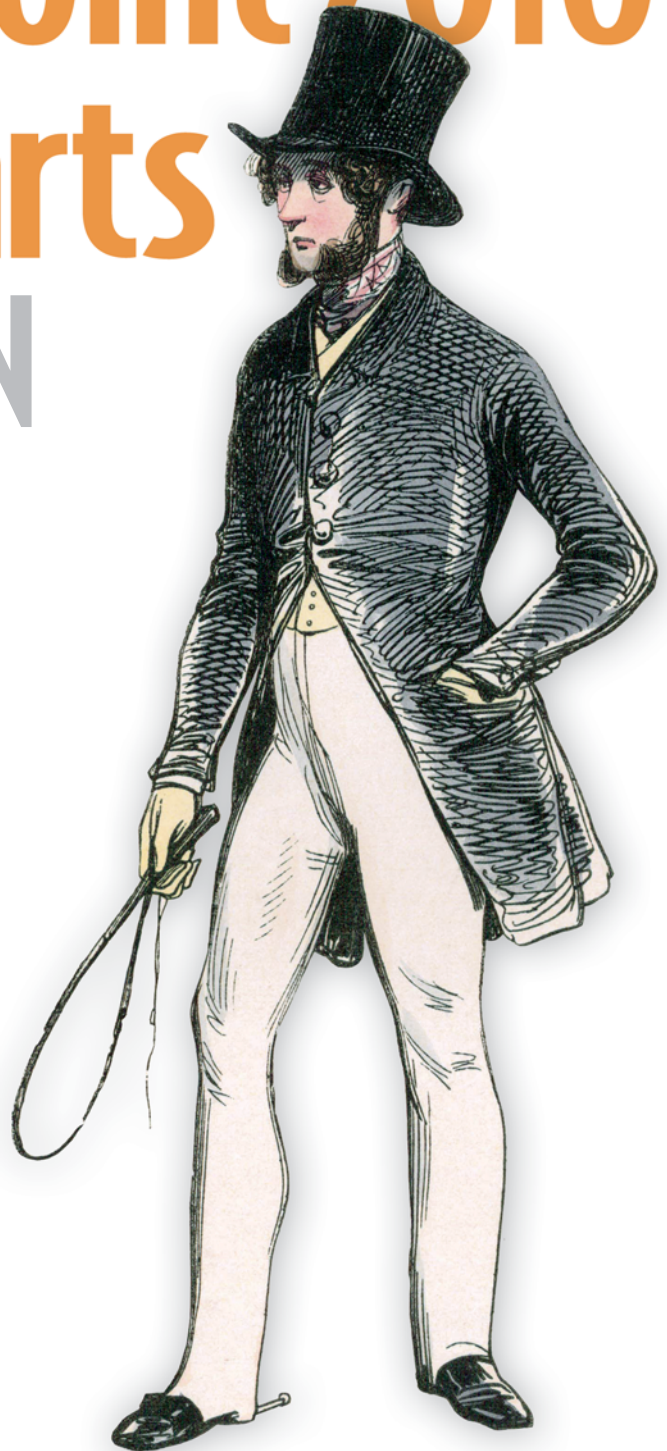


# SharePoint 2010 Web Parts IN ACTION

Wictor Wilén

SAMPLE CHAPTER





***SharePoint 2010  
Webparts in Action***

Wictor Wilén

Chapter 3

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 CONNECTIONS .....333**

14 ■ Connecting Web Parts 335

15 ■ Building pages and dashboards 359

## Part 2

# *Developing SharePoint 2010 Web Parts*

**P**art 2 covers 11 full chapters of Web Part development topics. In part 1, you learned what Web Parts are, how you can use them, and how you can configure applications with them. But, you'll eventually end up with a scenario requiring you to build your own custom Web Part—and that's why you're reading this book.

We'll start from the ground up by introducing you to Visual Studio 2010 and the SharePoint Development Tools. You'll learn how to build Web Parts using best practices to avoid falling into traps, in many cases, using standard ASP.NET techniques. So if you've been building ASP.NET applications and controls, you'll find quite a lot of common tasks and I'll show you the differences.

The longest chapter of this book, chapter 5, will walk you through the various ways that you can make your Web Parts customizable. Next, in chapter 6 you'll learn how to localize your Web Parts and use resources such as JavaScripts and images. An essential part of SharePoint development is how you package and deploy your solutions, and, in chapter 7, I'll show you just that.

Sooner or later you'll get to the point that your Web Part and solution misbehave, and then it's time to troubleshoot those problems. Chapter 8 is all about troubleshooting and how to avoid running into these situations. Once your Web Part is up and running, you may need to fine-tune it for performance. Chapters 10 and 11 focus on the new features introduced in SharePoint 2010, such as integration with the Ribbon menu and other dynamic features of the SharePoint web interface.

Silverlight is a key component of the Microsoft web technologies, and you'll learn how Silverlight can be used in combination with Web Parts. Chapter 12 gives you insight into how to turn your Web Parts into mobile-friendly components so that your users can access on any devices the functionality that you build. Chapter 13 discusses design patterns; you'll see how to build robust and testable Web Parts using proven patterns and techniques.

I hope you have a great time reading through this part of the book and enjoy the code samples. You can download most of the code samples from the book's website at <http://www.manning.com/wilen> and use them as a reference for your own projects.

# *Building Web Parts with Visual Studio 2010*

---

## ***This chapter covers***

- Using Visual Studio 2010
- Taking advantage of SharePoint Developer Tools
- Building Visual Web Parts
- Building traditional Web Parts

You should now have a fairly good understanding of what Web Parts are and how end users will work with them. It's time to start building some of your own. Previous versions of SharePoint lacked the tools and applications to create a good development experience, which made it hard to be productive. You had to use tools provided by the community, create your own, or do a lot of manual editing of XML-based manifests and configuration files. The new SharePoint Developer Tools that ship with Visual Studio 2010 solve these problems and allow you to focus on the fun part: development.

Developing Web Parts for SharePoint 2010 isn't just about wiring up components in a control that you can then use to build pages or applications in SharePoint as you might do when working in ASP.NET applications. Solutions built for SharePoint need

a few extra steps before they can be invoked in the host environment. This chapter shows you how to build, package, and deploy SharePoint Web Part solutions using the new development tools for SharePoint 2010 in Visual Studio 2010. These tools allow you to focus on programming tasks rather than packaging the solution.

We'll begin by reviewing the requirements for the development environment. Then, I'll walk you through two ways of building Web Parts; Visual Web Parts and traditional Web Parts. During these walkthroughs, we'll look into the various files and artifacts that your SharePoint Web Part solution consists of in Visual Studio and explore the features of the development tools. We'll also discuss the extensibility of Visual Studio and the development tools. The last section of this chapter will explain how to upgrade solutions built for earlier SharePoint versions to SharePoint 2010.

### **3.1 Requirements for your development environment**

Before we start with the programming tasks, you need to prepare your development workstation. Earlier versions of SharePoint required that you work directly on a Windows Server machine with SharePoint installed. This was because it wasn't possible to install SharePoint on a Windows client operating system (OS) in a supported way. You could build and package your solutions on a client workstation and then later manually add the solution to the server for testing. But, this procedure was unsupported, took a lot of time, and didn't allow you to debug your solution. And it's something I really can't recommend.

SharePoint 2010 has support for installation on a 64-bit Windows 7 or 64-bit Windows Vista Service Pack 1 machine so that you can develop, build, test, and debug directly on your normal client OS. This strategy is good for some scenarios, such as showing demos, testing, and small development tasks. But I recommend that you invest in hardware that allows you to run virtual machines (VMs) in which you have your SharePoint 2010 installation, Visual Studio 2010, and other tools. VMs allow you to create an environment that looks similar to your production environment, and you can more easily create new machines when necessary. VMs also let you take snapshots of your environment, which allows you to roll back the server when you mess up your machine instead of having to reinstall your client OS. You need 64 bit-capable hardware with enough RAM to host your VMs. The recommended minimum amount of RAM is 4 GB for a development machine. Even though it might work pretty well, I recommend that you use 8 GB on your client. This allows you to have a local client installation of SharePoint Foundation and enough memory to spin up a VM. It's the speed of the disks that matters (along with the amount of RAM). Hosting VMs on external USB drives isn't recommended. If you can convince your boss to invest in one or more solid-state disks, you can achieve great performance on a laptop.

