Pro SharePoint 2003 Development Techniques

Nikander Bruggeman and
Margriet Bruggeman

Apress®

Pro SharePoint 2003 Development Techniques

ISBN-13 (pbk): 978-1-59059-761-3

ISBN-10 (pbk): 1-59059-761-3

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

Lead Editor: Jonathan Hassell
Technical Reviewer: Scot Hillier
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: Tracy Brown Collins
Copy Edit Manager: Nicole Flores
Copy Editor: Jennifer Whipple
Assistant Production Director: Kari Brooks-Copony
Production Editor: Laura Cheu
Compositor: Susan Glinert Stevens
Proofreader: Dan Shaw
Indexer: Michael Brinkman
Artist: April Milne
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/Download section.

CHAPTER 1

Incorporating .NET 2.0 in SharePoint

Microsoft SharePoint Products and Technologies has been around for a couple of years now and has proven to be very useful for companies around the world that are implementing portal, team collaboration, or enterprise content management strategies.

The development landscape has changed considerably since SharePoint Products and Technologies was introduced in 2003, largely influenced by the release of Microsoft .NET Framework 2.0. This chapter discusses how to incorporate the exciting new features of .NET 2.0 into SharePoint Products and Technologies.

First, the chapter discusses the service packs for Office SharePoint Portal Server 2003 and Windows SharePoint Services, the two products that make up SharePoint Products and Technologies. We cover installation of these as well as ASP.NET 2.0 and how to use data stores.

This chapter also discusses how to create web parts with Visual Studio .NET 2005 and extend your development possibilities with its Guidance Automation Toolkit (GAT).

To finish this chapter we will show you how to incorporate the new ASP.NET 2.0 server controls within web parts.

SharePoint Products and Service Packs

SharePoint Products and Technologies is made up of two different products: Office SharePoint Portal Server 2003 and Windows SharePoint Services. At the end of 2005, Microsoft released Service Pack 2.0, two service packs for SharePoint Portal Server 2003 and Windows SharePoint Services. These two service packs contain four new enhancements. The following three enhancements apply to both SharePoint Portal Server and Windows SharePoint Services:

- *The ability to run on Microsoft Windows Server 64-bit versions*: SharePoint Portal Server and Windows SharePoint Services both support being run in Windows on Windows 64 (WOW 64) 32-bit emulation mode on 64-bit versions of Windows, although there are no performance improvements. However, other applications and the operating system itself might be able to take advantage of the 64-bit server functionality.

- *Use of SQL Server 2005*: Microsoft SQL Server 2005 includes features such as extended XML support and integrated Common Language Runtime (CLR) support. Windows SharePoint Services and SharePoint Portal Server 2003 do not take advantage of that new functionality. However, there are general benefits to be gained in upgrading to SQL Server 2005, such as better performance.
- *Improved extranet deployment options*: In Windows SharePoint Services and SharePoint Portal Server some of the hyperlinks within web pages and e-mail messages are absolute URLs. Earlier releases of Windows SharePoint Services generated those absolute URLs by using the protocol scheme, host, and port of the web request that Windows SharePoint Services received, or by using the base URL of the site, which could prevent Windows SharePoint Services from supporting certain advanced extranet scenarios where a reverse proxy server is deployed in front of the server running Windows SharePoint Services. This is solved by the new service packs.

The fourth enhancement, support for the Microsoft .NET Framework 2.0 Common Language Runtime and ASP.NET 2.0, only applies to Windows SharePoint Services. SharePoint Portal Server 2003 (with or without Service Pack 2.0) will not support .NET Framework 2.0. The only kind of support you have in SharePoint Portal Server 2003 for .NET Framework 2.0 is the ability to call .NET 2.0 web services and redisplay pages that are written in ASP.NET 2.0. Developing web parts in Visual Studio .NET 2005 is only possible for Windows SharePoint Services with Service Pack 2.0 installed.

The support in Windows SharePoint Services for ASP.NET 2.0 does not include integration with the ASP.NET 2.0 web part Framework. If you deploy a web part that is built in ASP.NET 2.0 to a Windows SharePoint Services server, it will function as a normal web form control.

Although Windows SharePoint Services with Service Pack 2.0 runs on the new Common Language Runtime, it is not redesigned to take advantage of the new features of ASP.NET 2.0. For example, there is no support for master pages in Windows SharePoint Services. On the other hand, the new class libraries found in ASP.NET 2.0 can be used within custom SharePoint web parts.

After applying Windows SharePoint Services Service Pack 2.0, the SharePoint worker process can run in ASP.NET 2.0 mode, but you can also choose to run in ASP.NET 1.0 mode. You will only be able to use the new features and the security and performance enhancements of ASP.NET 2.0 when Windows SharePoint Services runs in ASP.NET 2.0 mode. You can run ASP.NET 1.x and 2.0 at the same time on a server if you are using different Windows SharePoint Services virtual servers. Windows SharePoint Services itself has the same functionality, regardless of whether you are running on ASP.NET 2.0 or ASP.NET 1.x.

Installing Windows SharePoint Services and ASP.NET 2.0

In this section, we show you how to install Windows SharePoint Services Service Pack 2 and ASP.NET 2.0.

As a starting point, every SharePoint server will have ASP.NET 1.1 installed on it. However, it is possible to run ASP.NET 1.1 and ASP.NET 2.0 side by side on the same server. There is an important limitation: if you install ASP.NET 2.0 on a server, you cannot install SharePoint Portal Server 2003 on that machine anymore. As a consequence, upgrading to ASP.NET 2.0 is

not an option for SharePoint Portal Server 2003, although it is an appealing option for Windows SharePoint Services. For Windows SharePoint Services installations, if you upgrade to ASP.NET 2.0 (and the .NET Framework 2.0, which is automatically upgraded when installing ASP.NET 2.0) you will be able to take advantage of the many improvements to the development framework and environment. Your code will also be able to take advantage of new security and performance enhancements.

In our experience the installation process works best if you start with a clean install of Windows Server 2003 R2. The R2 release is required; previous versions of Windows Server 2003 will not work. After installing Windows Server 2003 R2, the first thing to do is to install Internet Information Services (IIS). This can be done by going to Add or Remove Programs via the Control Panel and then choosing Add/Remove Windows Components. Make sure not to install FrontPage Server Extensions. Do not install ASP.NET at this time; otherwise the installation process will fail.

After installing IIS, you are ready to install ASP.NET 2.0. Go back to Add or Remove Programs and click Add/Remove Windows Components. Select Microsoft .NET Framework 2.0 and click Next to install.

The next step is to configure IIS for ASP.NET 2.0. Go to the Internet Information Services Manager by typing **inetmgr** at the command prompt and right-click the virtual server you want to configure, and then click Properties. In the Default Web Site Properties window go to the ASP.NET tab and choose the right ASP.NET version (version 2.0) from the drop-down list and click OK. Figure 1-1 shows the ASP.NET tab in the Default Web Site Properties window. Finally, restart IIS by typing **iisreset** at the command prompt.

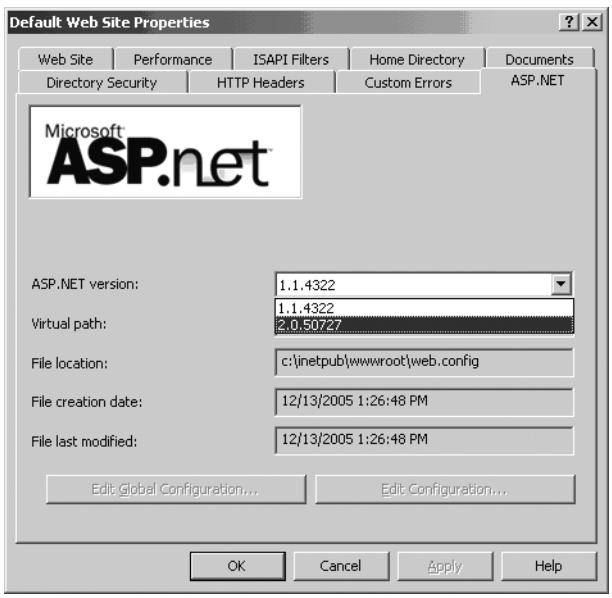

Figure 1-1. *The ASP.NET tab of the Default Web Site Properties window*

Data Stores

There are three possible data stores to choose from when installing Windows SharePoint Services: Windows Microsoft SQL Server 2000 Desktop Engine (WMSDE), which has limitations but is free; SQL Server 2000; or SQL Server 2005. WMSDE is a special version of the Microsoft SQL Server Desktop Engine (MSDE) and is designed to be used by Windows components. It is not limited in the same way the MSDE is. In it, the maximum size limit and current connections limit have been removed. Nevertheless, WMSDE is still more limited than SQL Server 2000 and it is certainly more limited than SQL Server 2005. WMSDE does not support full-text search, can be managed locally but not remotely, cannot be used in a web farm scenario, and does not include the SQL Enterprise Manager tool.

Installing SharePoint Services When Using WMSDE

If you want to use WMSDE as the data store for Windows SharePoint Services, follow these steps:

1. Go to Start ➤ Administrative Tools ➤ Manage Your Server and click Add or Remove a Role.
2. Click Next.
3. Select the SharePoint Services role, as shown in Figure 1-2, and follow the steps of the wizard.

This installation will also install Service Pack 2.0 of Windows SharePoint Services.

