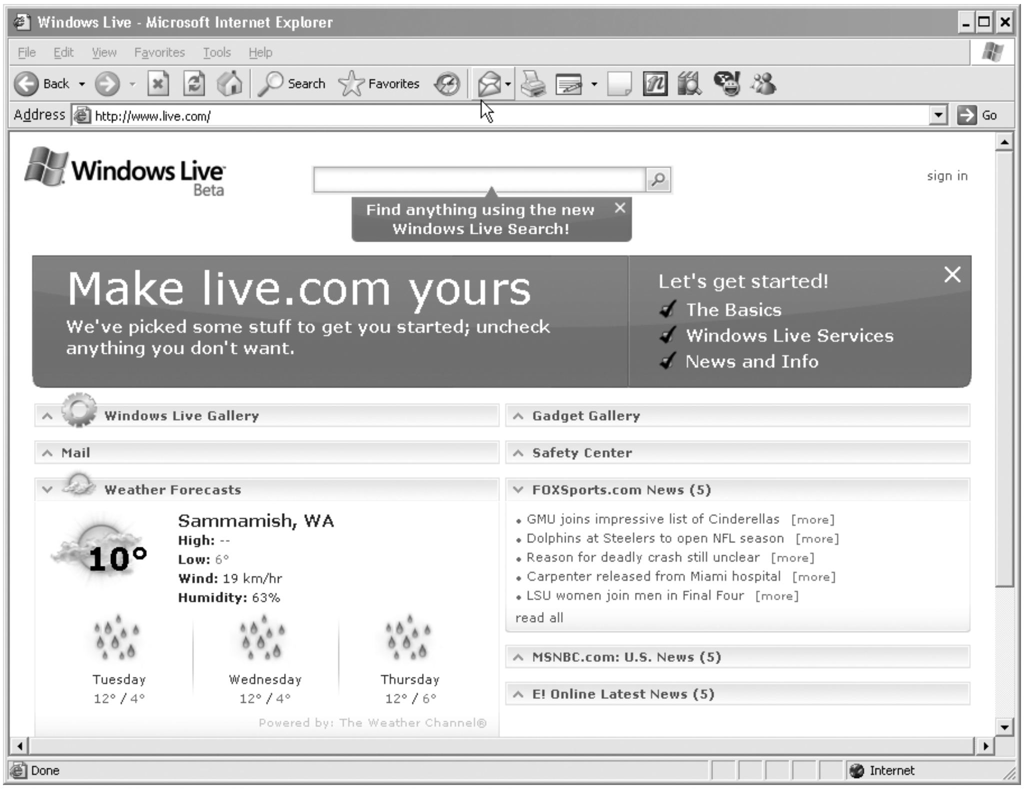
```
<atlas:UpdatePanel ID="UpdatePanel1" runat="server">
<Triggers>
  <atlas:ControlEventTrigger ControlID="tc1"
         EventName="Tick" />
</Triggers>
<ContentTemplate>
  <asp:Label ID="Label1"
             runat="server"
             Text="Label">
  </asp:Label>
</ContentTemplate>
</atlas:UpdatePanel>
```

And now, whenever the timer ticks, the UpdatePanel control will have an update triggered automatically.

Introducing the Gadget Control

Atlas is designed to work with the new technologies that Microsoft is using to extend public portals at Live.com and Start.com. These sites aggregate panes of information on the screen using a technology called *gadgets*.

Figure 6-25 shows the Live.com site with a number of gadgets, including sports headlines and weather.


