Determining What Files Need to Be Deployed

Introduction

Deploying an ASP.NET web application entails copying the ASP.NET-related files from the development environment to the production environment. The ASP.NET-related files include ASP.NET web page markup and code and client- and server-side support files. Client-side support files are those files referenced by your web pages and sent directly to the browser - images, CSS files and JavaScript files, for example. Server-side support files include those that are used to process a request on the server-side. This includes configuration files, web services, class files, Typed DataSets, and LINQ to SQL files, among others.

In general, all client-side support files should be copied from the development environment to the production environment, but what server-side support files get copied depends on whether you are explicitly compiling the server-side code into an assembly (a `.dll` file) or if you are having these assemblies auto-generated. This tutorial highlights what files need to be deployed when explicitly compiling the code into an assembly versus having this compilation step occur automatically.

Explicit Compilation Versus Automatic Compilation

ASP.NET web pages are divided into declarative markup and source code. The declarative markup portion includes HTML, Web controls, and databinding syntax; the code portion contains event handlers written in Visual Basic or C# code. The markup and code portions are typically separated into different files: *`WebPage`*`.aspx` contains the declarative markup while *`WebPage`*`.aspx.cs` houses the code.

Consider an ASP.NET page named `Clock.aspx` that contains a Label control whose `Text` property is set to the current date and time when the page loads. The declarative markup portion (in `Clock.aspx`) would contain the markup for a Label Web control - `<asp:Label runat="server" id="TimeLabel" />` - while the code portion (in `Clock.aspx.cs`) would have a `Page_Load` event hander with the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
    TimeLabel.Text = "The time at the beep is: " + DateTime.Now.ToString();
}
```

In order for the ASP.NET engine to service a request for this page, the page's code portion (the *`WebPage`*`.aspx.cs` file) must first be compiled. This compilation can happen explicitly or automatically.

If the compilation happens explicitly then the entire application's source code is compiled into one or more assemblies (`.dll` files) located in the application's `Bin` directory. If the compilation happens automatically then the resulting auto-generated assembly is, by default, placed in the `Temporary ASP.NET Files` folder, which can be found at `%WINDOWS%\Microsoft.NET\Framework\<version>`, although this location is configurable via the `<compilation>` element in `Web.config`. With explicit compilation *you* must take some action to compile the ASP.NET application's code into an assembly, and this step occurs prior to deployment. With automatic compilation the compilation process occurs on the web server when the resource is first accessed.

Regardless of what compilation model you use, the markup portion of all ASP.NET pages (the *WebPage*`.aspx` files) need to be copied to the production environment. With explicit compilation you need to copy up the assemblies in the `Bin` folder, but you do not need to copy up the ASP.NET pages' code portions (the *WebPage*`.aspx.cs` files). With automatic compilation you need to copy up the code portion files so that the code is present and can be compiled automatically when the page is visited. The markup portion of each ASP.NET web page includes a `@Page` directive with attributes that indicate whether the page's associated code was already explicitly compiled or whether it needs to be automatically compiled. As a result, the production environment can work with either compilation model seamlessly and you do not need to apply any special configuration settings to indicate that explicit or automatic compilation is used.

Table 1 summarizes the different files to deploy when using explicit compilation versus automatic compilation. Note that regardless of the compilation model used you should always deploy the assemblies in the `Bin` folder, if that folder exists. The `Bin` folder contains the assemblies specific to the web application, which include the compiled source code when using the explicit compilation model. The `Bin` directory also contains assemblies from other projects and any open-source or third-party assemblies you may be using, and these need to be on the production server. Therefore, as a general rule of thumb, copy the `Bin` folder up to production when deploying. (If you are using the automatic compilation model and are not using any external assemblies then you won't have a `Bin` directory - that's OK!)

Compilation Model	**Deploy Markup Portion File?**	**Deploy Source Code File?**	**Deploy Assemblies in `Bin` Directory?**
Explicit Compilation	Yes	No	Yes
Automatic Compilation	Yes	Yes	Yes (if it exists)

Table 1: What files you deploy depends on the compilation model used.