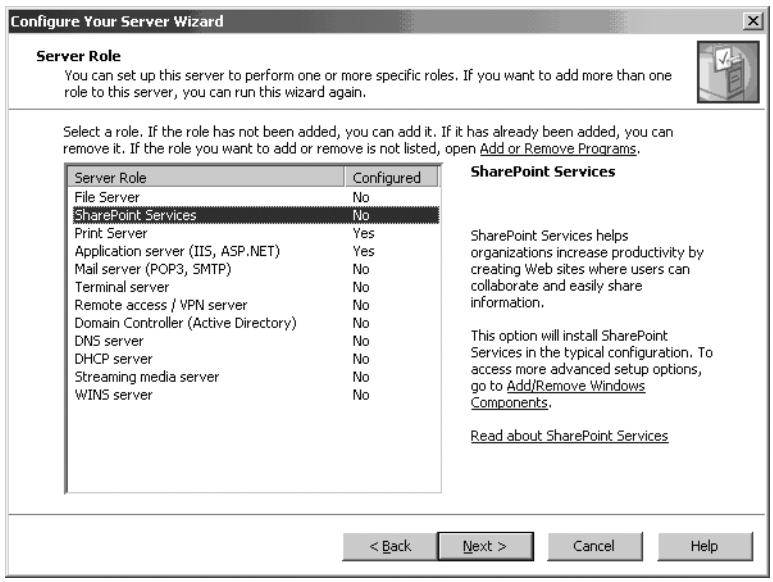

Figure 1-2. *Choose SharePoint Services in the Configure Your Server Wizard.*

Installing SharePoint Services When Using SQL Server 2000 or SQL Server 2005

When using SQL Server 2000 or 2005 as the data store, you have to install Windows SharePoint Services in a different way. Via Add/Remove Windows Components, select Windows SharePoint Services as shown in Figure 1-3. This installation already contains Service Pack 2.

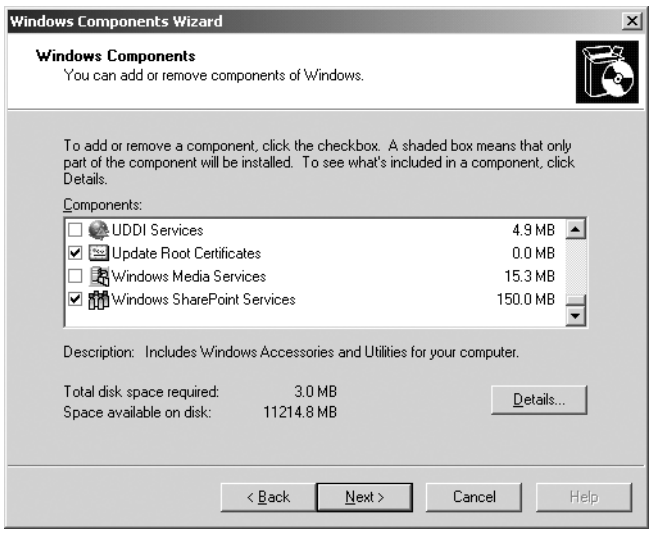

Figure 1-3. *Select SharePoint Services in the Windows Components Wizard.*

How to Configure ASP.NET 2.0

ASP.NET code runs at the trust level that is assigned to it. This trust level is determined by the code access security policy file specified in the web.config file. The Wss_minimaltrust.config file is an example of a code access security policy file, and it is the default policy file used in SharePoint Products and Technologies.

ASP.NET 2.0 implements a new security change that helps to lock down security for a virtual server in Internet Information Services 6.0. As a result of this security change, the permissions of web pages, web parts, and controls are limited to the intersection of the default ASP.NET permission set and the permission set currently granted by the trust level under which the code runs. The current trust level is defined in the web.config file of a SharePoint virtual server. Because this lockdown is incompatible with the security permission set required by Windows SharePoint Services, it must be disabled, which can be done via the web.config file of a SharePoint virtual server.

Another configuration requirement for ASP.NET 2.0 and Windows SharePoint Services is related to a new ASP.NET feature called *event validation*, which is responsible for monitoring callbacks to the ASP.NET infrastructure and making sure callback sources equal control targets. ASP.NET 2.0 event validation is not compatible with Windows SharePoint Services because some SharePoint pages use callbacks that are not associated to any particular control. This results in page execution errors, unless ASP.NET 2.0 event validation is turned off for any SharePoint-extended web application on a server.

The SharePoint stsadm.exe command line tool can be used to update the settings of the web.config file of a SharePoint virtual server so that ASP.NET 2.0 is compatible with Windows SharePoint Services. Use the following command:

```
stsadm -o upgrade -forceupgrade -url http://[MyVirtualServer]
```

When you use this command (which we will call the `stsadm` upgrade command from now on), you will see the pop-up window shown in Figure 1-4.

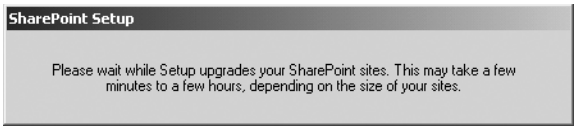

Figure 1-4. *Pop-up window when the* `stsadm` *command is running*

This command will update the web.config file that is located in the root folder of the virtual server. By default, this folder is [drive letter]:\inetpub\wwwroot. The upgrade operation will add the `processRequestInApplicationTrust` attribute to the `<trust>` element. Setting this attribute to `false` determines that the permission grant set is restricted to the permissions configured in the trust policy file. The following line of code shows an example of the trust level defined in a web.config file. In this case, the trust level is set to `Full`:

```
<trust level="Full" originUrl="" processRequestInApplicationTrust="false" />
```

The `stsadm` upgrade command will also add a namespaces section under the `<pages>` element. This section is new in .NET Framework 2.0. The `<namespaces>` element defines a collection of import directives to use during assembly precompilation. These are configured in the root web.config file. You can remove any of these namespaces from the collection by using the `<remove>` element in the web.config file for an application. You must remove the `System.Web.UI.WebControls.WebParts` namespace because it is the ASP.NET web part namespace and it conflicts with the SharePoint web part class, which has the following namespace: `Microsoft.SharePoint.WebPartPages.WebPart`. The following code listing demonstrates how to remove the `Microsoft.SharePoint.WebPartPages.WebPart` namespace:

```
<namespaces>
  <remove namespace="System.Web.UI.WebControls.WebParts" />
</namespaces>
```

Finally, the `stsadm` upgrade command will add an `enableEventValidation` attribute to the `<pages>` element, which turns event validation off. The following code listing demonstrates how to disable event validation:

```
<pages enableSessionState="false" enableViewState="true" enableViewStateMac="true" ➥
validateRequest="false" enableEventValidation="false">
```

When you have followed these steps, you can make your own SharePoint site using ASP.NET 2.0. Figure 1-5 shows you the first screen of a SharePoint site.

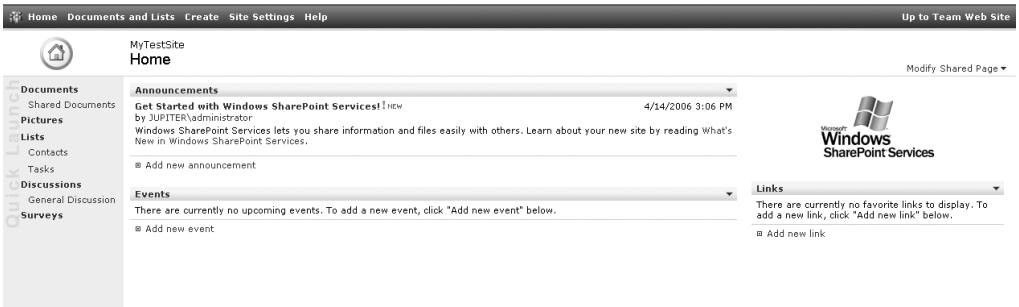

Figure 1-5. *The first screen of a default SharePoint site*

If you fail to run the `stsadm` and you browse to a Windows SharePoint Services site on a server with ASP.NET 2.0 installed on it, you will see the error that is shown in Figure 1-6.

Figure 1-6. *Error when running SharePoint site in ASP.NET 2.0 without running the `stsadm` command*

If you want to go back to using ASP.NET 1.x instead of ASP.NET 2.0, you have to run the exact same `stsadm` command again to remove the elements and attributes from the web.config file. If you forget to do this and run your SharePoint site in ASP.NET 1.x, you will get an error, as shown in Figure 1-7.

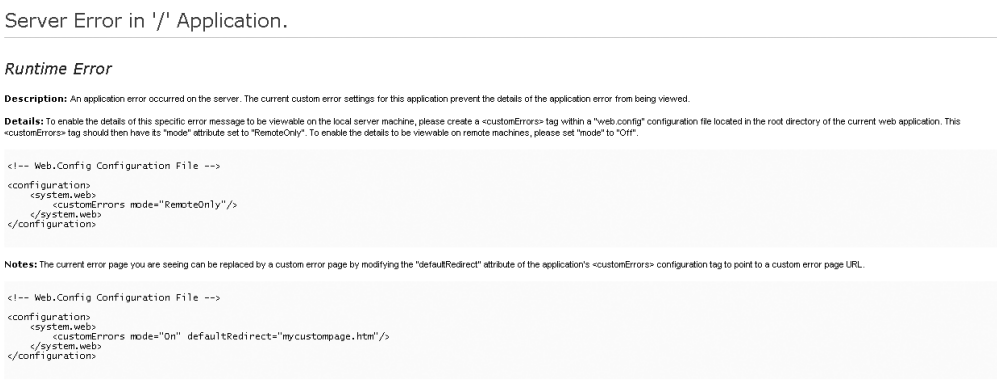

Figure 1-7. *Error when running SharePoint site in ASP.NET 1.x without running the* `stsadm` *command*

Things to Remember

When your virtual server runs in ASP.NET 1.1 mode, you cannot call another assembly that is compiled in .NET 2.0. This is known as *upstreaming*. Downstreaming is possible; if your virtual server runs in ASP.NET 2.0, you can call another assembly that is compiled in .NET 1.x.