Windows Live - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.live.com/
Go
Windows Live Beta
sign in
Find anything using the new Windows Live Search!
Make live.com yours
We've picked some stuff to get you started; uncheck anything you don't want.
Let's get started!
The Basics
Windows Live Services
News and Info
Windows Live Gallery
Gadget Gallery
Mail
Safety Center
Weather Forecasts
FOXSports.com News (5)
MSNBC.com: U.S. News (5)
E! Online Latest News (5)
Done
Internet

Sammamish, WA
10°
High: --
Low: 6°
Wind: 19 km/hr
Humidity: 63%
Tuesday 12° / 4°
Wednesday 12° / 4°
Thursday 12° / 6°
Powered by: The Weather Channel®

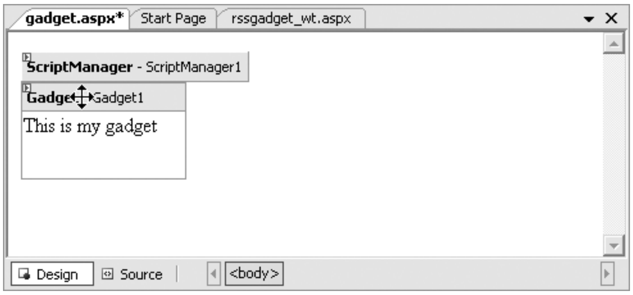
- GMU joins impressive list of Cinderellas [more]
- Dolphins at Steelers to open NFL season [more]
- Reason for deadly crash still unclear [more]
- Carpenter released from Miami hospital [more]
- LSU women join men in Final Four [more]

read all

Figure 6-25. *Live.com showing weather and sports gadgets*

You can use the Atlas Gadget server control to implement your own gadgets for Live.com using Atlas.

Doing this is extremely easy. In Visual Studio .NET, you simply drag the Gadget control onto the design surface for an ASP.NET web form. You'll also need to have a ScriptManager control on the web form. You can see this in Figure 6-26.



This is my gadget

Figure 6-26. *Adding a gadget to an ASP.NET web form*

This generates the following ASP.NET code:

```
<body>
<form id="form1" runat="server">
<div>
 <atlas:ScriptManager ID="ScriptManager1"
                      runat="server"
                      EnablePartialRendering="True">
 </atlas:ScriptManager>
</div>
 <atlas:Gadget ID="Gadget1" runat="server">
  <ContentTemplate>
    This is my gadget<br />
  </ContentTemplate>
  </atlas:Gadget>
</form>
</body>
```

As you can see, as with other controls, a templated area within the control tag specifies what appears at runtime when the control is rendered.

Listing 6-1 shows an example of a full page that hosts a gadget containing an RSS reader. It comes from the Atlas samples download at `http://atlas.asp.net`.