Your development workstation or VM should have at least SharePoint Foundation 2010 or SharePoint Server 2010 installed along with Visual Studio 2010 and SharePoint Designer 2010. SharePoint Designer isn't a necessity when developing for SharePoint but, as you've seen in previous chapters, it can add great value for rapid application development. To get started quickly without setting up an environment

of your own, download the evaluation image from Microsoft at [www.microsoft.com/downloads/](http://www.microsoft.com/downloads/) and search for “2010 Information Worker Demonstration and Evaluation Virtual Machine.”

### 3.2 *Developing for SharePoint 2010 in Visual Studio 2010*

Visual Studio 2010 is the primary development environment that you’ll use to build your Web Parts and other SharePoint 2010 projects. You can use other environments or tools, but the SharePoint Developer Tools for Visual Studio 2010 will save you a lot of time and allow you to be more productive than before. The Developer Tools are a new addition to the Visual Studio suite, even though there were extensions for Visual Studio 2005 and 2008 that you could download. The new SharePoint Developer Tools are targeted for SharePoint 2010 development only.

**NOTE** There were extensions for SharePoint 2007 (Microsoft Office SharePoint Services [MOSS] 2007 and Windows SharePoint Services [WSS] 3) called Visual Studio Extensions for Windows SharePoint Services (VSeWSS) that could be downloaded to Visual Studio 2005 and, then, 2008. These extensions were criticized and never exited the Community Technology Preview (CTP) stage. For SharePoint 2007 development, the community moved faster than Microsoft and created great tools that offered more flexibility than VSeWSS. The two most popular community contributions were STSDev (<http://stsdev.codeplex.com>) and WSPBuilder (<http://wspbuilder.codeplex.com>).

The SharePoint Developer Tools in Visual Studio allow you to edit and configure your SharePoint solutions using visual designers instead of working with the XML files directly. You can just hit F5 to run and debug your project, just like any other solutions built with Visual Studio. In the following sections, we’ll walk through the initial SharePoint 2010 development experience with Visual Studio 2010 and look at the various types of projects that you can create.

Visual Studio 2010 comes in many flavors, ranging from the free edition, called Visual Studio 2010 Express, to the top-notch edition, called Visual Studio 2010 Ultimate (which contains everything you need from design, development, testing, and life cycle management). All editions (except for the free Express edition) have the SharePoint Developer Tools.

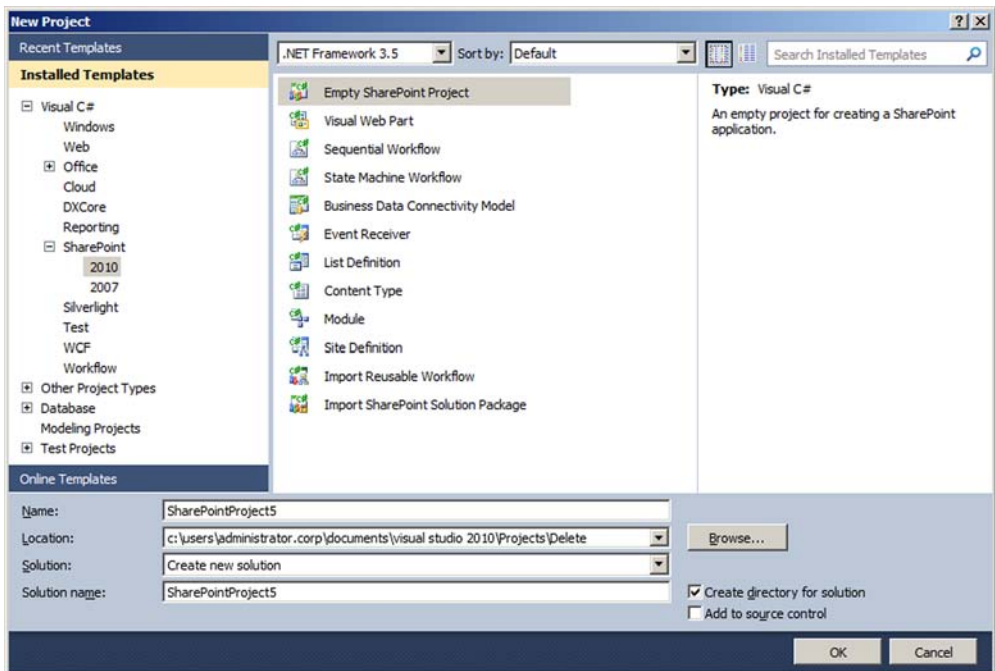
You need to run Visual Studio 2010 as administrator if you have the User Account Control enabled on your server or workstation—deployment and debugging need full access to the processes and the files in SharePoint root.

To create a project in Visual Studio, always start with a project template. The SharePoint Developer Tools install a set of templates for SharePoint projects, and we’re going to use some of them throughout this book.

The SharePoint Extensions for Visual Studio 2010 come with a set of project templates for SharePoint 2010 development. They range from an empty project to specialized projects like Web Parts, lists, and workflows. To create a SharePoint project, you need to start a new project in Visual Studio by selecting File > New > Project. The New Project dialog box will display all the available project templates.

The first time you open this dialog box, select the target framework for your project. The default framework in Visual Studio is version 4.0, but SharePoint 2010 is based on Microsoft .NET 3.5 Service Pack 1, so select version 3.5 as your target framework. Visual Studio will change your project so that .NET Framework 3.5 is used even if you select version 4 when you create SharePoint projects. You change the target framework using the dropdown list in the New Project dialog box. After selecting the framework, look at the SharePoint templates in the template category selector on the left, as shown in figure 3.1. Select Visual C# > SharePoint > 2010 to view the available templates for SharePoint 2010 development. Depending on your Visual Studio edition and the extensions you've installed, the templates will vary. Because SharePoint is based on Microsoft .NET, you can use the language of your choice, such as Visual Basic, F#, or C++. If you prefer to use Visual Basic, select Visual Basic > SharePoint > 2010 to display those templates, which are exactly the same as for C#. This book will use C# in all code and samples. The reason for choosing C# is that it's currently the most popular .NET programming language and you'll find most samples and demos on the Internet and that sites such as CodePlex use it.

In the New Project dialog box, specify the name of your project, the solution, and the location in the file system. A Visual Studio 2010 solution is a collection of projects; you can add more projects later. Don't confuse the Visual Studio solution with a SharePoint 2010 solution: each project based on the SharePoint templates will result



**Figure 3.1** The New Project dialog box in Visual Studio 2010 shows the available SharePoint 2010 templates and includes a brief description of the selected template.

in one SharePoint solution package. You can't create a SharePoint project using the Visual Studio 2010 SharePoint Developer Tools on a machine that doesn't have SharePoint 2010 installed.

The default templates in Visual Studio 2010 for SharePoint 2010 development are listed in table 3.1. When working with Web Parts, you'll mainly use the Empty SharePoint project and the new Visual Web Part template—which is the one you'll use when building the first Web Part in this book.

**Table 3.1 All out-of-the-box SharePoint 2010 templates in Visual Studio 2010**

Template Name	Description
Blank Site Definition	Creates a site definition.
Business Data Connectivity Model	Creates a model for the Business Connectivity Services.
Content Type	Creates a content type.
Empty SharePoint Project	Creates an empty SharePoint 2010 solution in which you can add items and Features.
Event Receiver	Creates an event receiver.
Import Reusable Workflow	Imports workflows from SharePoint Designer 2010.
Import SharePoint Solution Package	Imports WSP files that have been exported from SharePoint 2010.
List Definition	Creates a list definition.
Module	Creates a module.
Sequential Workflow	Creates a sequential workflow.
Site Definition	Creates a site definition project.
State Machine Workflow	Creates a state machine workflow.
Visual Web Part	Creates a project with a Visual Web Part item.

As you can see, you have a lot of different options and project templates to select from. Choose the appropriate one to get an easy start; you can easily add project items later to add more functionality. You can even create your own templates or download new ones using the Visual Studio Extension Manager.