Taking a Trip Down Memory Lane

What compilation approach is used depends, in part, on how the ASP.NET application is managed in Visual Studio. Since .NET's inception in the year 2000 there have been four different versions of Visual Studio - Visual Studio .NET 2002, Visual Studio .NET 2003, Visual Studio 2005, and Visual Studio 2008. Visual Studio .NET 2002 and 2003 managed ASP.NET applications using the *Web Application Project model*. The key features of the Web Application Project model are:

- The files that makeup the project are defined in a single project file. Any files *not* defined in the project file are not considered part of the web application by Visual Studio.
- Uses explicit compilation. Building the project compiles the code files within the project into a single assembly that is placed in the `Bin` folder.

When Microsoft released Visual Studio 2005 they dropped support for the Web Application Project model and replaced it with the *Web Site Project model*. The Web Site Project model differentiated itself from the Web Application Project model in the following ways:

- Rather than having a single project file that spells out the project's files, the file system is used instead. In short, any files within the web application folder (or subfolders) are considered part of the project.
- Building a project in Visual Studio does not create an assembly in the `Bin` directory. Instead, building a Web Site Project reports any compile-time errors.
- Support for automatic compilation. Web Site Projects are typically deployed by copying the markup and source code to the production environment, although the code can be precompiled (explicit compilation).

Microsoft revived the Web Application Project model when it released Visual Studio 2005 Service Pack 1. However, Visual Web Developer continued to only support the Web Site Project model. The good news is that this limitation was dropped with Visual Web Developer 2008 Service Pack 1. Today you can create ASP.NET applications in Visual Studio (and Visual Web Developer) using either the Web Application Project model or the Web Site Project model. Both models have their pros and cons. Refer to Introduction to Web Application Projects: Comparing Web Site Projects and Web Application Projects for a comparison of the two models and to help decide what project model works best for your situation.

Exploring the Sample Web Application

The download for this tutorial includes an ASP.NET application called Book Reviews. The website mimics a hobby website someone might create to share their book reviews with the online community. This ASP.NET web application is very simple and consists of the following resources:

- `Web.config`, the application's configuration file.
- A master page (`Site.master`).
- Seven different ASP.NET pages:
    - `~/Default.aspx` - the site's homepage.
    - `~/About.aspx` - an "About the Site" page.
    - `~/Fiction/Default.aspx` - a page listing the fiction books that have been reviewed.
        - `~/Fiction/Blaze.aspx` - a review of the Richard Bachman novel *Blaze*.
    - `~/Tech/Default.aspx` - a page listing the technology books that have been reviewed.
        - `~/Tech/CYOW.aspx` - a review of *Create Your Own Website*.
        - `~/Tech/TYASP35.aspx` - a review of *Teach Yourself ASP.NET 3.5 in 24 Hours*.
- Three different CSS files in the `Styles` folder.
- Four image files - a Powered by ASP.NET logo and images of the covers of the three reviewed books - all located in the `Images` folder.
- A `Web.sitemap` file, which defines the site map and is used to display menus in the `Default.aspx` pages in the root directory and `Fiction` and `Tech` folders.
- A class file named `BasePage.cs` that defines a base `Page` class. This class extends the functionality of the `Page` class by automatically setting the `Title` property based on the page's position in the site map. In a nutshell, any ASP.NET code-behind class that extends `BasePage` (instead of `System.Web.UI.Page`) will have its title set to a value depending on its position in the site map. For instance, when viewing the `~/Tech/CYOW.aspx` page, the title is set to "Home : Technology : Create Your Own Website".

Figure 1 shows a screen shot of the Book Reviews website when viewed through a browser. Here you see the page `~/Tech/TYASP35.aspx`, which reviews the book *Teach Yourself ASP.NET 3.5 in 24 Hours*. The breadcrumb that spans the top of the page and the menu in the left column are based on the site map structure defined in `Web.sitemap`. The image in the right upper corner is one of the book cover images located in the `Images` folder. The website's look and feel are defined via cascading style sheet rules spelled out by the CSS files in the `Styles` folder, while the overarching page layout is defined in the master page, `Site.master`.