ASP.NET 2.0 web parts can be added to pages running in Windows SharePoint Services after Service Pack 2.0 is applied. They will not be treated as web parts; instead, they will be seen as standard web form controls. Out of the box, Windows SharePoint Services will not use any ASP.NET 2.0 constructs. It is not necessary to specify assembly redirection or runtime information for the ASP.NET 2.0 web part assemblies unless you want to deploy them. If you want to deploy web part packages that contain Common Language Runtime version 2.0–compiled web part assemblies, you will need to create a stsadm.exe.config file for stsadm.exe that specifies the following:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v2.0.50727" />
    <supportedRuntime version="v1.1.4322" />
  </startup>
</configuration>
```

This config file has to be placed next to stsadm.exe. The default location is [drive letter]:\Program Files\Common Files\Microsoft Shared\web server extensions\60\BIN. If you do not create this config file, you will get the following error message: “Version 1.1 is not a compatible version.”

Creating Web Parts via Visual Studio .NET 2005

After installing ASP.NET 2.0 and Windows SharePoint Services, you can start creating web parts. Start by opening Visual Studio .NET 2005 and create a project that is based on the class library project template.

First of all, you have to make sure that the dll that will be created when you compile your project will be placed in a bin folder under the root folder of your virtual server. If you do not have a bin folder under the root folder, you can create one yourself. By default, the location of the root folder will be [drive letter]\inetpub\wwwroot. Set the output path to the bin folder of the root folder of the SharePoint virtual server by right-clicking your project, choosing Properties, and clicking the Build tab. In the Output section, you will find the output path property, as shown in Figure 1-8.

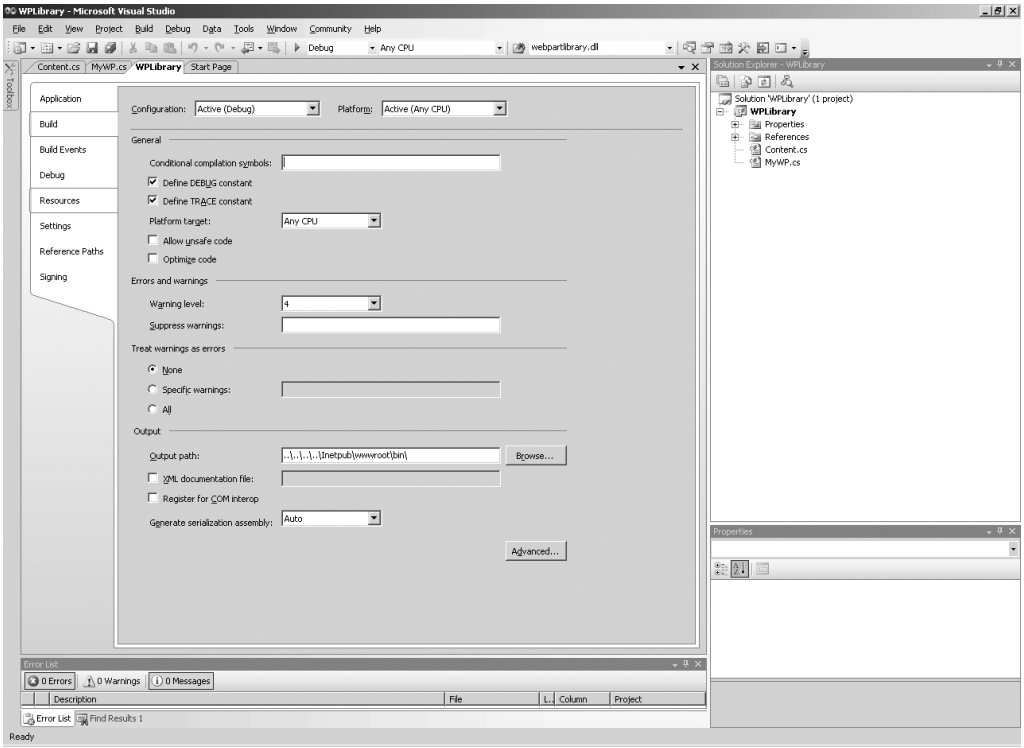

Figure 1-8. *Setting the output path property in Visual Studio .NET 2005*

Before starting to code the web part, you also have to add references in your project to the System.Web and the Microsoft.SharePoint dlls. By default, you can find the Microsoft.SharePoint.dll on the following location: [drive letter]:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\60\ISAPI\.

The following code example shows a list of all the imported namespaces. It also shows that the web part inherits from the `Microsoft.SharePoint.WebPartPages.WebPart` class, the base class for all SharePoint web parts. To prove we are indeed using .NET Framework 2.0, we have overridden the `CreateChildControls()` method and used generics (a new feature of .NET Framework 2.0) to print a Hello World message to the page. The following code listing contains the entire code for a web part that uses .NET 2.0 generics to print a Hello World message to a SharePoint page:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace LoisAndClark.WPLibrary
{
  public class MyWP : WebPart
  {
    protected override void CreateChildControls()
    {
      Content obj = new Content();
      string str1 = obj.MyContent<string>("Hello World!");
      this.Controls.Add(new System.Web.UI.LiteralControl(str1));
    }
  }
}
```

The generic method shows that the SharePoint site is really running on .NET Framework 2.0. The code of the generic method looks like this:

```
public string MyContent<MyType>(MyType arg)
{
  return arg.ToString();
}
```

The next thing to do is to add the assembly to the SafeControls list in the web.config file. Doing this allows the web part to run within a SharePoint page.

```
<SafeControl Assembly="WPLibrary" Namespace="LoisandClark.WPLibrary" TypeName="*" ➥
Safe="True" />
```

There are two ways to import the web part in a SharePoint site: either you create a Description Web Part File (.dwp) file, or you browse to the web part gallery of your SharePoint site and upload the web part as a new web part. If the web part is not signed with a public key, the content of a .dwp file should look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<WebPart xmlns="http://schemas.microsoft.com/WebPart/v2" >
  <Title>Title of Web Part</Title>
```