### 3.3 Building your first Visual Web Part

Previous developer tools for SharePoint lacked a good visual designer environment. Visual Studio 2010 contains a new project item called *Visual Web Part*. To illustrate how a Visual Web Part works (and how the SharePoint Developer extensions in Visual Studio 2010 work), I'll guide you through your first Visual Web Part project. We'll create a Visual Web Part project with a Visual Web Part that takes text input and displays it in the Web Part when you click a button. The following sections will show you how to create and build the project and then eventually deploy the solution and use the Web

Part by adding it to a page. When you're creating a project for SharePoint, Visual Studio will guide you through the setup using a wizard. During this first Web Part project, we'll explore the other files that are created within your SharePoint project.

### 3.3.1 *The Visual Web Part template*

When creating a solution in Visual Studio, you always start with a project template. The first template that we're going to use is the *Visual Web Part* template. Visual Web Parts employ a user control to build the user interface. The user control is loaded by the Web Part class during runtime and rendered. Using a user control enables you to use a visual designer in Visual Studio to design and build your Web Part. This allows for more rapid development and increased productivity. If you're an ASP.NET developer, you'll be familiar with the Visual Studio interface and this method of building Web Parts.

The disadvantage of using Visual Web Parts is that these Web Parts are dependent on files that live in the SharePoint Root in the file system. These files are loaded and compiled dynamically during runtime, which isn't allowed in Sandboxed mode. (You'll learn more about Sandboxed mode in chapter 7.) The Visual Web Parts also require that you write some plumbing code when writing custom Web Part properties (see chapter 5) because you have two classes that need to be updated: the Web Part class and the control template class.

**NOTE** The Microsoft SharePoint Tools team has released a set of extensions for Visual Studio 2010 that's available for download using the Extension Manager in Visual Studio. In the extension manager, search for "SharePoint Power Tools" to find the extension. These tools contain a sandbox-compatible Visual Web Part.

First, you need to create a new SharePoint 2010 project using the Visual Web Part template in Visual Studio, as described earlier. Make sure that you're using Microsoft .NET 3.5 as your target framework.

#### **Naming projects in Visual Studio**

You should carefully name your projects when working with SharePoint. Do this for other projects as well. With a good naming convention, it's easier to locate the source of any failures during troubleshooting. A good convention is to use your company name and the name of the project.

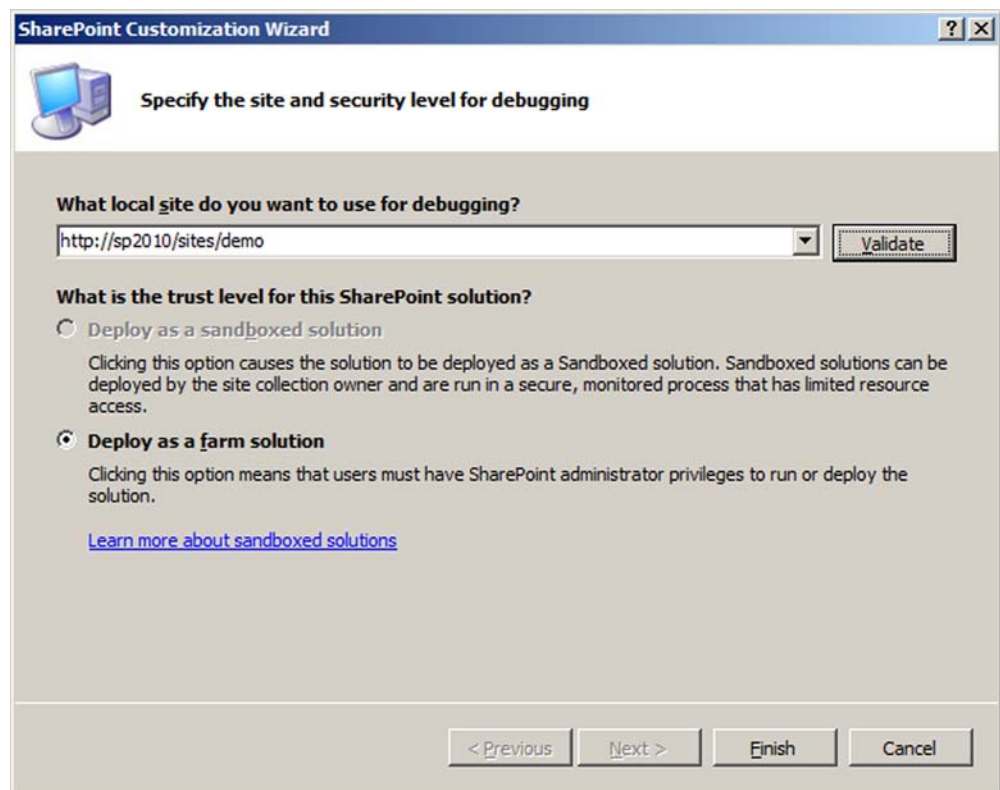
Throughout this book I'll use the following standard: `WebPartsInAction.ChNN.<Project name>`, where `NN` is replaced by the chapter number and `<Project name>` with the actual project we're creating.

Our first project is going to be called `WebPartsInAction.Ch3.VisualWebPart`. Note that, when you enter a project name in the Name field, your Visual Studio solution

will be assigned the same name automatically. If you plan to create several projects within your solution or if you prefer to have another name for the solution, you can change the solution's name by entering a separate name for it in the Solution Name field. Another benefit of using a naming convention is that the default namespace in your project will use the name of the project. When you've selected the template and specified a project name, click OK; the SharePoint Customization wizard appears.

### 3.3.2 The SharePoint Customization wizard

The SharePoint Customization wizard will appear for all SharePoint 2010 templates in Visual Studio 2010 and have different configuration steps depending on the chosen template. The first step in the wizard asks you which site you'd like to use for debugging and what trust level you'd like to have for the solution, as shown in figure 3.2. The site URL you enter must be a local site; this is also the site that's going to be started when you debug your solution and the site in which your solution will be deployed.



**Figure 3.2** The SharePoint Customization wizard when creating a Visual Web Part; note that the Deploy as a sandboxed solution option is disabled.

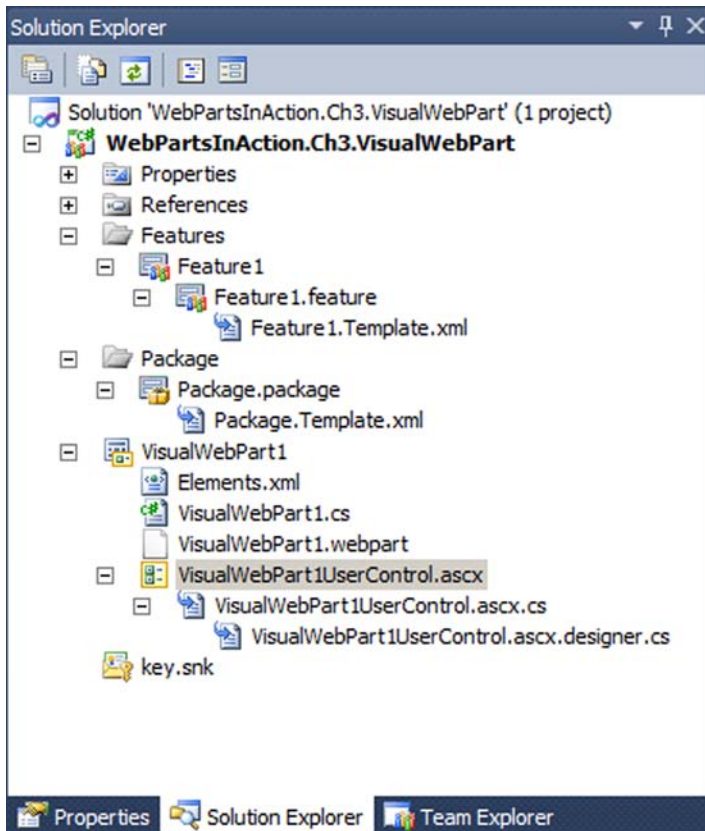
You have two trust levels to choose from:

- *Sandboxed solution*—The solution will run in a secure, protected, and monitored process; chapter 7 will show you more details.
- *Farm solution*—The solution will run as a fully trusted component (in the global assembly cache) in the farm.