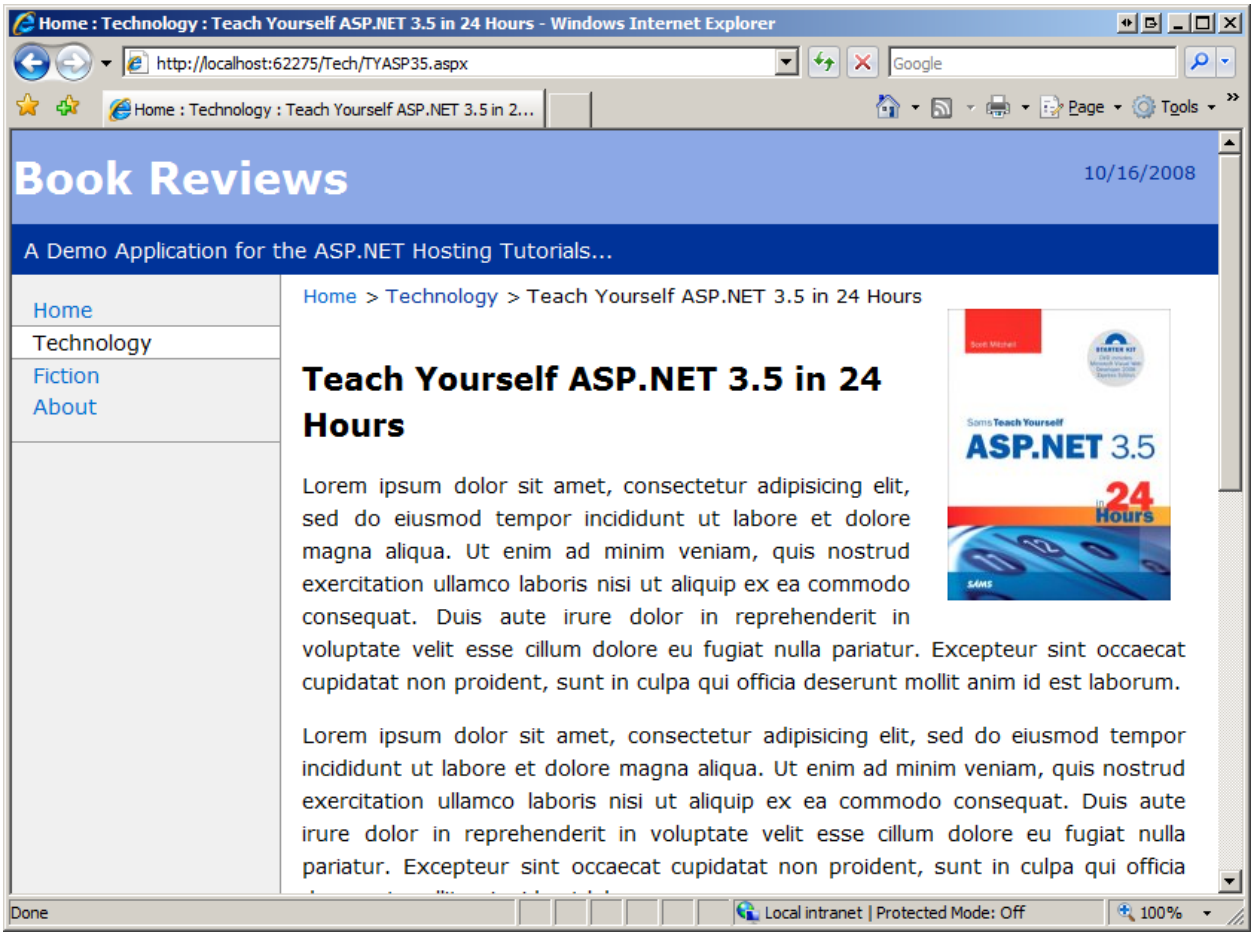

Figure 1: The Book Reviews website offers reviews on an assortment of titles.