```
  <Description>Description of Web Part</Description>
  <Assembly>Assembly</Assembly>
  <TypeName>Namespace.Classname</TypeName>
</WebPart>
```

After creating a .dwp file you can browse to a SharePoint site, click Modify Shared Page ➤ Add Web Parts ➤ Import. Browse to the .dwp file that is located on your computer and click Import.

The other way to import a web part in a SharePoint site is via the Web Part Gallery. The Web Part Gallery is only available on the top-level site administration page, which means that if you add a web part there, it will be available on all the subsites. The way to get there is to click Site Settings ➤ Go to Site Administration ➤ Manage Web Part Gallery. You can also go to the Web Part Gallery page by entering the following URL: `http://[MyServerName]/_catalogs/wp/Forms/AllItems.aspx`. Then click New Web Part. This page will display all available web parts for this gallery, as shown in Figure 1-9.

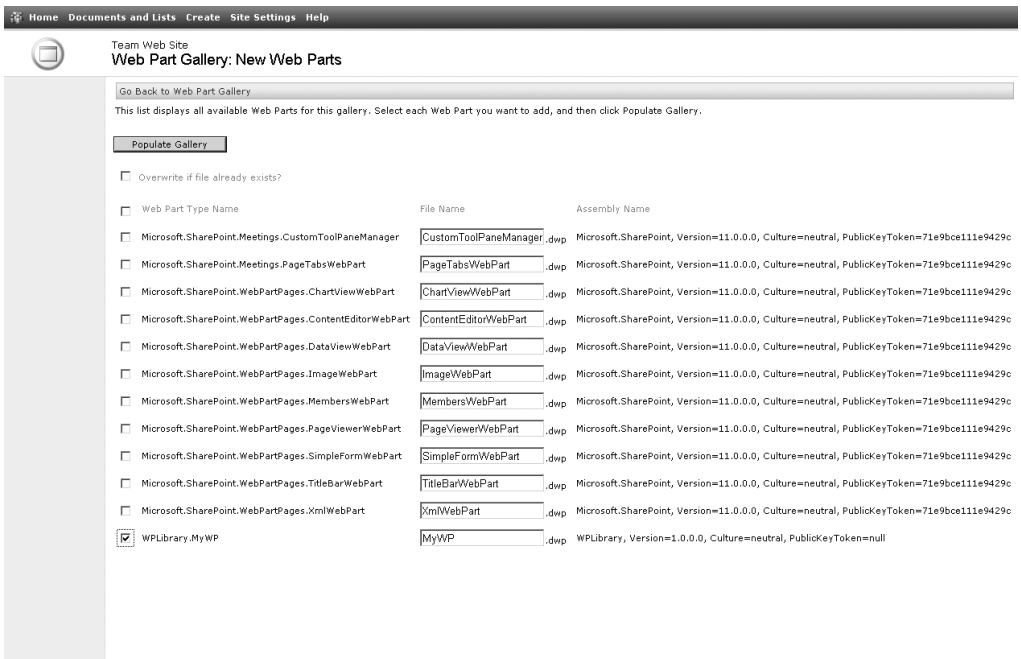

Figure 1-9. *The New Web Parts page of the Web Part Gallery*

Click the check box next to your web part and then click Populate Gallery. Notice that the MyWP web part (previously selected in Figure 1-9) is listed in the Web Part Gallery in the form of a reference to its web part description file (MyWP.dwp). This is shown in Figure 1-10. If the web part is not shown, open a command prompt and type **iisreset**. Then check again.

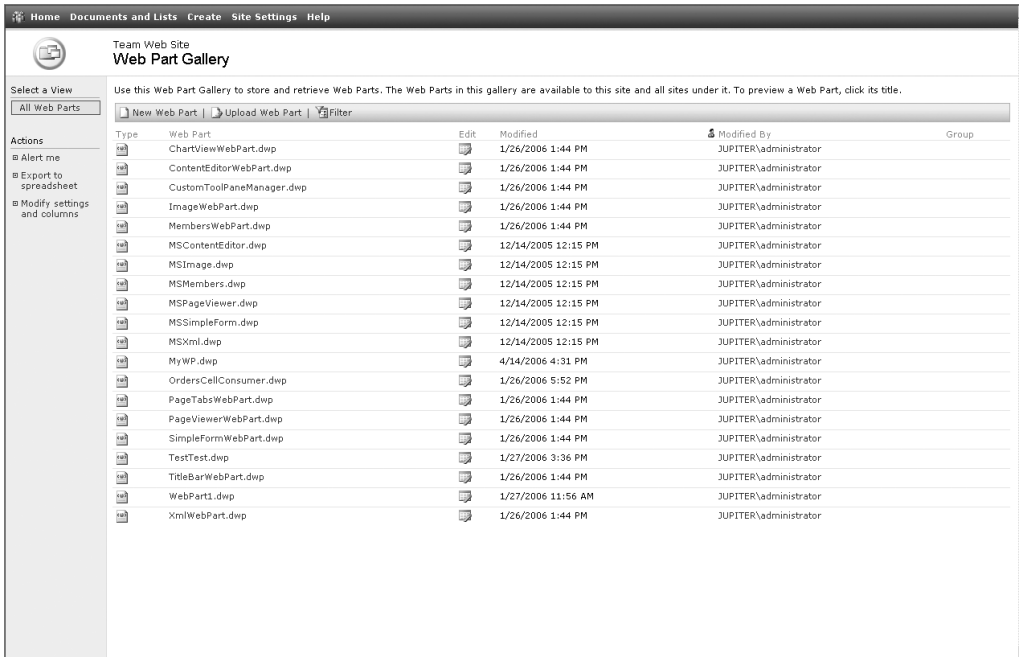

Figure 1-10. *The Web Part Gallery*

Go back to your SharePoint site and click Modify Shared Page ➤ Add Web Parts ➤ Browse. You will find your web part in the Team Web Site Gallery. Now you can add your web part by dragging and dropping it onto the web part page. After you add your web part, you will have a SharePoint site running on .NET Framework 2.0 containing a web part created with Visual Studio .NET 2005.

Enhancing Development of Web Parts with the Guidance Automation Toolkit

Creating a web part with Visual Studio .NET 2005 is more work than it used to be in the days of Visual Studio .NET 2003. If you create a web part with Visual Studio .NET 2003, you can use the web part template that can be downloaded from the Microsoft web site. This template is able to do the basic things needed to build web parts for you, thus saving you the effort of doing a boring bit of work. This is not the case with Visual Studio .NET 2005.

You can facilitate the development of web parts with the Guidance Automation Toolkit (GAT). The GAT is an extension to Visual Studio .NET 2005 that allows architects to automate the easy parts of development so that the developer can concentrate on the other parts. The GAT can be used to create assets that are developed in-house or by third parties, such as Microsoft. In the case of building a web part, the GAT can be used to build a package you can use as a template to start making web parts in Visual Studio .NET 2005. These packages are also known as *guidance packages*. We will show you how to use the GAT to create such a package.

Guidance Package Development

To help you with guidance package development, the GAT includes a Guidance Package Development Template. This template can be used to create a solution for guidance package development that includes the elements you need to create your own guidance package. The GAT is shipped with documentation that contains extensive information about the guidance package development process.

The web part library template can be downloaded from our web site: `http://www.lcbridge.nl/download`. The download is in the form of a Windows Installer file (.msi), which can be used to install a web part library template that can be used within Visual Studio .NET 2005.

In the previous section, we showed that you do not need a .dwp file to import a web part on a SharePoint site. However, if you want to strong-name your assembly you do need to create a .dwp file. Strong-named assemblies are explained later in this chapter in the "Installing and Using the Web Part Library Template" section.

We have created an item template that makes it possible to add a .dwp file to your web part library solution. If you want to use the web part library template, your computer needs to have the following software installed:

- Microsoft Windows SharePoint Services and Service Pack 2.0.
- Microsoft Visual Studio .NET 2005.