Listing 6-1. *ASP.NET Page Containing RSS Gadget*

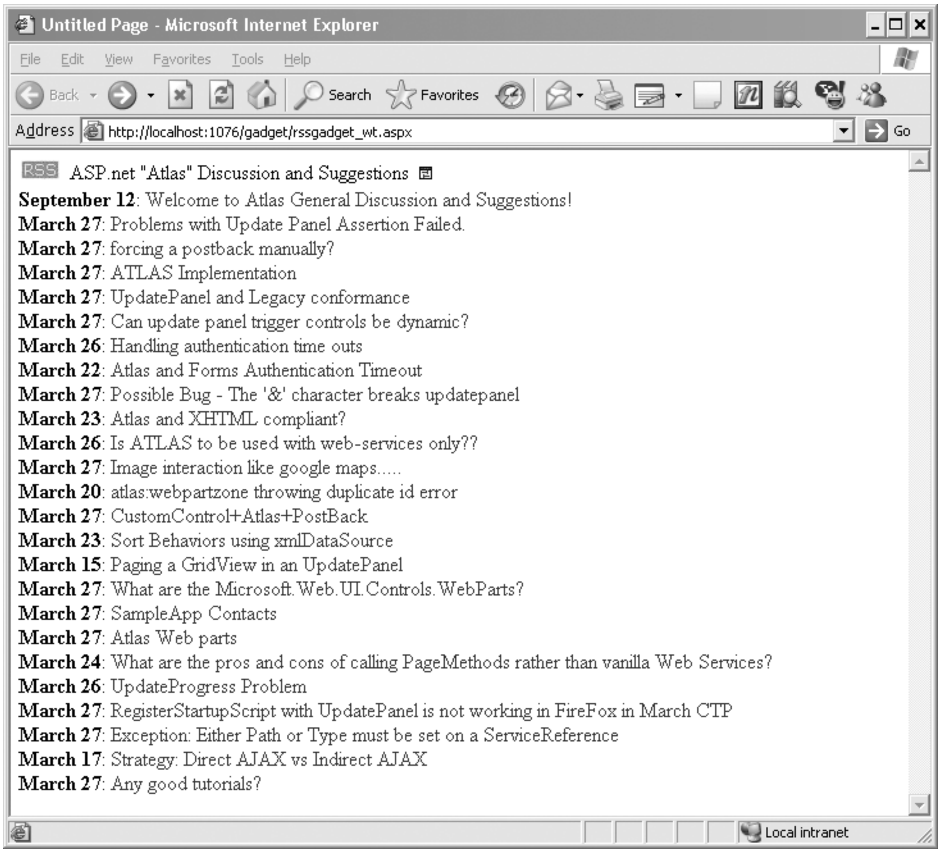
```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
        "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
```

```
<body>
<form id="form1" runat="server">
<div>
 <atlas:ScriptManager runat="server" ID="scriptManager" />

 <atlas:gadget runat="server" ID="AspNetRssFeed"
              Title="ASP.net Forums"
              Description="RSS feeds from ASP.NET forums">
 <ContentTemplate>

    <div style="visibility:hidden;display:none;">
      <div id="RssItemNoDataTemplate">
        <span id="DescriptionLoading">Loading ...</span>
      </div>

     <!-- Layout template for item links -->
     <div id="RssViewLayout">
     <!-- item -->
     <div id="RssItemLayout">
     <div id="RssItemView">
       <span id="RssItemDate"
             style="font-weight:bold;">
       </span>:
       <a target="_blank" href="#"
          id="RssItem" class="ForumLink">
       </a>
     </div>
     </div>
     </div>
    </div>

    <div id="RssViewList" class="Dialog"
         style="position:absolute;width:95%;overflow:auto;">

    <div class="DialogBanner">
      <img src="images/rss.jpg" alt="RSS" />&nbsp;
      <div id="ForumTitle"
           style="display:inline;">
        ASP.net "Atlas" Discussion and Suggestions
      </div>
                     &nbsp;
      <a onclick="javascript: return false;"
         href="#" id="RefreshForum">
      <img alt="Get forum feed"
           src="images/refresh.jpg" /></a>
      </div>
```

```
              <!-- List -->
<div id="RssView"></div>
</div>

<script type="text/xml-script">
<page xmlns:script="http://schemas.microsoft.com/xml-script/2005">
<components>
  <!-- Data -->
  <timer id="sizeTimer" interval="500"
         tick="OnTimerTick" enabled="true" />

  <xmlDataSource id="FeedSource" autoLoad="true"
                 xpath="//item"
                 serviceURL="aspnetforums.asbx?mn=Get">
    <parameters
          feedURL="http://forums.asp.net/rss.aspx?ForumID=1007" />
  </xmlDataSource>

  <!-- Get feed -->
  <button id="RefreshForum">
    <click>
      <invokeMethod target="FeedSource" method="load" />
    </click>
  </button>

  <!-- List of links -->
  <control id="RssViewList" visibilityMode="Collapse" />
  <listView id="RssView"
            itemTemplateParentElementId="rssItemLayout">
  <bindings>
    <binding id="BindListView" property="data"
             dataContext="FeedSource" dataPath="data" />
  </bindings>

  <layoutTemplate>
    <template layoutElement="RssViewLayout" />
  </layoutTemplate>

  <emptyTemplate>
    <template layoutElement="RssItemNoDataTemplate"/>
  </emptyTemplate>
  <itemTemplate>
    <template layoutElement="RssItemView">
    <hyperLink id="RssItem">
     <bindings>
```

```
            <xpathBinding property="text"
                          xpath="./title/text()" />
            <xpathBinding property="navigateURL"
                          xpath="./link/text()" />
           </bindings>
          </hyperLink>
          <label id="RssItemDate">
           <bindings>
            <xpathBinding property="text"
                          xpath="./pubDate/text()"
                          transform="ToDateString" />
           </bindings>
          </label>
         </template>
        </itemTemplate>
        </listView>

        </components>
    </page>
  </script>
</ContentTemplate>

<Scripts>
  <atlas:scriptreference path="rssgadget_wt.js" />
</Scripts>

<Styles>
  <atlas:StyleReference Path="rssgadget.css" />
</Styles>

</atlas:gadget>
</div>
</form>
</body>
</html>
```