This application does not use a database; each review is implemented as a separate web page in the application. This tutorial (and the next several tutorials) walk through deploying a web application that does not have a database. However, in a future tutorial we will enhance this application to store reviews, reader comments, and other information within a database, and will explore what steps need to be performed to correctly deploy a data-driven web application.

> **Note:** These tutorials focus on hosting ASP.NET applications with a web host provider and do not explore ancillary topics like ASP.NET's site map system or using a base `Page` class. For more information on these technologies, and for more background on other topics covered throughout the tutorial, refer to the Further Reading section at the end of each tutorial.

This tutorial's download has two copies of the web application, each implemented as a different Visual Studio project type: BookReviewsWAP, a Web Application Project, and BookReviewsWSP, a Web Site Project. Both projects were created with Visual Web Developer 2008 SP1 and use ASP.NET 3.5 SP1. To work with these projects start by unzipping the contents to your Desktop. To open the Web Application Project (BookReviewsWAP), navigate to the `BookReviewsWAP` folder and double-click the Solution

file, `BookReviewsWAP.sln`. To open the Web Site Project (BookReviewsWSP), launch Visual Studio and then, from the File menu, choose the Open Web Site option, browse to the `BookReviewsWSP` folder on your Desktop, and click OK.

The remaining two sections in this tutorial look at what files you will need to copy to the production environment when deploying the application. The next two tutorials - *Deploying Your Site Using FTP* and *Deploying Your Site Using Visual Studio* - show different ways to copy these files to a web host provider.

Determining the Files to Deploy for the Web Application Project

The Web Application Project model uses explicit compilation - the project's source code is compiled into a single assembly each time you build the application. This compilation includes the ASP.NET pages' code-behind files (`~/Default.aspx.cs`, `~/About.aspx.cs`, and so on), as well as the `BasePage.cs` class. The resulting assembly is named `BookReviewsWAP.dll` and is located in the application's `Bin` directory.

Figure 2 shows the files that make up the Book Reviews Web Application Project.

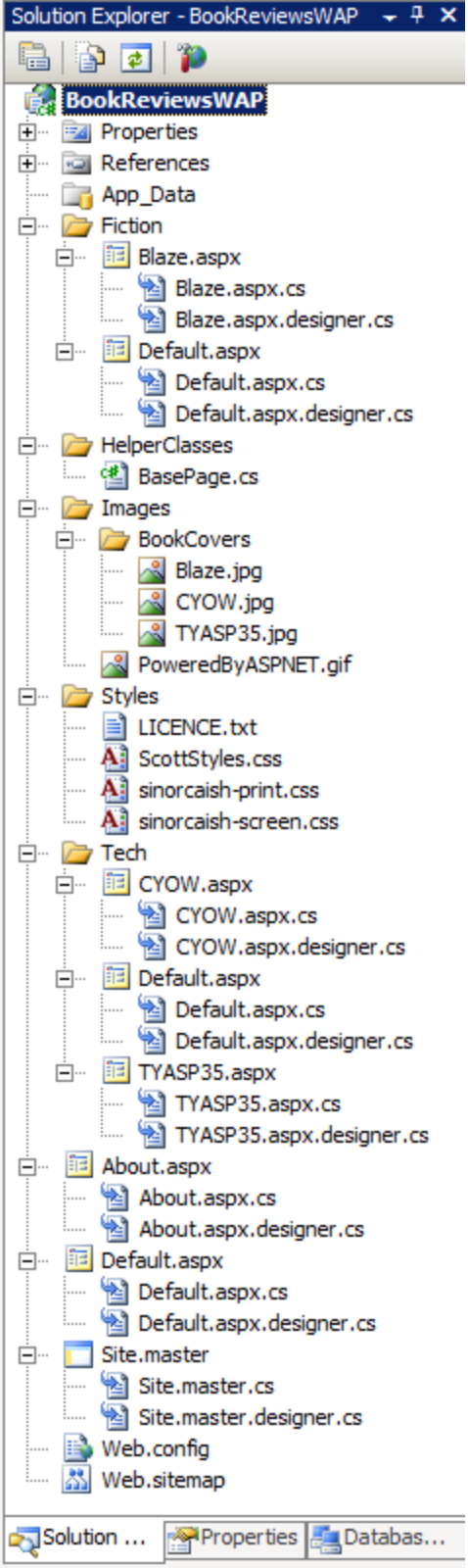

Figure 2: The Solution Explorer lists the files that comprise the Web Application Project.