Because this is a Visual Web Part project, you don't have an option to select the trust level; it has to be a Farm solution, also known as a full-trust solution. To end the wizard, click Finish, and Visual Studio will create the solution, project, and project items for you.

### 3.3.3 Explore the SharePoint Project Items

Once you finish the wizard, you're taken to the Visual Studio environment and your project is created. When creating a Visual Web Part, Visual Studio automatically creates a Package, a Feature, and a Visual Web Part for you. You can browse these items using the Solution Explorer in Visual Studio; click View > Solution Explorer if you don't see the Solution Explorer in your environment. Figure 3.3 shows the Solution Explorer after we've created a Visual Web Part and, as you can see, it contains several SharePoint project items, nodes, and source code files.



**Figure 3.3**  
The Solution Explorer in Visual Studio 2010 allows you to navigate the files in your project.

**NOTE** The items in a Visual Studio 2010 SharePoint project are also called SharePoint project items (SPIs). For example, a Visual Web Part is a SPI, even though it contains several files.

#### THE PACKAGE PROJECT ITEM

The *Package*, found under the Package root node in your project, describes the SharePoint files, assemblies, and Features included in the solution package. At some point, Visual Studio will create a Windows SharePoint Package (WSP) file of your Package, based on what's defined in the Package editor. The resulting file is a cabinet (CAB) file with all the files you need for your solution. Chapter 7 will tell you more about the WSP file. You'll also use the Package editor when you need to include references to other assemblies, as you'll see in later chapters.

The package project item consists of an XML file called the package manifest, and you edit it using a designer or manually by writing XML code. When editing the package project item manually, you work with a special file called `package.template.xml`, which is, during the packaging process, merged with the actual package manifest.

#### THE FEATURE PROJECT ITEM

Visual Web Part projects always include a Feature, found under the Feature node in the project. A *Feature* is a set of components, definitions, and actions that are deployed onto a specified scope. Each Package can contain several Features. Visual Studio has a designer that you can use to edit the Feature in the same way you do with the Package. You can also edit the underlying XML manually by using a template file, as you do with Packages. The designer can be used to add items to or remove them from your Feature. When we build the next Web Part, we'll take a closer look at the Feature designer.

#### THE VISUAL WEB PART PROJECT ITEM

The Visual Web Part project item, located in the project root, consists of no more than six files. First you have the user control, which is the file ending with `.ascx`. This is the control that the Web Part will load and display, and it contains all controls and events. The user control also makes this Web Part editable by the Visual Studio control designer, which allows you to visually design the control. The user control also has two code-behind files: one that you'll use for your custom code-behind code and one that's used by the Visual Studio designer to define the controls, properties, and events that you edit using the designer. The Visual Web Part has another code file—`VisualWebPart1.cs`, as you can see in figure 3.3; this is where the Web Part class is defined and this Web Part class loads the user control.

**NOTE** Partial class definitions were introduced in C# 2. Using the `partial` keyword on classes allows the class to be split over several files. Visual Studio uses partial classes to separate the code generated by the designers and the code that the developer edits. For instance, the user control in the Visual Web Part is split into two files: one for you to add logic to and one appended with `designer.cs`, which is used by the Control designer.

The Feature needs to know how to add the Web Part to SharePoint. You specify this information in the `elements.xml` file, called the elements manifest, which you see in the following code snippet. This file contains instructions to SharePoint to add the Web Part to the gallery as well as its name and group. The elements manifest also references the Web Part controls description file (`.webpart`), which contains the reference to the Web Part class and assembly as well as its default property values. This is the file that's added to the Web Part Gallery.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/" >
  <Module
    Name="VisualWebPart1"
    List="113"
    Url="_catalogs/wp">
    <File
      Path="VisualWebPart1\VisualWebPart1.webpart"
      Url="WebPartsInAction.Ch3.VisualWebPart_VisualWebPart1.webpart"
      Type="GhostableInLibrary" >
      <Property
        Name="Group"
        Value="Custom" />
    </File>
  </Module>
</Elements>
```

The elements manifest file has a `Module` element that specifies the destination of the items. The `File` element identifies the source of the Web Parts control description file to upload using the `Path` attribute and specifies the destination using the `Url` attribute. If you change the name of your Web Parts control description file, you need to manually update the `Path` property to reflect your change.

To specify properties for the file in the Web Part Gallery, you use the `Property` element. It's important to specify into which categories the file will be placed in the Web Part Gallery; you do this by editing the `Group` property. By default, Visual Studio 2010 adds files to the `Custom` category. You should change this setting to something related to your solution. Here, we're changing the value from `Custom` to `My Web Parts`:

```
<Property Name="Group" Value="My Web Parts" />
```

The Web Parts control description file that SharePoint creates for you looks like the following listing.

### Listing 3.1 The default Web Part control description file for a Visual Web Part

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="WebPartsInAction.Ch3.VisualWebPart.VisualWebPart1.
        VisualWebPart1,$SharePoint.Project.AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;
    </importErrorMessage>
```

**Declares  
class and  
assembly**

1

```

</metaData>
<data>
  <properties>
    <property name="Title" type="string">VisualWebPart1</property>
    <property name="Description" type="string">
      My Visual WebPart
    </property>
  </properties>
</data>
</webPart>
</webParts>

```

Sets property  
default values

2

The default Web Part control description file contains two interesting parts. One is the `type` element **1**, which has a `name` attribute. The `name` attribute specifies the Web Part's class, including the namespace, followed by a special Visual Studio 2010 parameter that's replaced with the full assembly name during the packaging process. If you change the name or namespace of your Web Part, you must manually change it here.

The second part of the Web Parts control description file is the `properties` element, in which you set the default values of the Web Part's properties. By default, Visual Studio adds the `Title` and `Description` properties **2**. Change the values of these property elements so that they look like this:

```

<property name="Title" type="string">My Visual WebPart</property>
<property name="Description" type="string">
  Visual Web Parts rock!
</property>

```

When this Web Part is deployed to the Web Part Gallery, it'll have the title and description as specified here. The title in the Web Part header will also use the `Title` property as the default value.

**TIP** In chapter 2, you learned how to export a Web Part. You can save some XML typing if you deploy your Web Part and configure it using the web interface. Using the `export` verb, you can download a `.webpart` file from which you just copy and paste the properties instead of manually typing them into the file. Make sure that you don't overwrite the `metaData` tag or the Visual Studio token used for the full assembly name. You'll learn more about the contents of this file in chapter 5.

### 3.3.4 Adding functionality to your Visual Web Part

The Visual Web Part you just created contains no controls or content by default, but you can add them by double-clicking on the Visual Web Part item in the Solution Explorer (or expand the item and click on the user control file). Visual Studio will open the User Control designer in Source mode. You can switch between Design and Source mode using the buttons in the lower-left corner of the user control file. You can also use Split mode, which allows you to edit the source and directly see the preview in the designer, or vice versa.

To add controls to the user control, you have two options. You can use the Toolbox, found under `View > Toolbox`, to drag and drop controls to your user control. The

drag-and-drop approach works for both the Design and the Source modes. The second option is to write the control code in the Source editor. If you've worked with Visual Studio before, you'll be familiar with this way of working.

#### ADDING CONTROLS TO THE VISUAL WEB PART

In this first sample, you're going to add three controls: a `TextBox`, a `Label`, and a `Button` control. These are standard ASP.NET controls, and you'll find them in the Standard group in the Toolbox. Drag and drop the controls into your user control. Then, add a `<br/>` tag after each control so that it looks a bit nicer. If you switch to Design mode, you'll immediately see a preview of how your Web Part will look. Note that the preview won't take themes or SharePoint CSS into consideration, but you'll get a fair view of how you've organized your controls.

Note that you should use the closed line break tag `<br/>` instead of `<br>`. This closed tag must be used to comply with the XHTML standard that's used in the SharePoint 2010 interface.