Figure 6-27 shows this page in action, reading the RSS string and rendering it using the `<ContentTemplate>` tag.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:1076/gadget/rssgadget_wt.aspx
Go
Local intranet

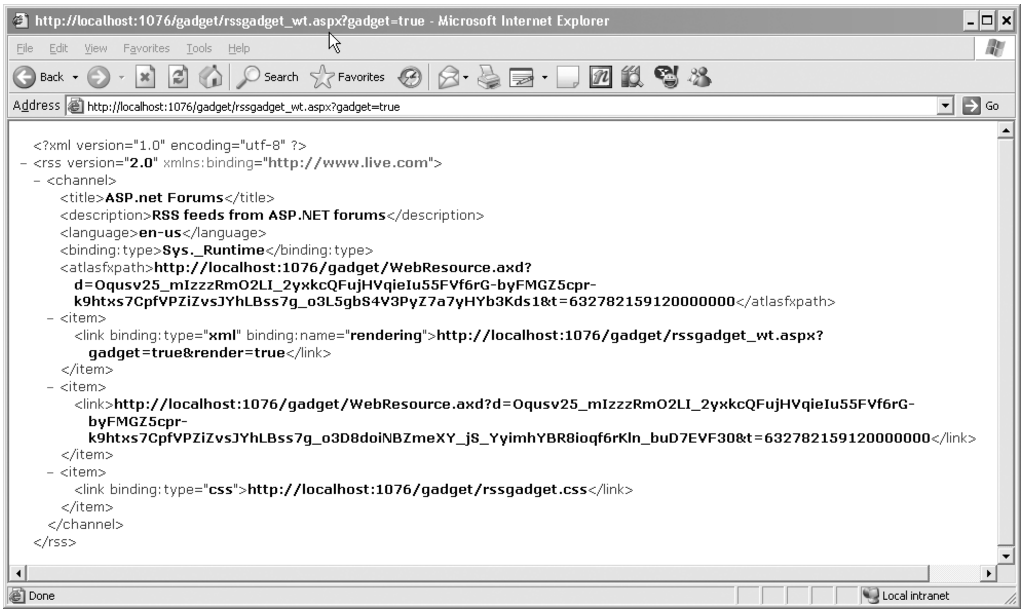
RSS ASP.net "Atlas" Discussion and Suggestions

September 12: Welcome to Atlas General Discussion and Suggestions!
March 27: Problems with Update Panel Assertion Failed.
March 27: forcing a postback manually?
March 27: ATLAS Implementation
March 27: UpdatePanel and Legacy conformance
March 27: Can update panel trigger controls be dynamic?
March 26: Handling authentication time outs
March 22: Atlas and Forms Authentication Timeout
March 27: Possible Bug - The '&' character breaks updatepanel
March 23: Atlas and XHTML compliant?
March 26: Is ATLAS to be used with web-services only??
March 27: Image interaction like google maps.....
March 20: atlas:webpartzone throwing duplicate id error
March 27: CustomControl+Atlas+PostBack
March 23: Sort Behaviors using xmlDataSource
March 15: Paging a GridView in an UpdatePanel
March 27: What are the Microsoft.Web.UI.Controls.WebParts?
March 27: SampleApp Contacts
March 27: Atlas Web parts
March 24: What are the pros and cons of calling PageMethods rather than vanilla Web Services?
March 26: UpdateProgress Problem
March 27: RegisterStartupScript with UpdatePanel is not working in FireFox in March CTP
March 27: Exception: Either Path or Type must be set on a ServiceReference
March 17: Strategy: Direct AJAX vs Indirect AJAX
March 27: Any good tutorials?