- Guidance Automation Extensions (GAX). This runtime component is required when running guidance packages. The Guidance Automation Extensions can be downloaded at `http://msdn.microsoft.com/vstudio/teamsystem/Workshop/gat/download.aspx`.
- Guidance Automation Toolkit. The GAT documentation says that you do not need GAT to use guidance packages, although in our experience this is not true. The current beta versions require it, even if you are not creating guidance packages but only using them.

Installing and Using the Web Part Library Template

In this section we show how to install and use the web part library template. Later on, we will discuss how to use GAT to create the web part library template itself and how to create a Windows Installer package for guidance packages. If you are only interested in using the web part library template to enhance the Visual Studio .NET 2005 web part development experience, this section contains all the information you need.

Download the WebPartLibrarySetup.msi file from our web site (`http://www.lcbridge.nl/download`). You can install the web part library template by double-clicking the Windows Installer Package. Close all instances of Visual Studio .NET 2005 before installing the package. The Windows Installer package will install all assemblies related to the web part library template in a folder dedicated to the guidance package. The Guidance Automation Extensions do not support assemblies in the Global Assembly Cache (GAC) and will not load assemblies located in there, even if the assembly is explicitly referenced in the guidance package. Double-click the .msi file to install the package. This will open a pop-up window with welcome text, as shown in Figure 1-11.

■ **Note** To create the Windows Installer package that is used to install the web part library template, we used the Technology Preview version of the Guidance Automation Toolkit because the official GAT was not released at the time of this writing.

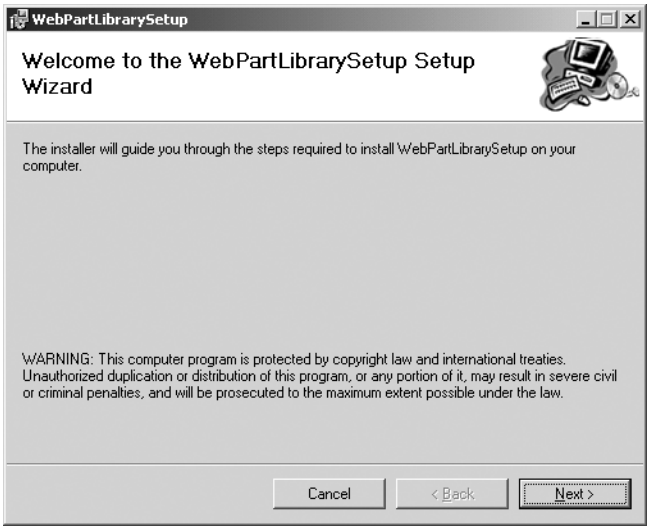

Figure 1-11. *First step of the setup of the web part library template*

Next, choose the folder where you want the template to be installed and click Next. See Figure 1-12.

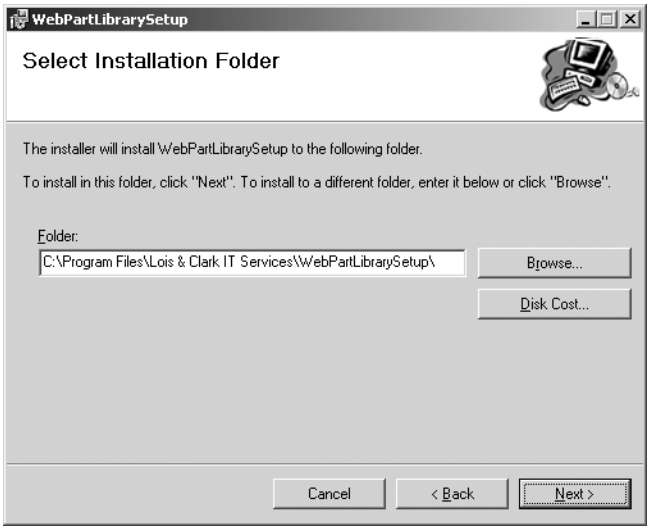

Figure 1-12. *Second step of the setup of the web part library template*

Click Next to start the installation, as shown in Figure 1-13.

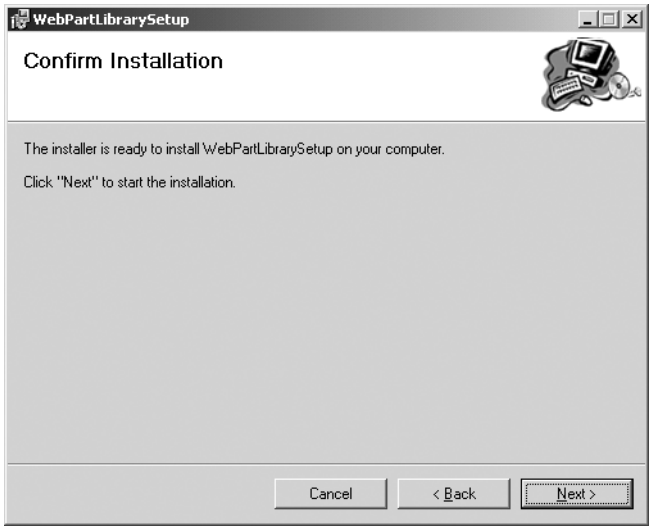

Figure 1-13. *Last step of the setup of the web part library template*

When the installation has succeeded, you will find the web part library template under Add and Remove Programs, as seen in Figure 1-14. This is the place where you can remove or repair the template. You cannot install two versions of the same guidance package at the same time. If you do attempt to install a guidance package with the same name as an existing guidance package, the Guidance Automation Extensions will throw an exception, informing you that you must uninstall the previous instance of the guidance package before installing the new one.

You will also find evidence of installing the web part library template in Visual Studio .NET 2005. Open Visual Studio .NET 2005 and choose File ➤ New ➤ Project. In the New Project window, you will see a new project type called Guidance Packages. Click Guidance Packages and underneath it you will find a package called WebPartLibrary. In the right window pane, you will see a template called WebPartLibrary Solution, as shown in Figure 1-15.

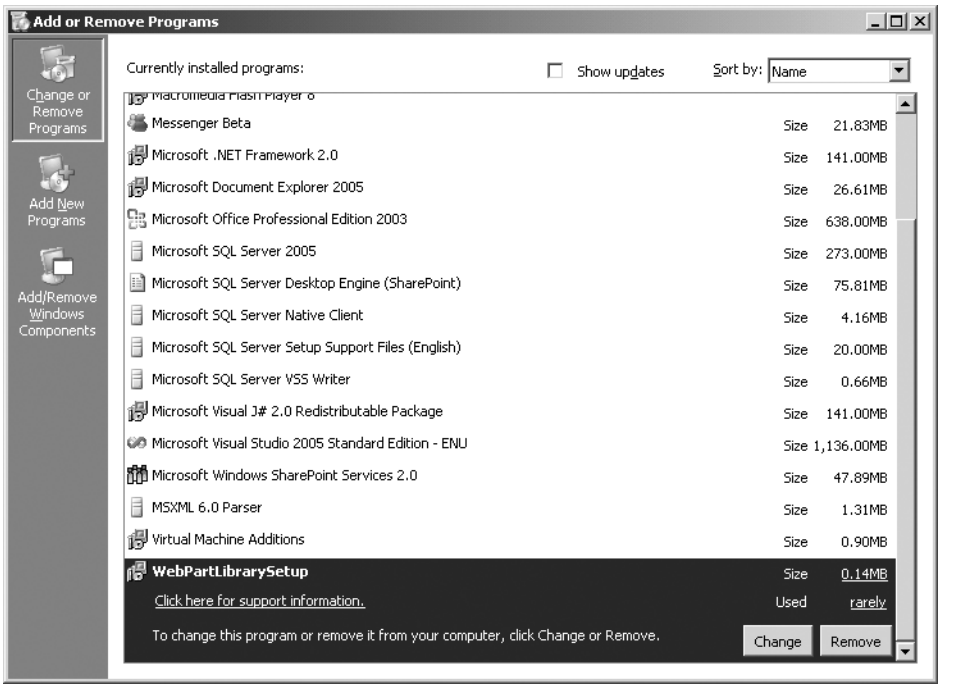

Figure 1-14. *The Add and Remove Programs window*

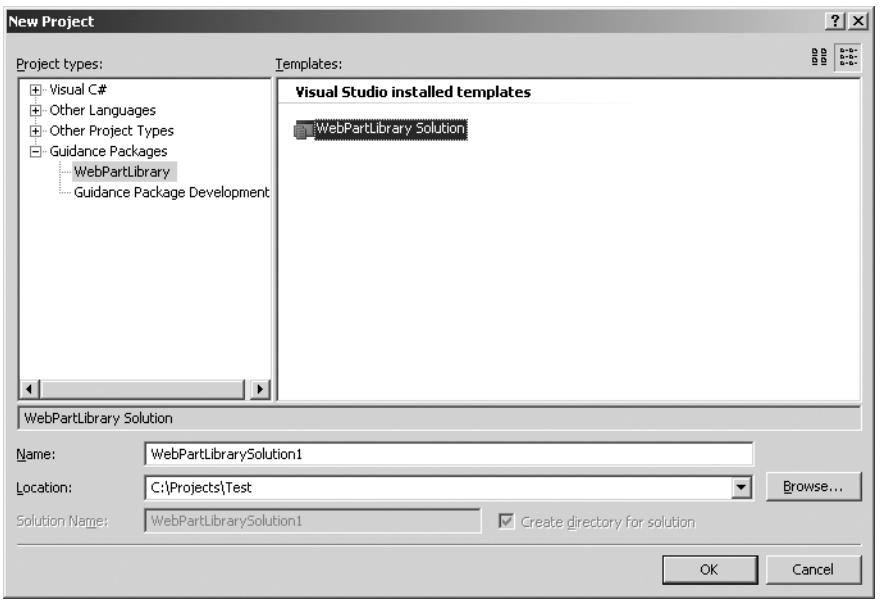

Figure 1-15. *Choose a project in Visual Studio .NET 2005.*

Click WebPartLibrary Solution, fill in a location and a descriptive name, and click OK. This will start a wizard page where you will fill in a project name and the name of the class that will be created initially. Then click Finish, as shown in Figure 1-16.

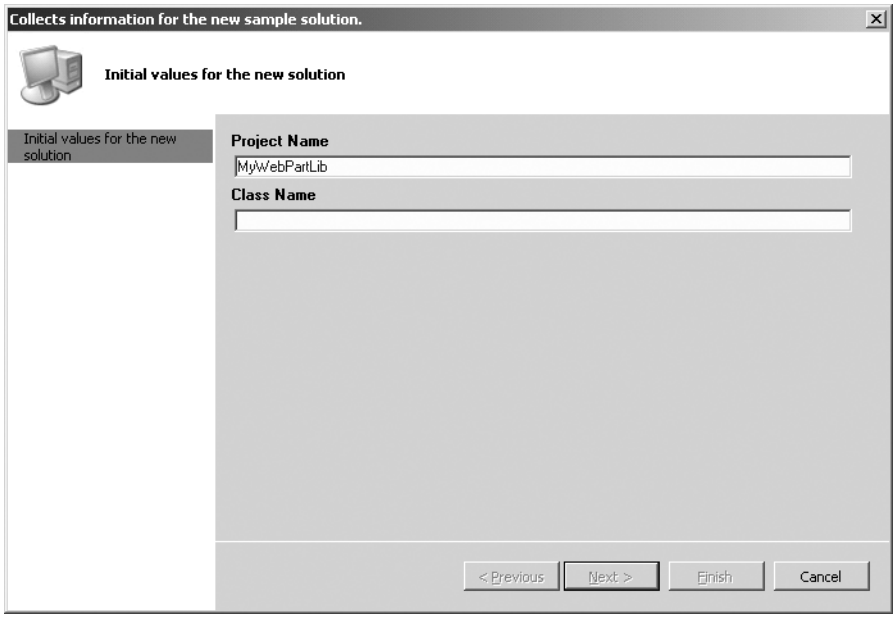

Figure 1-16. *Wizard page belonging to the project creation*

A couple of things will be created at this point:

- A solution containing a properties directory with an assembly.info file
- A references directory containing the following references:
  - System
  - System.Data
  - System.Web
  - System.Xml
  - Microsoft.SharePoint
- A class file called WebPart.cs

Figure 1-17 shows the new solution created with the help of the web part library template.

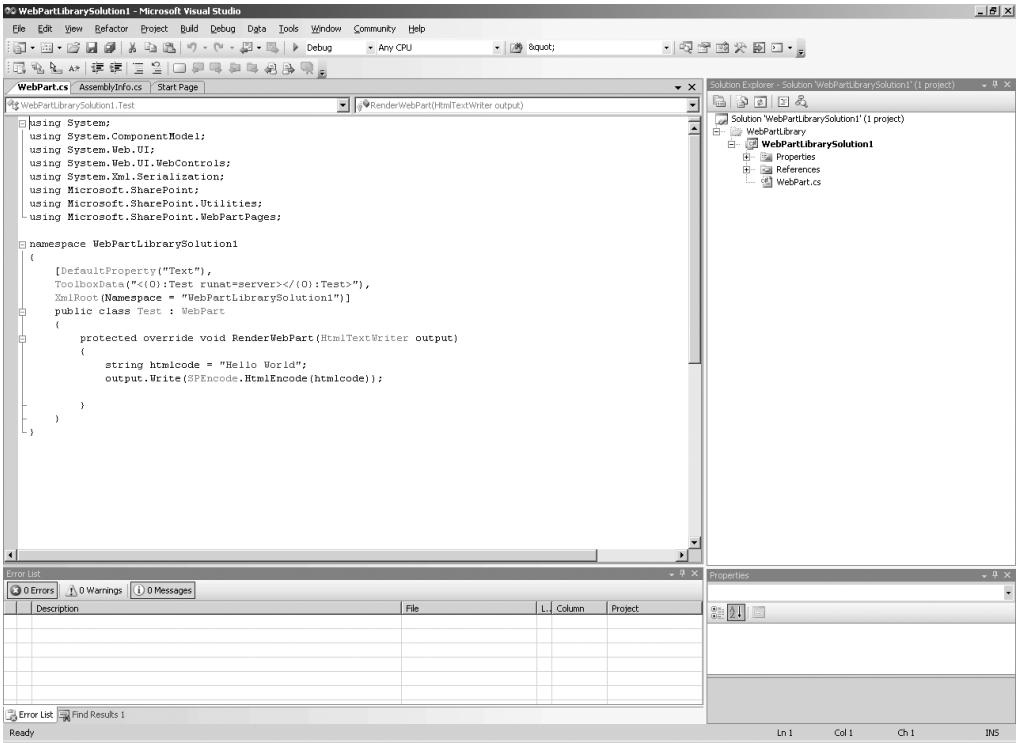

Figure 1-17. *Web part library template in Visual Studio*

The WebPart.cs class file contains the following code:

```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;

namespace WebPartLibrarySolution1
{
  [DefaultProperty("Text"),
  ToolboxData("<{0}:Test runat=server></{0}:Test>"),
  XmlRoot(Namespace = "WebPartLibrarySolution1")]
  public class Test : WebPart
  {
    protected override void RenderWebPart(HtmlTextWriter output)
    {
```

```
      string htmlcode = "Hello World";
      output.Write(SPEncode.HtmlEncode(htmlcode));
    }
  }
}
```

The class itself inherits from the `Microsoft.SharePoint.WebPartPages.WebPart` class, which makes it a web part that is suitable to be used within a SharePoint site.

There are two things left to do. First, check if the output path of the project is changed to the bin directory of the root folder of the virtual server. If this is not the case, change the output path. This makes testing the web part considerably easier. Otherwise you would have to copy the web part dll manually each time you compile. By default, the root folder is [drive letter]\inetpub\wwwroot. If the bin folder does not exist, you have to create one yourself. Secondly, as shown in the following code listing, you need to add your assembly to the SafeControls list of the web.config file, which is also located at the root of the virtual server:

```
<SafeControl Assembly="MyWebPartLib" Namespace="MyWebPartLib" TypeName="*" ➥
Safe="True" />
```

Here, we are using a partially qualified assembly name, which is great for creating code examples. We could have chosen to use fully qualified names, a practice that is recommended for production code. Those names include the following information: assembly name, version number, culture (which is always set to neutral for code assemblies), and the developer identity (a public key token).

Assemblies with fully qualified names are also known as *strong-named assemblies*. Strong-named assemblies make it easier to enforce security policies for assemblies, because you can assign security permissions based on developer identity. Another advantage is that strong names make creating unique assembly names easier, thus reducing chances for name conflicts. The content of a strong-named assembly cannot be tampered with after compilation, as strong-named assemblies contain a hash code representing the binary content of the assembly. This hash code is unique for every assembly, and the .NET CLR makes sure the hash code matches the assembly content during load time. If someone has tampered with your assembly after compilation, the hash code will not match the content and will not be loaded. A final advantage is that version policies are only applied to strong-named assemblies, not to assemblies with partially qualified names. Strong names are mandatory for assemblies that need to be installed in the GAC. The next code shows a SafeControl entry for a strong-named assembly:

```
<SafeControl Assembly="MyWebPartLib" Namespace="MyWebPartLib, ➥
Version=1.0.0.0, Culture=neutral, ➥
PublicKeyToken=71e9bce111e9429d" TypeName="*" ➥
Safe="True" />
```

If you want to register your web part on a SharePoint site via a web part description file, you can create one by right-clicking your project and choosing Add New Item. Under Categories, you will find a new category called WebPartLibrary. Click this category and choose the template called Web Part Dwp. Give the .dwp file a descriptive name and click Add. Figure 1-18 shows the Add New Item window.

You can also create a .dwp file by right-clicking your project and then choosing Web Part Dwp, as shown in Figure 1-19. This will open the Add New Item window as well.

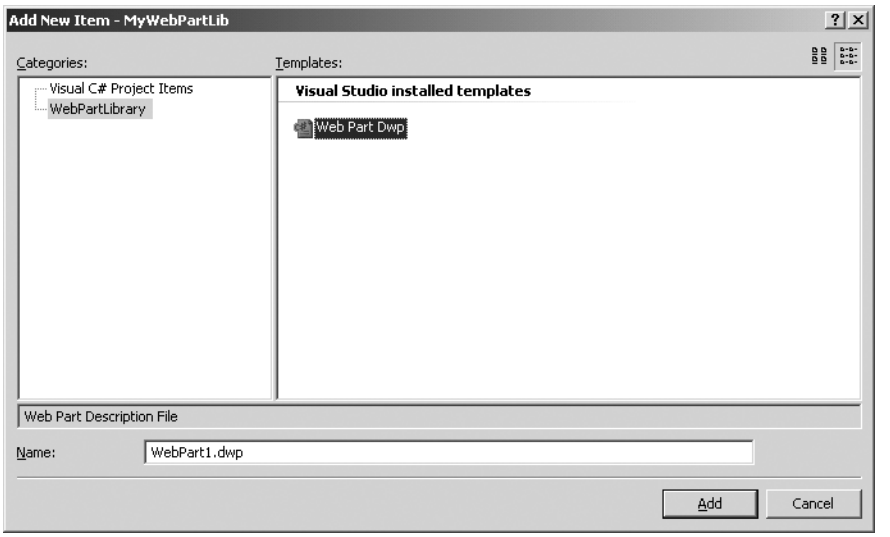

Figure 1-18. *The Add New Item window in Visual Studio .NET 2005*

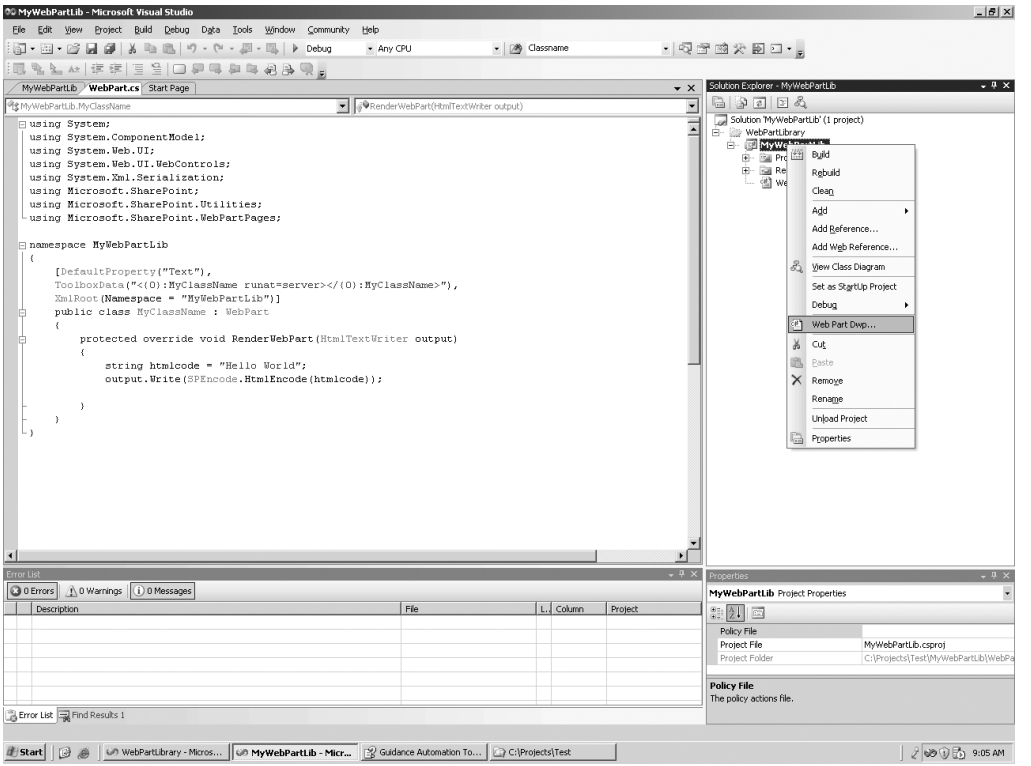

Figure 1-19. *Add a new web part .dwp file by right-clicking your project.*

This will start a wizard that collects information about the web part description file. The assembly name and namespace will have default values. You will only have to type the name of the class, a title for your web part, and a description. The title and description are shown on the SharePoint site page the web part is imported into. The wizard is shown in Figure 1-20.

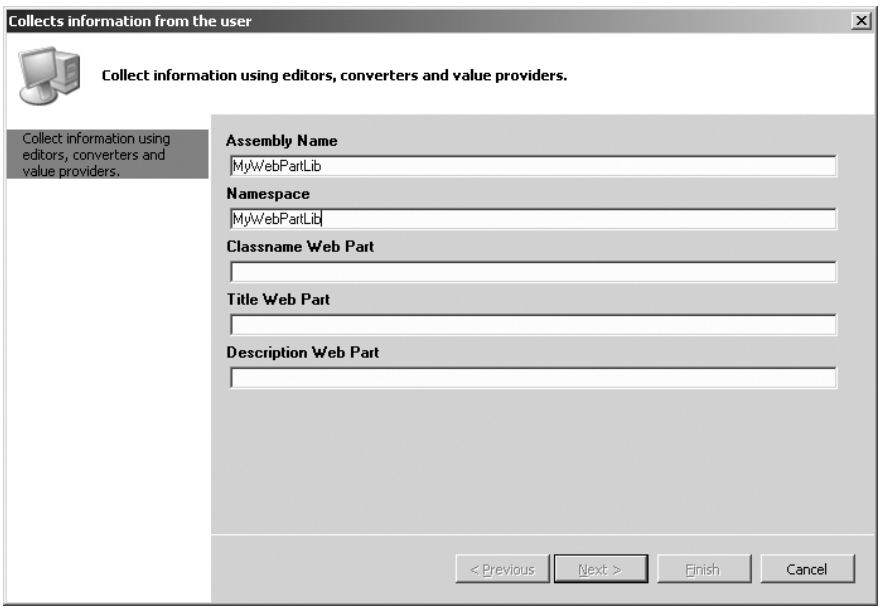

Figure 1-20. *The web part .dwp file wizard*

At this point, you have created your first web part via the web part library template. Now you can start adding your own code to the web part.

Guidance Automation Toolkit

In the previous section, you saw how to install the web part library template. If you want to create your own guidance packages, you will need to install the GAT. In this section, we discuss the general concepts you need to be aware of before you can start using the GAT. First you have to install the Guidance Automation Extensions, which can be downloaded at the Microsoft MSDN web site: `http://msdn.microsoft.com/vstudio/teamsystem/workshop/gat/`. Next, you should install the GAT itself. In this book we used the June 2006 Technology Preview version.

The GAT contains a couple of elements that work together to provide the automation functionality. The elements are recipes, actions, text template transformation templates, wizards, and Visual Studio templates, described in the following list:

- *Recipes*: Recipes automate a series of activities that are performed by a developer. Recipes are very suitable to automate repetitive actions.
- *Actions*: Actions are the most fine-grained parts of a recipe. Each action is an atomic unit of work. The order in which actions are executed is determined by the recipe definition. Actions are able to accept input via arguments gathered by a recipe or output received from another action that has already been run by the recipe.

- *Text template transformation templates (T4)*: T4 templates return a string that is inserted in the template output and contains a combination of text and *scriptlets*. Scriptlets are expressions written in VB.NET or C# that are interpreted when the text template transformation engine that is a part of the GAT runs a template. The outcome of a scriptlet is dynamic and is added to the template output. The text part of a template is always inserted unmodified in the template output. You can also use actions to generate text from a T4 template. The text template transformation engine is not just a part of the GAT; it is also included in the Domain Specific Language (DSL) Toolkit. We discuss the DSL Toolkit in detail in Chapter 5.

- *Wizards*: Recipes can be associated to wizards. A wizard contains one or more steps that are displayed as pages to guide the developer and to gather information used by a recipe. A wizard page contains one or more fields, where each field is associated to a recipe argument. Every field includes a type converter and a user interface (UI) type editor. The type converter validates the value of a field and converts the value from a user interface representation to a type representation. The UI type editor is used to render the user interface that collects the information.

  A wizard typically provides the developer with a number of default values that are either set when the recipe reference is created or are obtained by the recipe framework. Regardless of the presence of default values, a developer always has to step through the entire wizard that shows the defaults to the developer. This gives the developer a chance to acknowledge the default values.

  There are cases when it is not the responsibility of a developer to define or acknowledge a field value. Some fields are obtained by the framework or the wizard itself. This process is known as *value propagation*. Such fields are not shown in the wizard. If a field is required, the wizard framework will not allow access to later pages in the wizard via the wizard sidebar until all required fields contain data.

- *Visual Studio .NET templates*: In essence, a Visual Studio .NET template is nothing more than XML data that describes how to create Visual Studio solutions, projects, or items. Templates are unfolded by the Visual Studio template engine. Using the GAT, you can associate Visual Studio templates with recipes. This association means that when a template is unfolded, the wizard extension calls the recipe to let it collect parameter values, also known as *arguments*. The arguments are used to execute actions that may further transform solution items created by the template. Guidance packages can be managed via the Guidance Package Manager in Visual Studio .NET 2005. You can find the Guidance Package Manager via the Tools menu. Once a guidance package is installed and enabled for a particular solution, recipes can be executed to carry out the required tasks.

Creation of the Web Part Library Template

In this section, we explain how to use the GAT to create the web part library template that is discussed previously in the section "Installing and Using the Web Part Library Template." The GAT contains a predefined solution for developing a guidance package, which we will use to create a web part library template.

The first step for building a web part library template is to create a new folder in the Templates ➤ Solutions ➤ Projects folder. We will call this folder WebPartLibrary. This folder will contain the following files: WebPart.cs, WebPartLibrary.csproj, and a WebPartLibrary.vstemplate file. It will also contain a folder named Properties, and that folder will contain the assembly.info file.

The WebPartLibrary.vstemplate file is a Visual Studio template that looks identical to a normal Visual Studio template except that it contains additional information used by the recipe framework. The template includes a `<WizardExtension>` element that specifies a class in the recipe framework that implements template extensions for the GAT. The following XML shows the content of our WebPartLibrary Visual Studio template:

```
<VSTemplate Version="2.0.0" Type="Project" ➥
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>WebPart Library Project</Name>
    <Description>WebPart class library project </Description>
    <Icon Package="{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}" ID="4547" />
    <ProjectType>CSharp</ProjectType>
    <SortOrder>20</SortOrder>
    <CreateNewFolder>false</CreateNewFolder>
    <DefaultName>ClassLibrary</DefaultName>
    <ProvideDefaultName>true</ProvideDefaultName>
  </TemplateData>
  <TemplateContent>
    <Project File="WebPartLibrary.csproj" ReplaceParameters="true">
      <ProjectItem ReplaceParameters="true">Properties\AssemblyInfo.cs</ProjectItem>
      <ProjectItem ReplaceParameters="true">WebPart.cs</ProjectItem>
    </Project>
  </TemplateContent>
  <WizardExtension>
    <Assembly>
      Microsoft.Practices.RecipeFramework.VisualStudio, Version=1.0.51206.0, ➥
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
    </Assembly>
    <FullClassName>
      Microsoft.Practices.RecipeFramework.VisualStudio.Templates.UnfoldTemplate
    </FullClassName>
  </WizardExtension>
</VSTemplate>
```

This XML is called a *solution template*. Solution templates are launched via the `New` command on the File menu.

The next file of the package is the WebPartLibrary.csproj file. This is a project template that contains a project description. The template unfolds to a project in an existing solution. The project template will include references to other assemblies and will be responsible for creating any class files, the project file (.csproj), and the assembly.info file. The following XML shows the project template that can be used to create the web part library project:

```
<Project DefaultTargets="Build" ➥
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == '' ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>
    <ProductVersion>8.0.30703</ProductVersion>
    <SchemaVersion>2.0</SchemaVersion>
    <ProjectGuid>$guid1$</ProjectGuid>
    <OutputType>Library</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>$safeprojectname$</RootNamespace>
    <AssemblyName>$safeprojectname$</AssemblyName>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>\inetpub\wwwroot\bin\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>\inetpub\wwwroot\bin\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System"/>
    <Reference Include="System.Data"/>
    <Reference Include="System.Xml"/>
    <Reference Include="System.Web"/>
    <Reference Include="Microsoft.SharePoint"/>
  </ItemGroup>
  <ItemGroup>
    <Compile Include="WebPart.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <Import Project="$(MSBuildBinPath)\Microsoft.CSHARP.Targets" />
</Project>
```

The WebPart.cs file that is created by the project template looks like this:

```
using System;
using System.ComponentModel;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Xml.Serialization;
using Microsoft.SharePoint;
using Microsoft.SharePoint.Utilities;
using Microsoft.SharePoint.WebPartPages;

namespace $safeprojectname$
{
  [DefaultProperty("Text"),
  ToolboxData("<{0}:$ClassName$ runat=server></{0}:$ClassName$>"),
  XmlRoot(Namespace = "$safeprojectname$")]
    public class $ClassName$ : WebPart
    {
      protected override void RenderWebPart(HtmlTextWriter output)
      {
        string htmlcode = "Hello World";
        output.Write(SPEncode.HtmlEncode(htmlcode));
      }
    }
}
```

All arguments preceded by a $ will be replaced once the project is created. Some values are entered in the wizard by the developer, and some values will have dynamic values based on other values. For example, we have decided that the default namespace of a solution will be identical to the solution name. In the assembly.info file you will also find arguments that are preceded with $. You can look at the assembly.info code here:

```
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyTitle("$projectname$")]
[assembly: AssemblyDescription("")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("$registeredorganization$")]
[assembly: AssemblyProduct("$projectname$")]
[assembly: AssemblyCopyright("Copyright © $registeredorganization$ $year$")]
[assembly: AssemblyTrademark("")]
[assembly: AssemblyCulture("")]
[assembly: Guid("$guid1$")]

[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
```

The next thing to discuss is the item template. We have created a new item template in the Templates ➤ Items folder. Let us add an item template, which is responsible for making a .dwp file. You can see the item template XML here:

```
<VSTemplate Version="2.0.0" Type="Item" ➥
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">
  <TemplateData>
    <Name>Web Part Dwp</Name>
    <Description>Web Part Description File</Description>
    <Icon Package="{FAE04EC1-301F-11d3-BF4B-00C04F79EFBC}" ID="4515" />
    <ProjectType>CSharp</ProjectType>
    <SortOrder>10</SortOrder>
    <DefaultName>WebPart.dwp</DefaultName>
  </TemplateData>
  <TemplateContent>
    <ProjectItem ReplaceParameters="true">WebPart.dwp</ProjectItem>
  </TemplateContent>
  <WizardExtension>
    <Assembly>
      Microsoft.Practices.RecipeFramework.VisualStudio, Version=1.0.51206.0, ➥
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a
    </Assembly>
    <FullClassName>
      Microsoft.Practices.RecipeFramework.VisualStudio.Templates.UnfoldTemplate
    </FullClassName>
  </WizardExtension>
  <WizardData>
    <Template xmlns=http://schemas.microsoft.com/pag/gax-template ➥
    SchemaVersion="1.0" Recipe="NewItemClass"/>
  </WizardData>
</VSTemplate>
```

The item template will be used when the developer wants to add a .dwp file to the project. The item template uses a recipe called NewItemClass that is discussed later in this section. The code for the WebPart.dwp file looks like this:

```
<?xml version="1.0" encoding="utf-8"?>
<WebPart xmlns="http://schemas.microsoft.com/WebPart/v2">
  <Title>$TitleWebPart$</Title>
  <Description>$DescriptionWebPart$</Description>
  <Assembly>$AssemblyName$</Assembly>
  <TypeName>$Namespace$.$ClassName$</TypeName>
</WebPart>
```

All arguments preceded by a $ will be replaced once the developer creates a new .dwp file.

Now you have seen the solution template, the project template, the item template, and all corresponding files. The last file that we have not discussed yet is WebPartLibrary.xml. You can find this file directly underneath the project in the Solution Explorer. This file contains the XML configuration code, consisting of all the recipes, actions, and wizards. The first part of the

WebPartLibrary.xml file contains a recipe called BindingRecipe. The <Action> element contains the item template called WebPartTemplate.vstemplate. The <Arguments> element contains all arguments that will be asked for in the first wizard page when you create the solution. The next code listing shows an example WebPartLibrary.xml file:

```
<Recipe Name="BindingRecipe">
  <Types>
    <TypeAlias Name="RefCreator" ➥
    Type="Microsoft.Practices.RecipeFramework.Library.Actions. ➥
    CreateUnboundReferenceAction, Microsoft.Practices.RecipeFramework.Library"/>
  </Types>
  <Caption>Creates unbound references to the guidance package</Caption>
  <Actions>
    <Action Name="CreateSampleUnboundItemTemplateRef" Type="RefCreator" ➥
    AssetName="Items\WebPartTemplate.vstemplate" ➥
    ReferenceType="WebPartLibrary.References.ClassLibraryReference, ➥
    WebPartLibrary" />
  </Actions>
</Recipe>
<Recipe Name="CreateSolution">
  <Caption>Collects information for the new sample solution.</Caption>
  <Arguments>
    <Argument Name="ProjectName">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
    </Argument>
    <Argument Name="ClassName">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
    </Argument>
  </Arguments>
  <GatheringServiceData>
    <Wizard xmlns=" ➥
    http://schemas.microsoft.com/pag/gax-wizards" ➥
    SchemaVersion="1.0">
      <Pages>
        <Page>
          <Title>Initial values for the new solution</Title>
          <Fields>
            <Field Label="Project Name" ValueName="ProjectName" />
            <Field Label="Class Name" ValueName="ClassName" />
          </Fields>
        </Page>
      </Pages>
    </Wizard>
  </GatheringServiceData>
</Recipe>
```

The next recipe that is important for the web part library template is called NewItemClass. This recipe contains all arguments that are collected by the wizard when you create a .dwp file. Some arguments have default values, such as assembly name and namespace.