#### ADDING EVENTS TO THE VISUAL WEB PART

Our Web Part will take the value from the `TextBox` when the user clicks the button and will write that value into the label, so we need to add an event to the button control. To add the click event, switch to Design mode and double-click on the button. Visual Studio will then automatically create a new method in the code-behind of the user control and wire that method to the control by adding an attribute to the control tag in the user control source. Once this is done, Visual Studio will open the code-behind file and place your cursor at the new method. Add the code to the new method that takes the value from the `TextBox` and wires it into the label. The following snippet shows what the click event method should look like after you add the code. The method takes the `Text` property from the `TextBox` and assigns it to the `Text` property of the label:

```
protected void Button1_Click(object sender, EventArgs e) {
    Label1.Text = TextBox1.Text;
}
```

The next snippet shows the markup source of the user control after you add the event to the button. The control, import, and register directives are omitted before the first line and the `Label` control has been manually modified so that it has an empty string as the default value for its `Text` property. You can see the automatically added `onclick` event, which is hooked to the newly created method in your code-behind:

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<br/>
<asp:Label ID="Label1" runat="server" Text=""></asp:Label>
<br/>
<asp:Button ID="Button1" runat="server" Text="Click me!"
    onclick="Button1_Click"/>
```

← Attribute added  
automatically

#### EXAMINING THE WEB PART CLASS

Before you build and deploy your Visual Web Part, let's see how the Web Part invokes the user control. The Visual Web Part project item consists of a Web Part class file that

contains the implementation of the class that inherits from the `System.Web.UI.WebControls.WebParts.WebPart` class.

The first thing to notice is a constant definition that points out the location of the user control file that's used. This location points to the `TEMPLATECONTROLTEMPLATES` folder in the SharePoint Root folder; that's where the user control file is placed during deployment.

```
private const string _ascxPath =
    @"~/_CONTROLTEMPLATES/VisualWebPart1/
    ➔ VisualWebPart1/VisualWebPart1UserControl.ascx";
```

The `CreateChildControls` method is automatically created for you by Visual Studio. This method will load the user control using the constant and add the control to the control tree of the Web Part class.