Figure 6-27. *Running the RSS reader gadget*

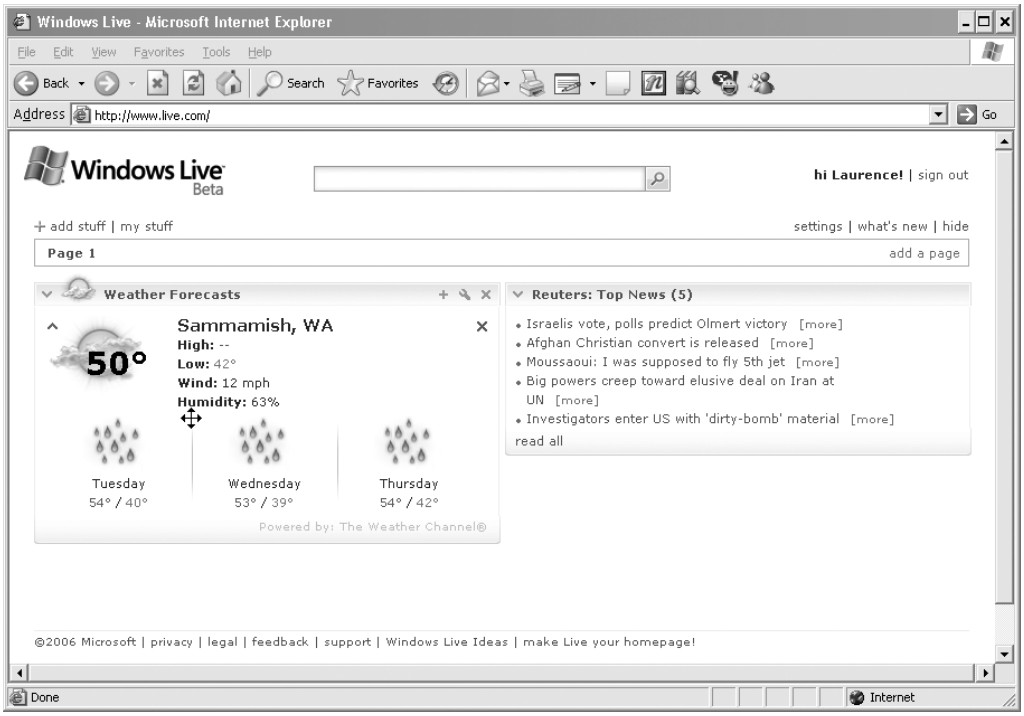
When you run this and append the text ?gadget=true to the URI, you will receive the output shown in Figure 6-28.

Figure 6-29 shows Live.com when you are signed in. You can sign into this site using Microsoft Passport. To add this gadget to Live.com, you use the Add Stuff link at the top left of the page.


http://localhost:1076/gadget/rssgadget_wt.aspx?gadget=true - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:1076/gadget/rssgadget_wt.aspx?gadget=true
Go
Done
Local intranet

```
<?xml version="1.0" encoding="utf-8" ?>
- <rss version="2.0" xmlns:binding="http://www.live.com">
  - <channel>
      <title>ASP.net Forums</title>
      <description>RSS feeds from ASP.NET forums</description>
      <language>en-us</language>
      <binding:type>Sys._Runtime</binding:type>
      <atlasfxpath>http://localhost:1076/gadget/WebResource.axd?
        d=Oqusv25_mIzzzRmO2LI_2yxkcQFujHVqieIu55FVf6rG-byFMGZ5cpr-
        k9htxs7CpfVPZiZvsJYhLBss7g_o3L5gbS4V3PyZ7a7yHYb3Kds1&t=632782159120000000</atlasfxpath>
    - <item>
        <link binding:type="xml" binding:name="rendering">http://localhost:1076/gadget/rssgadget_wt.aspx?
          gadget=true&render=true</link>
      </item>
    - <item>
        <link>http://localhost:1076/gadget/WebResource.axd?d=Oqusv25_mIzzzRmO2LI_2yxkcQFujHVqieIu55FVf6rG-
          byFMGZ5cpr-
          k9htxs7CpfVPZiZvsJYhLBss7g_o3D8doiNBZmeXY_jS_YyimhYBR8ioqf6rKln_buD7EVF30&t=632782159120000000</link>
      </item>
    - <item>
        <link binding:type="css">http://localhost:1076/gadget/rssgadget.css</link>
      </item>
    </channel>
  </rss>
```