To deploy a ASP.NET application developed using the Web Application Project model start by building the application so as to explicitly compile the most recent source code into an assembly. Next, copy the following files to the production environment:

- The files that contain the declarative markup for every ASP.NET page, such as `~/Default.aspx`, `~/About.aspx`, and so on. Also, copy up the declarative markup for any master pages and User Controls.
- The assemblies (`.dll` files) in the `Bin` folder. You do not need to copy the program database files (`.pdb`) or any XML files you may find in the `Bin` directory.

You do not need to copy the ASP.NET pages' source code files to the production environment, nor do you need to copy the `BasePage.cs` class file.

> **Note:** As Figure 2 shows, the `BasePage` class is implemented as a class file in the project, placed in folder named `HelperClasses`. When the project is compiled the code in the `BasePage.cs` file is compiled along with the ASP.NET pages' code-behind classes into the single assembly, `BookReviewsWAP.dll`. ASP.NET has a special folder named `App_Code` that is designed to hold class files for Web Site Projects. The code in the `App_Code` folder is automatically compiled and therefore should **not** be used with Web Application Projects. Instead, you should place your application's class files in a normal folder named `HelperClasses`, or `Classes`, or something similar. Alternatively, you can place class files in a separate Class Library project.

In addition to copying the ASP.NET-related markup files and the assembly in the `Bin` folder, you also need to copy the client-side support files - the images and CSS files - as well as the other server-side support files, `Web.config` and `Web.sitemap`. These client- and server-side support files need to be copied to the production environment regardless of whether you use explicit or automatic compilation.

Determining the Files to Deploy for the Web Site Project Files

The Web Site Project model supports automatic compilation, a feature not available when using the Web Application Project model. With explicit compilation you must compile your project's source code into an assembly and copy that assembly to the production environment. On the other hand, with automatic compilation you simply copy the source code to the production environment and it is compiled by the runtime on demand as needed.

The Build menu option in Visual Studio is present in both Web Application Projects and Web Site Projects. Building a Web Application Projects compiles the project's source code into a single assembly located in the `Bin` directory; building a Web Site Project checks for any compile-time errors, but does not create any assemblies. To deploy an ASP.NET application developed using the Web Site Project model all you need to do is copy the appropriate files to the production environment, but I would encourage you to first build the project to ensure that there are no compile-time errors.

Figure 3 shows the files that make up the Book Reviews Web Site Project.