```
protected override void CreateChildControls() {
    Control control = Page.LoadControl(_ascxPath);
    Controls.Add(control);
}
```

If you look at the class definition, you'll notice that the Web Part inherits from the ASP.NET Web Part class. You can change which class the Web Part inherits from if you're using another base class or would like to use the SharePoint Web Part implementation.

### 3.3.5 Build and deploy the Visual Web Part

The SharePoint Developer Tools for Visual Studio 2010 use the same approach as any other Visual Studio project, so you can just press F5 to run and debug your project. But, before you do that, I'll guide you through the details of what happens when you press F5 or select `Debug > Start Debugging`.

First, Visual Studio compiles your project, which means that Visual Studio uses the correct language compiler and builds a .NET assembly file (a DLL file) of your source code. During the compilation, any warnings and errors will be reported in the Visual Studio Output and Error List windows. In this example, the Web Part class and user control code-behind files are compiled. Any errors in the user control markup source (the .ascx file) won't be detected; this file will be compiled during runtime. For example, if you add the button click event manually and incorrectly spell your event method in the user control source, the compiler won't detect the mistake.

After the compilation phase, Visual Studio needs to package the solution into a format that SharePoint understands: a Windows SharePoint Package (WSP). This file contains all the necessary files to deploy your solution into SharePoint and includes the compiled assembly, your Feature definitions, .webpart files, user controls, and other resources. You can manually invoke packaging by selecting `Build > Package` in Visual Studio.

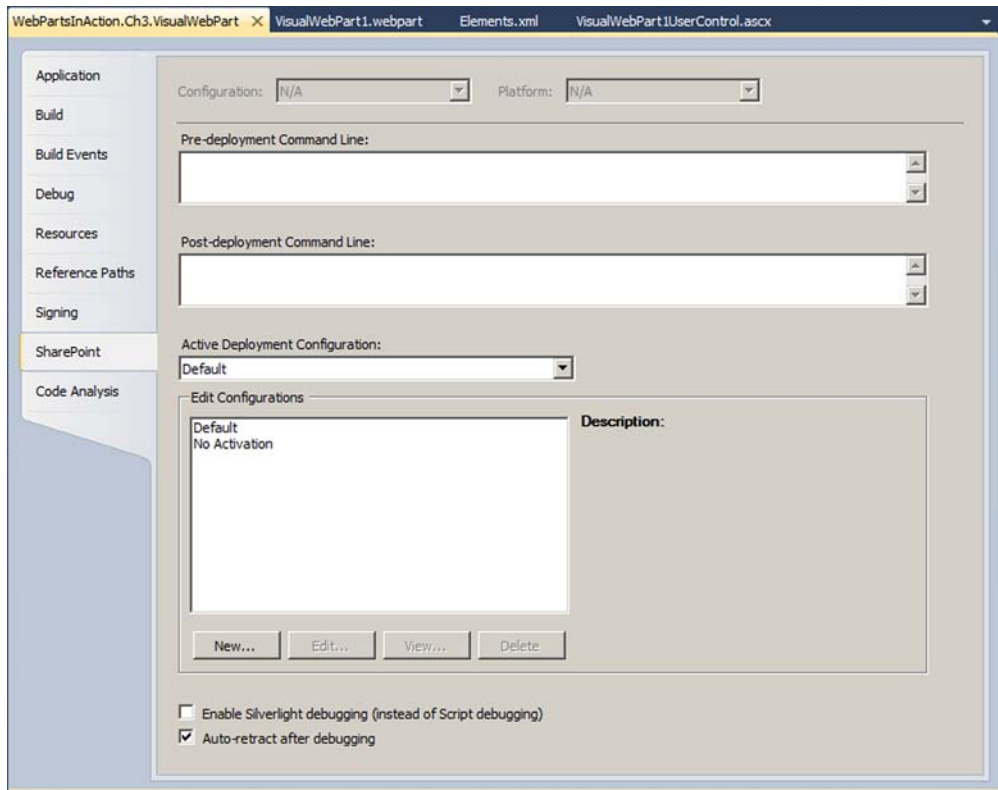
The SharePoint Developer Tools contain a set of deployment configurations that are invoked after the compilation and packaging processes when you press F5 to debug your solution. To open the properties for your project, do one of the following:

- Double-click the Properties item in your project.
- Right-click the project name and select Properties.
- Click the Properties button.

As shown in figure 3.4, SharePoint projects have a special tab for configuring the deployment configuration. Visual Studio 2010 comes with two configurations out of the box:

- *Default*—Retracts the old solution, adds the new one, and activates the Features.
- *No Activation*—Retracts the old solution and adds the new one without activating the Features.

You can choose which deployment configuration to use in your project and even *create* your own. Each configuration contains a number of deployment steps; you can watch the steps by selecting a configuration and clicking View. Each configuration has two sets of steps: one for deployment and for retraction. You might want to change from the Default configuration to the No Activation configuration if you'd like to manually activate the Feature, perhaps when you need to debug or test just that activation situation. At the bottom of the SharePoint Configuration tab is a check box called



**Figure 3.4** A SharePoint project has a special properties tab for selecting and editing deployment configurations.

Auto-retract After Debugging. This check box is selected by default, which means Visual Studio will always retract your solution when you're done with debugging. I find it convenient to deselect this check box to avoid retracting the package when going in and out of the debugging mode.

Depending on your solution and your needs, you can easily create your own deployment configuration by clicking the New button and then selecting the steps you want to use for deployment and retraction. You can also specify a command-line action to invoke before and after deployment. If you have specific needs, you can create custom deployment steps with a Visual Studio extension; see section 3.8 for more information.

You can manually deploy a project or solution by selecting Build > Deploy Solution and retract it by selecting Build > Retract. It's during the deployment step that Visual Studio will use the URL you specified while creating the project in the SharePoint Customization wizard. That URL will be the target of your deployment. Once you've deployed your solution without any errors and warnings, you can go to the site and test it.

During the deployment, Visual Studio also checks for any conflicts of Features and files. Conflicts may occur, for example, when you're building Web Parts and you give the Web Parts in different projects the same name. In a conflict, Visual Studio will detect that a .webpart file with that name exists in the gallery and will remove the previous one. As a general rule, try to name your Web Parts control description files appropriately so you don't encounter conflicts. Visual Studio automatically resolves conflicts with Web Parts in the gallery by removing the old .webpart file and adding the new one, but for other conflicts, such as list instances, Visual Studio will instead ask you for your input on how to resolve the conflict.

To perform all these steps with just one keystroke, press F5. Visual Studio 2010 will then do all these steps for you and, after the deployment, start Internet Explorer and browse to the site to which you deployed the solution. Finally, Visual Studio attaches the debugger to the ASP.NET worker process and Internet Explorer so that you can debug your solution.

### **3.3.6 Take your Web Part for a test drive**

To run your Web Part for the first time, just press F5. It will take a few seconds for Visual Studio 2010 to perform the build and deploy the Web Part. Once it's ready, you'll find yourself on the SharePoint site that you configured in the wizard. If this is the first time you've used Visual Studio to debug the application, you'll see a dialog box that asks if you want to enable debugging on the website. Click Yes to configure the web.config file so that you can debug your project. Visual Studio will automatically launch Internet Explorer with the welcome page of the site specified in the SharePoint configuration wizard. Choose Site Actions > New Page to create a new page where you'll host your deployed Web Part or use an existing page. The page will open in Edit mode.

To add your Web Part to the page, position the cursor in the wiki content area and then use the Ribbon to open the Web Part Gallery. Select the My Web Parts category that you defined in the elements manifest. In that category, you'll find the Web Part called VisualWebPart1. Select it, click Add, and then save the page. Now, test the Web Part by making an entry in the text box, as shown in figure 3.5, and clicking the button.



**Figure 3.5** The first Visual Web Part project that prints the value of the text box in a label when the button is clicked

If you go back to Visual Studio 2010 without closing Internet Explorer, you can navigate to the code-behind file for your user control and set a breakpoint in the `Button1_Click` method where you assign the value to the label. Do so by clicking in the gray area in the left of the Source Code editor or by pressing F9. Then, return to Internet Explorer and once again click the button. Visual Studio will interrupt the execution of the program when it hits the breakpoint, and you can debug your solution just as you would any other .NET-based application. Visual Studio automatically attaches to the IIS Worker Process, `w3wp.exe`, for debugging the compiled code and to Internet Explorer for script debugging when you press F5.

If you want to run your solution without any attached debugger, you can press Ctrl-F5 or choose Debug > Start Without Debugging. Visual Studio will then perform all building and packaging and finally start Internet Explorer but without attaching the debugger.

SharePoint 2010 needs a lot of resources, such as RAM and CPU. If you don't have a powerful machine, the deployment and debugger attachment process can take a while. You have a couple of options to speed up this process. First, you can run your solution without attaching the debugger by pressing Ctrl-F5. A second option is to manually start Internet Explorer and use the Deploy Solution command to test your solution. If you need to debug your session after using one of these methods, you can always attach the Visual Studio debugger to the IIS Worker Process manually.

### 3.4 Traditional Web Part projects

The traditional way of building Web Parts didn't involve any visual designers, so we had to build our Web Parts and control trees manually. The Visual Web Part has drastically changed the way we develop Web Parts and has helped us to be more productive. But, the Visual Web Part has some disadvantages that can't be overlooked. For example, Visual Web Parts can't be run in Sandboxed mode (which isn't a huge issue thanks to the downloadable SharePoint Power Tools mentioned earlier), and they require some extra plumbing when you're working with Web Part properties. The Visual Web Parts also have a slightly larger memory footprint than a traditionally built Web Part, which can be important in large scenarios and highly scalable solutions. Also, whenever you need to subclass another Web Part, you have to use traditional Web Parts.

Traditional Web Parts are built in nearly the same way as Visual Web Parts except that you need to programmatically add all controls and you don't have the visual aids. I'll walk you through building the same Web Part you built as a Visual Web Part to

illustrate the differences in developing with the visual aids and without. This technique can also be used in combination with the Visual Web Part or even when you combine several user controls into a single Web Part.

### 3.4.1 Create an empty SharePoint project

There are no out-of-the-box templates for creating Web Part projects, but Web Parts can be added to any SharePoint project in Visual Studio. For a simple Web Part project, you should start with the *Empty SharePoint Project* template. This template contains no items or Features when created. When you've selected the template, give it the name `WebPartsInAction.Ch3.WebPart1` and click OK to create the project. The SharePoint Customization wizard will start, just as with the Visual Web Part. The big difference here is that you can select which trust level you'd like to use for your SharePoint solution—the Visual Web Part only offered the option of creating fully trusted solutions. With the empty Web Part project, you can also choose to deploy your solution to the protected and monitored SharePoint sandbox process.

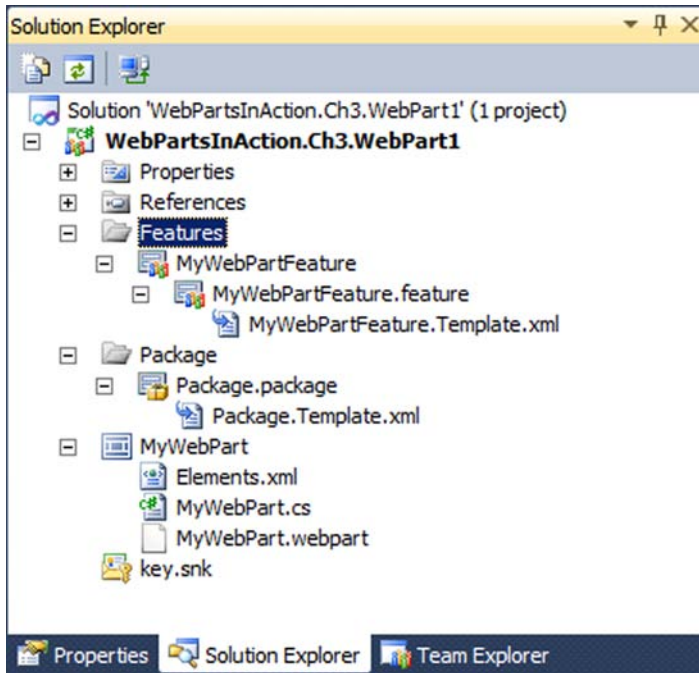
This time you're going to deploy your solution to the sandbox. Begin by selecting *Deploy As Sandboxed Solution*, enter the name of the site to use for debugging, and click *Finish*. The *Sandboxed Solution* option in the SharePoint Customization wizard is set as the default option. This is an indication on how Microsoft wants us to develop SharePoint solutions and I agree that, whenever it's possible, the sandbox should be used. When building solutions for Office 365 and SharePoint Online, *Sandboxed Web Parts* is the only to go. You'll learn more about this as we move along in this book.

### 3.4.2 Adding the Web Part to the project

As I mentioned, when your empty SharePoint project is created, it doesn't contain any Features or items, just a Package. To add a Web Part, click *Project > Add New Item* and browse to the *SharePoint > 2010* template folder. There, you need to select the Web Part project item and give it a name. Always try to add items using a meaningful name instead of the suggested name so you don't have to go through your files and classes renaming things. Name this Web Part item `MyWebPart`. Renaming an item can be quite painful because it can involve modifying several files and references.

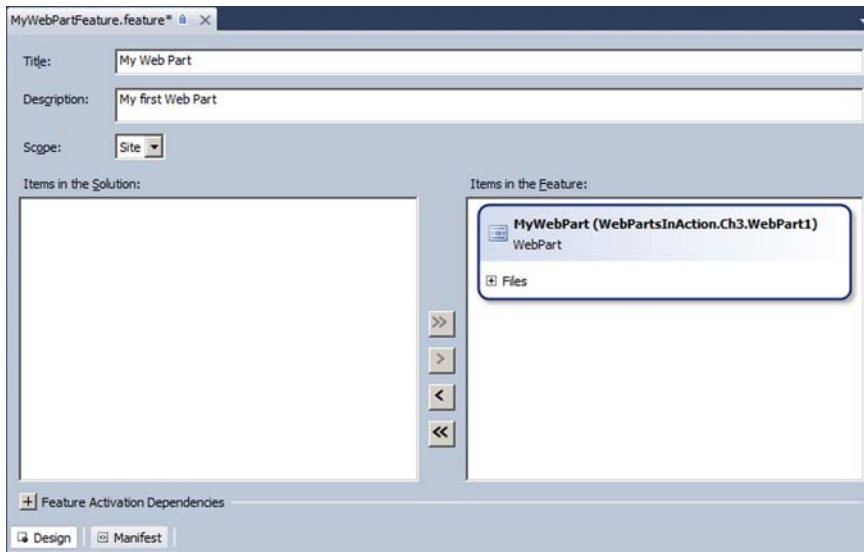
When you've added the Web Part item to the project, Visual Studio will add the Web Part class, the Web Parts control description file (`.webpart`), and a Feature elements manifest file (`elements.xml`). These files are grouped together in the Solution Explorer. Visual Studio also adds a new Feature to your project, which references the project item that includes the `elements.xml` and the `.webpart` file that's connected to the Web Part, as you can see in figure 3.6.

At any time, you can add more Web Parts or other items to the project as long as you don't violate the restrictions Visual Studio enforces when you use a sandbox-targeted project, which will be discussed in chapter 7. For instance, you can't add a Visual Web Part to this current project.



**Figure 3.6** The Solution Explorer in Visual Studio shows you all the files and Features of your Web Part project. The Web Part files are grouped together into a SharePoint project item.

Rename the Feature to `MyWebPartFeature` by right-clicking the Feature, selecting `Rename`, and typing a name. If you want, double-click on the Feature you just renamed and the Feature designer will open in Visual Studio, as shown in figure 3.7.



**Figure 3.7** Use the Feature designer in Visual Studio to edit your Features. You can change the title, description, and scope as well as configure which items should be part of the Feature.

You can use the designer to give the Feature a title and description that will be shown to web interface users. The Feature designer also contains functionality for changing the scope of your Feature. A Web Part project must always have the scope set to Site, which means that it's deployed to a site collection because the Web Part Gallery exists at the site collection level. Using the Feature designer, you can also add SharePoint items to or remove them from the Feature. You'd do this, for example, if you have multiple Features in your solution and you'd like to have different Web Parts in the different solutions. For now, though, let's leave our Feature as is. When you've changed the title and description, save and close the Feature designer.

### 3.4.3 Adding controls to the Web Part

The Web Part class file is automatically opened in Source Code view when you add the Web Part item to the project. Because this project item doesn't support any visual editing, you only have this view of the Web Part. The class file looks similar to the corresponding Visual Web Part class except for the code that loads the user control.

**NOTE** For the code samples in this book, I've stripped out most namespace declarations and all using statements.

You have to add controls to your Web Part manually, using the `CreateChildControls` method (except in certain scenarios that we'll cover in later chapters). The following listing shows what our code looks like after we've added a text box, a label, and a button.

#### Listing 3.2 Adding controls to the `CreateChildControls` method