Figure 6-28. *Running the Gadget control with Gadget=True set*


Windows Live - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.live.com/
Go
Windows Live Beta
hi Laurence! | sign out
+ add stuff | my stuff
settings | what's new | hide
Page 1
add a page
Done
Internet

Weather Forecasts

Sammamish, WA
50°
High: --
Low: 42°
Wind: 12 mph
Humidity: 63%

Tuesday	Wednesday	Thursday
54° / 40°	53° / 39°	54° / 42°

Powered by: The Weather Channel®

Reuters: Top News (5)

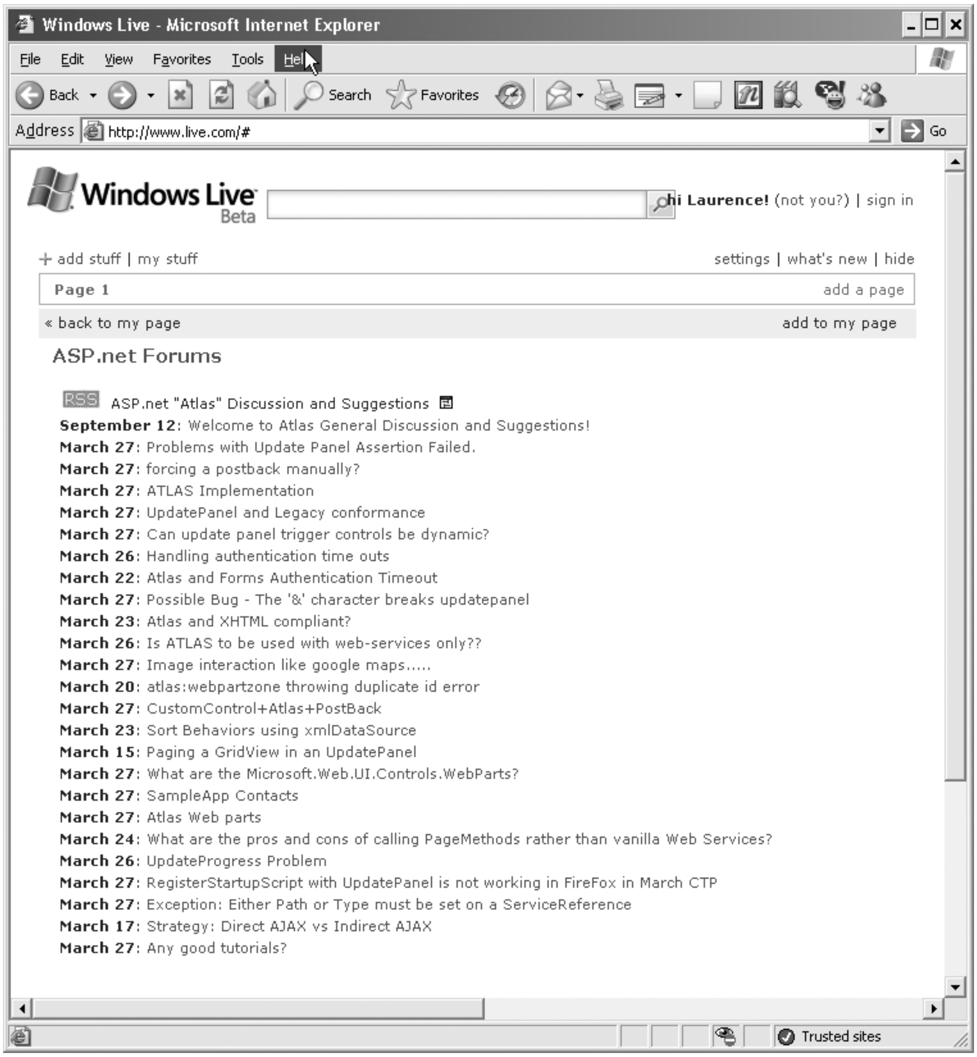
- Israelis vote, polls predict Olmert victory [more]
- Afghan Christian convert is released [more]
- Moussaoui: I was supposed to fly 5th jet [more]
- Big powers creep toward elusive deal on Iran at UN [more]
- Investigators enter US with 'dirty-bomb' material [more]