```
<Recipe Name="NewItemClass" Recurrent="true">
  <xi:include href="TypeAlias.xml" xmlns:xi="http://www.w3.org/2001/XInclude" />
  <Caption>Collects information from the user</Caption>
  <Description></Description>
  <HostData>
    <Icon ID="1429"/>
    <CommandBar Name="Project" />
  </HostData>
  <Arguments>
    <Argument Name="CurrentProject" Type="EnvDTE.Project, EnvDTE, ➥
    Version=8.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a">
    <ValueProvider Type="Microsoft.Practices.RecipeFramework. ➥
    Library.ValueProviders.FirstSelectedProject, ➥
    Microsoft.Practices.RecipeFramework.Library" />
    </Argument>
    <Argument Name="Namespace">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
      <ValueProvider Type="Evaluator" Expression= ➥
      "$(CurrentProject.Properties.Item('DefaultNamespace').Value)" />
    </Argument>
    <Argument Name="AssemblyName">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library" />
      <ValueProvider Type="Evaluator" Expression="$(CurrentProject.Name)" />
    </Argument>
    <Argument Name="ClassName">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
    </Argument>
    <Argument Name="TitleWebPart">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
    </Argument>
    <Argument Name="DescriptionWebPart">
      <Converter Type="Microsoft.Practices.RecipeFramework. ➥
      Library.Converters.NamespaceStringConverter, ➥
      Microsoft.Practices.RecipeFramework.Library"/>
    </Argument>
  </Arguments>
```

```
    <GatheringServiceData>
      <Wizard xmlns="http://schemas.microsoft.com/pag/gax-wizards" SchemaVersion="1.0">
        <Pages>
          <Page>
            <Title>
              Collect information using editors, converters and value providers.
            </Title>
            <Fields>
              <Field ValueName="AssemblyName" Label="Assembly Name">
                <Tooltip></Tooltip>
              </Field>
              <Field ValueName="Namespace" Label="Namespace">
                <Tooltip></Tooltip>
              </Field>
              <Field ValueName="ClassName" Label="Classname Web Part">
                <Tooltip></Tooltip>
              </Field>
              <Field ValueName="TitleWebPart" Label="Title Web Part">
                <Tooltip></Tooltip>
              </Field>
              <Field ValueName="DescriptionWebPart" Label="Description Web Part">
                <Tooltip></Tooltip>
              </Field>
            </Fields>
          </Page>
        </Pages>
      </Wizard>
    </GatheringServiceData>
  </Recipe>
```

After customizing the code and making your own templates, build the project and register it. You can do this by right-clicking the project file and selecting Register Guidance Package. This launches a recipe that is associated with the guidance package. The recipe registers the package you are developing on your computer. Registration is a form of installation that you can perform without leaving the Visual Studio development environment. It is also possible to unregister the package. This will reverse the registration.

After registering the guidance package, you can open a new instance of Visual Studio to test the functionality of the package.

ASP.NET 2.0 Server Controls

In this section we will show you a couple of the ASP.NET 2.0 server controls used within a SharePoint web part. Using ASP.NET 2.0 server controls gives the developer great functionality for database access, calendars, text boxes, drop-down lists, and a lot of other common composite web functionality. Controls are very essential to the ASP.NET programming model. In ASP.NET there are nearly 60 new server controls.

The FileUpload Control

File upload is possible in ASP.NET 1.x, but you have to jump through some hoops to get everything working. For example, you have to add `enctype="multipart/form-data"` to the page's `<form>` element. The new ASP.NET 2.0 FileUpload server control makes the process of uploading files to the hosting server as simple as possible.

The FileUpLoad control displays a text box and a Browse button that allow users to select a file to upload to the server. The user specifies the file to upload by entering the fully qualified path to the file on the local computer (for example, C:\Temp\TestFile.txt) in the text box of the control. The user can also select the file by clicking the Browse button and then locating it in the Choose File dialog box. You need to hook up an event handler to a Submit button which calls the `SaveAs()` method of the FileUpLoad control. Via this method you can specify the location where the file will be saved. The file will not be uploaded to the server until the user presses the Submit button. The code used to create a SharePoint web part with the FileUpload control looks like this:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace LoisAndClark.WPLibrary
{
  public class MyWP : WebPart
  {
    FileUpload objFileUpload = new FileUpload();

    protected override void CreateChildControls()
    {
      this.Controls.Add(new System.Web.UI.LiteralControl ➥
      ("Select a file to upload:"));
      this.Controls.Add(objFileUpload);

      Button btnUpload = new Button();
      btnUpload.Text = "Save File";
      this.Load += new System.EventHandler(btnUpload_Click);
      this.Controls.Add(btnUpload);
    }

    private void btnUpload_Click(object sender, EventArgs e)
    {
      string strSavePath = @"c:\temp\";
      if (objFileUpload.HasFile)
      {
        string strFileName = objFileUpload.FileName;
        strSavePath += strFileName;
```

```
        objFileUpload.SaveAs(strSavePath);
      }
      else
      {
        // Let the user know that the file was not uploaded.
      }
    }
  }
}
```

Figure 1-21 shows how this web part looks on a SharePoint site.

Figure 1-21. *The FileUpload control in a web part*

The BulletedList Control

Another new server control is the BulletedList control. This control, as the name says, displays a bulleted list of items in an *ordered* (<ol> HTML element) or *unordered* (<ul> HTML element) fashion. The BulletedList control has extra features such as data binding, support for custom images, and the option to choose bullet style. The bullet style lets you choose the style of the element that precedes the item. For example, you can choose to use numbers, squares, or circles. Child items can be rendered as plain text, hyperlinks, or buttons. The BulletedList control in the example uses a custom image that needs to be placed in a virtual directory on the server. The code looks like this:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace LoisAndClark.WPLibrary
{
  public class MyWP : WebPart
  {
    protected override void CreateChildControls()
    {
```

```
        BulletedList objBullist = new BulletedList();
        objBullist.BulletStyle = BulletStyle.CustomImage;
        objBullist.BulletImageUrl = @"/_layouts/images/favicon.gif";
        objBullist.Items.Add("One");
        objBullist.Items.Add("Two");
        objBullist.Items.Add("Three");
        objBullist.Items.Add("Four");
        this.Controls.Add(objBullist);
    }
  }
}
```

The result of the BulletedList control in a web part is shown in Figure 1-22.

Figure 1-22. *The BulletedList control in a web part*

The Wizard Control

The Wizard server control enables you to build a sequence of steps that are displayed to the end user. You could use the Wizard control to either display or gather information in small steps. The Wizard control lets you define a group of views in which only one view at a time is active and is rendered to the client. Each view consists of four zones: sidebar, header, content, and navigation. The (optional) sidebar part contains an overview of all the steps in the wizard. The header contains the header information. The content part contains whichever control or controls you like. The navigation part consists of buttons to navigate through the steps in the wizard.

When you are constructing a step-by-step process that includes logic for every step taken, use the Wizard control to manage the entire process. The Wizard control navigation buttons fire server-side events whenever the user clicks one of the buttons. The navigation buttons help to navigate to other wizard views on the same page. Navigation can be linear and nonlinear; in other words, you can jump from one view to another or navigate randomly to whichever view you like. All the controls in a wizard view are part of the page so you can access them in code using their control IDs.

A wizard, represented by a `Wizard` object, consists of multiple steps that are each represented by a `WizardStep` control. `WizardStep` controls are added to a wizard via the `Add()` method of the `WizardSteps` property of the `Wizard` object. In this example, we have defined six different steps. Each step contains a text box. The state of the text boxes in the wizard is maintained automatically. The order of the steps is completely based upon the order in which they are

added to the wizard. The `WizardSteps` property of a `Wizard` object contains a collection of steps. Changing the order of steps changes the order in which the end user sees them. Figure 1-23 shows the first step.

Figure 1-23. *Step 1 of the Wizard control in a web part*

The first step, the start step, always has one button called Next. The following steps will have two buttons, as seen in Figure 1-24.

Figure 1-24. *Step 2 of the Wizard control in a web part*

There are six steps in this example. The final step, step 6, will have a Previous and a Finish button, as you can see in Figure 1-25.

Figure 1-25. *The final step of the Wizard control in a web part*

This web part creates a very simple wizard; the Wizard control itself has a lot more options that we will not cover in this chapter. The code for the web part using the Wizard control looks like this:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using Microsoft.SharePoint.WebPartPages;

namespace LoisAndClark.WPLibrary
{
  public class MyWP : WebPart
  {
    protected override void CreateChildControls()
    {
      Wizard objWizard = new Wizard();
      objWizard.HeaderText = "Wizard Header";

      for (int i = 1; i <= 6; i++)
      {
        WizardStepBase objStep = new WizardStep();
        objStep.ID = "Step" + i;
        objStep.Title = "Step " + i;
        TextBox objText = new TextBox();
        objText.ID = "Text" + i;
        objText.Text = "Value for step " + i;
        objStep.Controls.Add(objText);
        objWizard.WizardSteps.Add(objStep);
      }
      this.Controls.Add(objWizard);
    }
  }
}
```

Summary

This chapter explored the incorporation of the new features of ASP.NET 2.0 in SharePoint Products and Technologies 2003. We showed how to use SQL Server 2000 and 2005 as a data store and how to create web parts in Visual Studio .NET 2005. Then we discussed the Guidance Automation Toolkit to enhance the creation of web parts and created a web part library template guidance package. Finally, we showed you how to use a couple of the new ASP.NET 2.0 server controls within a web part.