```
protected TextBox textBox;  
protected Label label;  
protected Button button;  
  
protected override void CreateChildControls() {  
    textBox = new TextBox();  
    label = new Label();  
    button = new Button();  
  
    button.Text = "Click me!";  
  
    this.Controls.Add(textBox);  
    this.Controls.Add(new LiteralControl("<br/>"));  
    this.Controls.Add(label);  
    this.Controls.Add(new LiteralControl("<br/>"));  
    this.Controls.Add(button);  
  
    base.CreateChildControls();  
}
```

1 Defines controls

2 Instantiates controls

3 Adds controls to control tree

First, you define the text box, label, and button controls as protected local variables 1. These controls are created in the `CreateChildControls` method 2, and then you add them to the Web Part controls collection in the order that you want your controls to appear 3.

Use the `LiteralControl` control to add the `<br/>` tags that you used in the Visual Web Part to write each control on a new line. This control is great for creating HTML

markup that doesn't need any connected control. For example, if you need to write HTML content in the Web Part, you can use the `LiteralControl` and build a string with HTML instead of using several ASP.NET controls. The more controls you use, the more processing is needed. In addition, the view state of the page may grow larger.

### 3.4.4 Adding the button-click event manually

Because you can't take advantage of the designer and automatically hook up the control events, you have to manually add the click event to the button. The event is wired up to the button control after the control is created in `CreateChildControls`. Using IntelliSense in Visual Studio helps you write the code. When you type `+=` after the `Button.Click` event and press the Tab key, Visual Studio writes the code for you. If you press Tab a second time, it also creates the event handler method skeleton for you. The source code row you wrote should look like this:

```
button.Click += new EventHandler(button_Click);
```

And the automatically created event handler looks like this:

```
void button_Click(object sender, EventArgs e) {  
    throw new NotImplementedException();  
}
```

Replace the exception with the assignment of the text box text to the label just as we did in the Visual Web Part example. To complete your Web Part, your event handler code should look like this:

```
void button_Click(object sender, EventArgs e) {  
    label.Text = textBox.Text;  
}
```

As in the Visual Web Part example, you can change which base class the Web Part will use here but, because this sample is deployed to the sandbox, you can't use a base class that inherits from the SharePoint implementation. Doing so will result in a runtime error when you add the Web Part to a page, even though the compilation will work fine.

### 3.4.5 Testing and debugging your Web Part

To run your Web Part, press F5 (just like you did with the Visual Web Part), and SharePoint 2010 will open if the compilation succeeds. Any compilation or packaging errors will be reported and stop the debugging process.

Once Internet Explorer has opened the SharePoint site, create a new page using the Site Actions menu. Or, you can edit the page you used earlier by clicking the Edit Page icon, inserting the Web Part on the page, and verifying that it's working.

Because you just created a sandboxed solution, this Web Part is running in the isolated sandbox process. Visual Studio deployed it to a special gallery in the site collection called the Solution Gallery. Click Site Actions > Site Settings and then click Solutions under the Galleries header to access the Solutions Gallery. This gallery

holds the WSP files that can be used within the site collection, and they're all running in the sandbox. Verify that your newly created solution is there. Visual Studio attaches the debugger to Internet Explorer and to the SPUCWorkerProcess.exe process (instead of the IIS Worker Process) that's the host for the sandbox.

**NOTE** In chapter 7, you'll learn more about sandboxed solutions and the Solution Gallery.

### 3.5 Upgrading SharePoint 2007 projects

You've just seen how to build Web Parts in two different ways. But, what if you've already worked with SharePoint 2007 and created solutions for it using Visual Studio or other tools? Then you can, in most cases, use these solutions as-is in SharePoint 2010. SharePoint 2010 is mostly backward compatible so, just like any other project, you should be sure to test your solutions thoroughly before deploying them into production. If you need to update or edit the projects, you should move the solutions to a SharePoint project in Visual Studio 2010. If you need to upgrade your solutions, there are several methods you can use.

#### 3.5.1 Upgrading from Visual Studio Extension for WSS 1.3

For SharePoint 2007, Microsoft has a VSeWSS extension for Visual Studio 2008. Projects created with this tool aren't directly compatible with the new developer tools in Visual Studio 2010. But, the SharePoint Developer Tools for Visual Studio 2010 contain a migration tool for this specific situation. This tool shows up as a new SharePoint project template called Import VSeWSS Project and lets you import the old project file. Because the structure of the new tools for SharePoint 2010 differs a lot from VSeWSS, you should thoroughly test your new project. The tool is provided by Microsoft but has no official support.

**NOTE** The VSeWSS Import Tool for Visual Studio 2010 can be downloaded from <http://code.msdn.microsoft.com/VSeWSSImport>. The tool is distributed as source code and needs to be compiled before usage.

#### 3.5.2 Upgrading from other project types

If you have previously used tools like STSDev or WSPBuilder, then there are no tools for importing these solutions to a SharePoint Project in Visual Studio 2010. I suggest that you start a new project and then manually add each item from the old project to the new one. This will ensure that your solution will work with Visual Studio 2010. Another approach might be the Import SharePoint Solution Package project, which allows you to import a WSP file into a new Visual Studio project. This approach doesn't import the source code from the assemblies, but it might give you a faster way to start migrating larger solutions.

If you're trying to upgrade your project so that you can use it to upgrade an already existing solution in SharePoint, you have to carefully export the solution manifest and, most importantly, the solution id. You must manually add and edit the

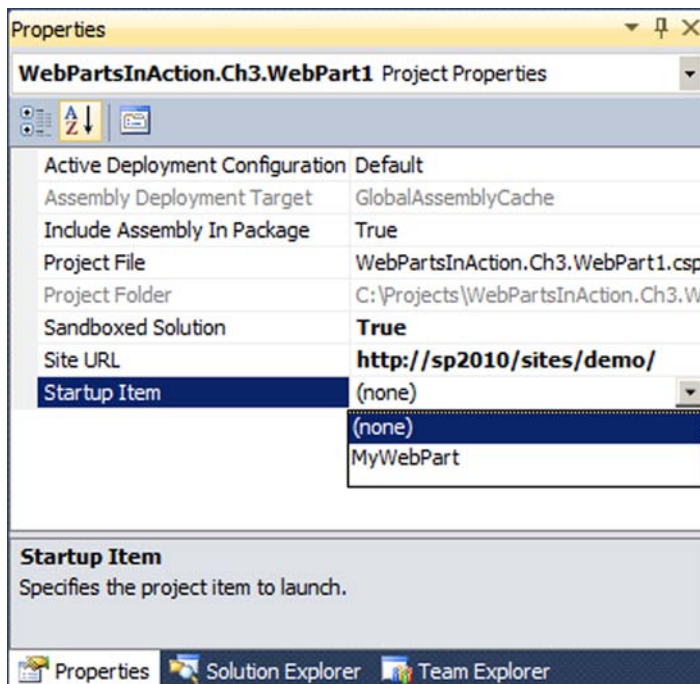
manifest in the Visual Studio 2010 SharePoint project. You have to choose this approach to upgrade your solution if you have a SharePoint 2010 installation that's upgraded from SharePoint 2007.

### 3.6 *SharePoint Project settings in Visual Studio*

When you created the two previous Web Parts, you used the SharePoint Customization wizard and it asked you for a specific URL and a deployment type (Sandbox or Farm solution). What do you do if you switch servers or discover that the SharePoint sandbox is too limited for your use? These settings are handled and stored by the Visual Studio project and can be changed at any time. You can access and change the settings by selecting the project in the Solution Explorer and then pressing F4 to open the Properties pane, as shown in figure 3.8. Don't confuse this with the project properties that you open when you double-click the Properties item or right-click on the Project and select Properties in the Solution Explorer.

Using this toolbox window, you can change the URL of the site collection you're using for debugging purposes by modifying the Site URL property. You can change the deployment type from fully trusted to sandboxed solutions by modifying the Sandboxed Solution property, which you set to True if it will be deployed to the sandbox.

Startup Item is a special property that, in an empty SharePoint project, has the value of (None) by default. But, as soon as you add the Web Part item to the project, you're able to change this property to the name of your Web Part or any other SharePoint items in the project. Selecting a Web Part project item here makes the browser start at



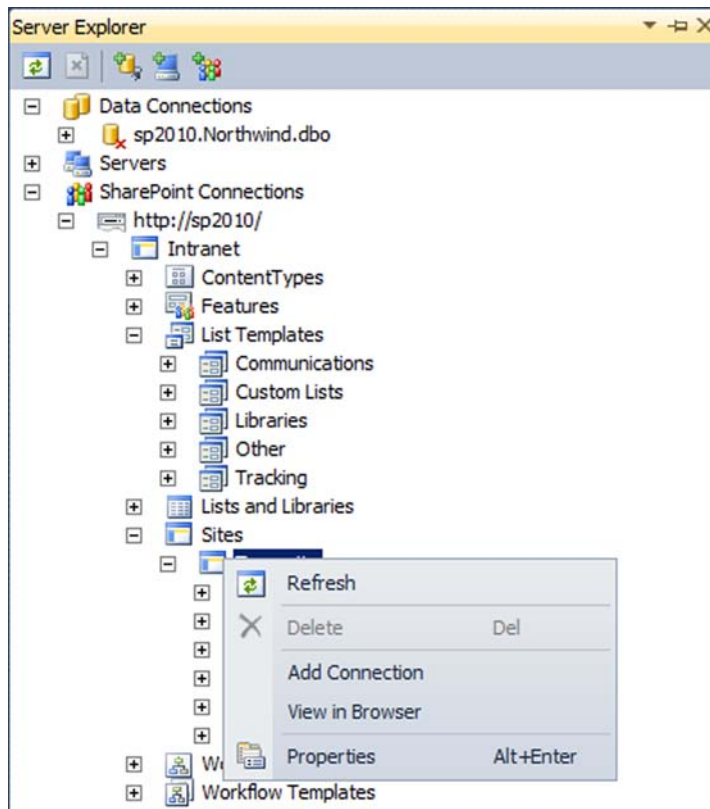
**Figure 3.8** The Project Properties window shows the debugging URL and whether the project is a sandboxed solution. You can also use this window to specify which of the SharePoint project items you'd like to use when debugging the project.

the default URL. But, when you choose a List instance item, for example, the debugging session will start at the URL of the List instance.

### 3.7 SharePoint Server Explorer in Visual Studio

In this chapter, we've looked at several improvements for SharePoint developers, and there are still some more that can make the development experience even better. The Server Explorer in Visual Studio 2010 lets you connect to a local SharePoint site if the SharePoint Developer Tools are installed. Open the Server Explorer by selecting View > Server Explorer. Using this window, you can add new SharePoint sites so that you can browse the details of the site and its subsites and perform actions on the various items. To add a new SharePoint site to the Server Explorer, right-click on SharePoint Connections and select Add Connection (see figure 3.9).

Figure 3.9 shows the Server Explorer with the Sites node expanded. You can use the Server Explorer to get information about the various items in a SharePoint site, for example, to determine the id of a field or a content type. If you have a site or a list selected, you can right-click the node and choose View In Browser to open your web browser and view the item. By default, the Server Explorer provides just read-only



**Figure 3.9** Use the Server Explorer in Visual Studio 2010 to explore your SharePoint site collection and see the contents within it.

property views of the items, but the extensibility options of Visual Studio 2010 allow you or third parties to customize the nodes, insert new nodes, and add functionality.

### 3.8 *Extensibility in Visual Studio 2010*

Visual Studio 2010 has been designed so that it can be extended in various ways to improve your productivity and interactivity. You might encounter situations when you need to customize the SharePoint development tools with new project templates or project items or create extensions for the deployment process.

The set of project templates and project item templates in Visual Studio can be customized to fit your needs. Perhaps you've created a base class from which to derive your Web Part projects and you've customized the elements manifest and Web Parts control definition files. You can create an extension to the SharePoint Developer Tools using Visual Studio with your customizations and share it with your colleagues. You can also extend the Server Explorer with new functionality or information.

The extensions that you create are built using a special project template called the VSIX project in Visual Studio. The result is a VSIX file that you can install on any machine with Visual Studio 2010. You can also upload your extensions to the Visual Studio online gallery to share your creative ideas with the community. When you're creating a project in Visual Studio, select Online Templates in the New Project dialog box and search for available extensions and templates. Enable, disable, or uninstall your extensions using the Tools > Extension Manager in Visual Studio.

**NOTE** The Community Kit for SharePoint—Development Tools Edition is a Visual Studio 2010 extension that's available for download (<http://cksdev.codeplex.com> or [www.communitykitforsharepoint.org](http://www.communitykitforsharepoint.org)). These extensions contain numerous improvements to the SharePoint Developer Tools, and I highly recommend that you install and use these extensions. For example, with these extensions, you can browse the Web Part Gallery and copy Web Part configurations.

### 3.9 *Summary*

This concludes our walkthrough of your first Visual Web Part and traditional Web Part using Visual Studio 2010. This chapter explored the two ways of creating Web Parts and introduced the SharePoint Developer Tools in Visual Studio 2010. You learned the build, package, and deploy routines used when working with SharePoint, both to a fully trusted location and into a sandbox. In the next chapter, we'll continue to build Web Parts and take a deeper look on how to build the user interface.

# 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](http://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 Mastyskarz  
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