read all

©2006 Microsoft | privacy | legal | feedback | support | Windows Live Ideas | make Live your homepage!

Figure 6-29. *Signed into Live.com*

You can then use the Advanced options to add the URL of your gadget, and you will see it rendered on your Live.com portal. You can see this in Figure 6-30.


Windows Live - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.live.com/#
Go
Windows Live Beta
hi Laurence! (not you?) | sign in
+ add stuff | my stuff
settings | what's new | hide
Page 1
add a page
« back to my page
add to my page
Trusted sites

ASP.net Forums

RSS ASP.net "Atlas" Discussion and Suggestions
September 12: Welcome to Atlas General Discussion and Suggestions!
March 27: Problems with Update Panel Assertion Failed.
March 27: forcing a postback manually?
March 27: ATLAS Implementation
March 27: UpdatePanel and Legacy conformance
March 27: Can update panel trigger controls be dynamic?
March 26: Handling authentication time outs
March 22: Atlas and Forms Authentication Timeout
March 27: Possible Bug - The '&' character breaks updatepanel
March 23: Atlas and XHTML compliant?
March 26: Is ATLAS to be used with web-services only??
March 27: Image interaction like google maps.....
March 20: atlas:webpartzone throwing duplicate id error
March 27: CustomControl+Atlas+PostBack
March 23: Sort Behaviors using xmlDataSource
March 15: Paging a GridView in an UpdatePanel
March 27: What are the Microsoft.Web.UI.Controls.WebParts?
March 27: SampleApp Contacts
March 27: Atlas Web parts
March 24: What are the pros and cons of calling PageMethods rather than vanilla Web Services?
March 26: UpdateProgress Problem
March 27: RegisterStartupScript with UpdatePanel is not working in FireFox in March CTP
March 27: Exception: Either Path or Type must be set on a ServiceReference
March 17: Strategy: Direct AJAX vs Indirect AJAX
March 27: Any good tutorials?

Figure 6-30. *Adding your gadget to Live.com*

One thing to note is permissions and security. If you have trouble viewing the gadget, make sure you have `*.live.com` and `*.start.com` in your trusted sites settings within IE.

As you can see, with Atlas creating gadgets is easy. You can integrate these gadgets into Live.com, Start.com, and a number of up-and-coming sites. They'll also be able to be integrated directly into the sidebar of Windows Vista!

Summary

This chapter introduced you to the server controls that are available to Atlas programmers. It walked you through using the ScriptManager control, which is at the heart of Atlas. This control empowers the download of the runtime as well as handles things such as error display messages. Additionally, you looked at the UpdatePanel control, which is at the heart of how Atlas enables Ajax functionality in existing ASP.NET pages using partial-page updates.

Other controls such as the Timer control and the UpdateProgress control are available on the server side to make your UI friendlier. Finally, you looked at some of the control extenders, which provide valuable client-side functionality to existing controls. The extenders are important in that they allow you to easily amend your existing ASP.NET applications unobtrusively. To extend ASP.NET controls for drag and drop, for example, you then simply add an extender to the page and point it at that control.

This chapter gave you a high-level overview of each control and how it works. In the next chapter, you will look at some applications and samples that use this functionality, as well as at the client-side controls you saw in Chapters 4 and 5, dissecting them to understand how you can program similar applications of your own in Atlas.