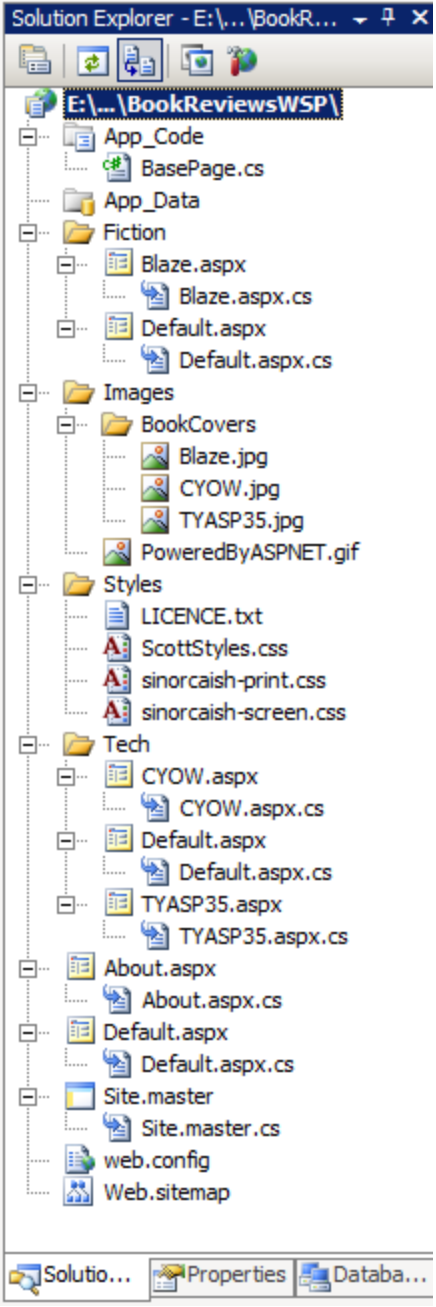

Figure 3: The Solution Explorer lists the files that comprise the Web Site Project..

Deploying a Web Site Project involves copying all of the ASP.NET-related files to the production environment - that includes the markup pages for ASP.NET pages, master pages, and User Controls, along with their code files. You also need to copy up any class files, such as `BasePage.cs`. Note that the `BasePage.cs` file is located in the `App_Code` folder, which is a special ASP.NET folder used in Web Site Projects for class files. The special folder needs to be created on production, as well, as the class files in the `App_Code` folder on the development environment must be copied to the `App_Code` folder on production.

In addition to copying the ASP.NET markup and source code files, you also need to copy the client-side support files - the images and CSS files - as well as the other server-side support files, `Web.config` and `Web.sitemap`.

Note: Web Site Projects can also use explicit compilation. A future tutorial will examine how to explicitly compile a Web Site Project.

Summary

Deploying an ASP.NET application entails copying the necessary files from the development environment to the production environment. The precise set of files that need to be synced depends on whether the ASP.NET application's code is explicitly or automatically compiled. The compilation strategy employed is influenced by whether Visual Studio is configured to manage the ASP.NET application using the Web Application Project model or the Web Site Project model.

The Web Application Project model uses explicit compilation and compiles the project's code into a single assembly in the `Bin` folder. When deploying the application, the markup portion of the ASP.NET pages and the contents of the `Bin` folder must be pushed up to the production environment; the source code in the application - the code files and code-behind classes, for example - do not need to be copied to the production environment.

The Web Site Project model uses automatic compilation by default, although it is possible to explicitly compile a Web Site Project, as we will see in future tutorials. Deploying an ASP.NET application that uses automatic compilation requires that the markup portion *and* source code must be copied to the production environment. The code is automatically compiled on the production environment when it is requested for the first time.

Now that we have examined what files need to be synced between the development and production environments we are ready to deploy the Book Reviews application to a web host provider.

Happy Programming!

Further Reading

For more information on the topics discussed in this tutorial, refer to the following resources:

- ASP.NET Compilation Overview
- ASP.NET User Controls
- Examining ASP.NET's Site Navigation
- Introduction to Web Application Projects
- Master Page Tutorials
- Sharing Code Between Pages
- Using a Custom Base Class For Your ASP.NET Pages' Code-Behind Classes
- Visual Studio 2005's Web Site Project System: What Is It and Why Did We Do It?

- Walkthrough: Converting a Web Site Project to a Web Application Project in Visual Studio

About the Author

Scott Mitchell, author of multiple ASP/ASP.NET books and founder of 4GuysFromRolla.com, has been working with Microsoft Web technologies since 1998. Scott works as an independent consultant, trainer, and writer. His latest book is *Sams Teach Yourself ASP.NET 3.5 in 24 Hours*. Scott can be reached at mitchell@4guysfromrolla.com or via his blog at http://ScottOnWriting.NET.

Special Thanks To...

Interested in reviewing my upcoming MSDN articles? If so, drop me a line at mitchell@4GuysFromRolla.com.