

Chapter 3

You can add resource files that are page-specific to the `\App_LocalResources` folder by constructing the name of the `.resx` file in the following manner:

- `Default.aspx.resx`
- `Default.aspx.fi.resx`
- `Default.aspx.ja.resx`
- `Default.aspx.en-gb.resx`

Now, the resource declarations used on the `Default.aspx` page are retrieved from the appropriate file in the `\App_LocalResources` folder. By default, the `Default.aspx.resx` resource file is used if another match is not found. If the client is using a culture specification of `fi-FI` (Finnish), however, the `Default.aspx.fi.resx` file is used instead. Localization of local resources are covered in detail in Chapter 30.

`\App_WebReferences`

The `\App_WebReferences` folder is a new name for the previous `Web References` folder used in previous versions of ASP.NET. Now you can use the `\App_WebReferences` folder and have automatic access to the remote Web services referenced from your application. Web services in ASP.NET are covered in Chapter 29.

`\App_Browsers`

The `\App_Browsers` folder holds `.browser` files, which are XML files used to identify the browsers making requests to the application and understanding the capabilities these browsers have. You can find a list of globally accessible `.browser` files at `C:\Windows\Microsoft.NET\Framework\v2.0xxxxx\CONFIG\Browsers`. In addition, if you want to change any part of these default browser definition files, just copy the appropriate `.browser` file from the `Browsers` folder to your application's `\App_Browsers` folder and change the definition.

Compilation

You already saw how Visual Studio 2005 compiles pieces of your application as you work with them (for instance, by placing a class in the `\App_Code` folder). The other parts of the application, such as the `.aspx` pages, can be compiled just as they were in ASP.NET 1.0/1.1 by referencing the pages in the browser.

When an ASP.NET page is referenced in the browser for the first time, the request is passed to the ASP.NET parser that creates the class file in the language of the page. It is passed to the ASP.NET parser based on the file's extension (`.aspx`) because ASP.NET realizes that this file extension type is meant for its handling and processing. After the class file has been created, the class file is compiled into a DLL and then written to the disk of the Web server. At this point, the DLL is instantiated and processed, and an output is generated for the initial requester of the ASP.NET page. This is detailed in Figure 3-11.

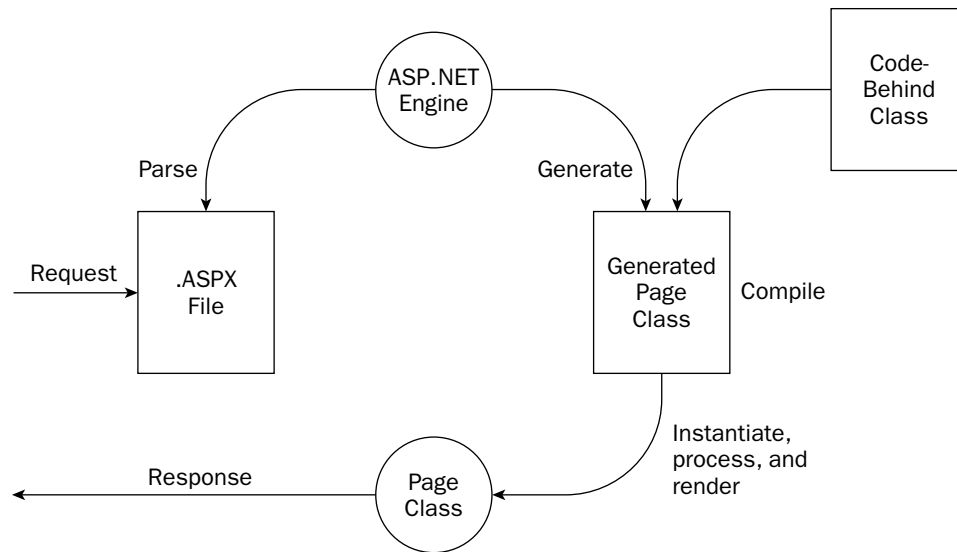


Figure 3-11

On the next request, great things happen. Instead of going through the entire process again for the second and respective requests, the request simply causes an instantiation of the already-created DLL, which sends out a response to the requester. This is illustrated in Figure 3-12.

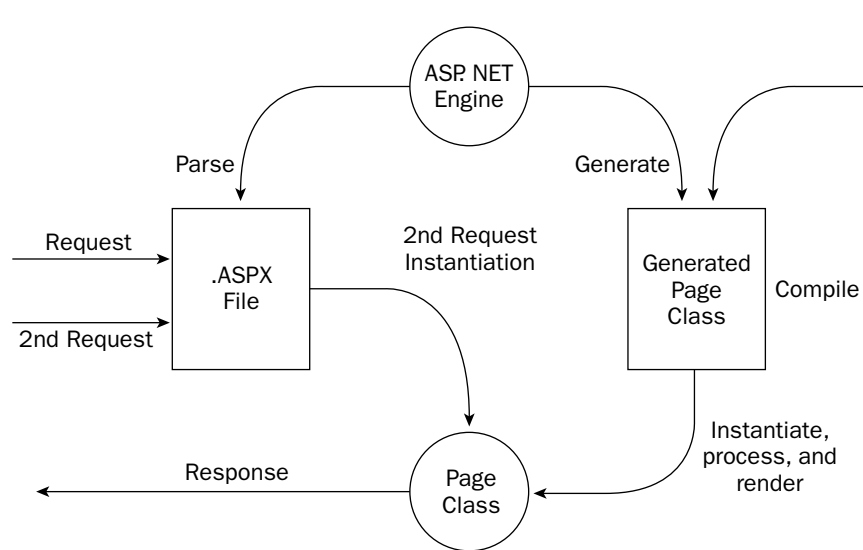


Figure 3-12

Chapter 3

Because of the mechanics of this process, if you made changes to your .aspx code-behind pages, you found it necessary to recompile your application. This was quite a pain if you had a larger site and didn't want your end users to experience the extreme lag that occurs when an .aspx page is referenced for the first time after compilation. Many developers, consequently, began to develop their own tools that automatically go out and hit every single page within their application to remove this first-time lag hit from the end user's browsing experience.

ASP.NET 2.0 introduces a few ways to precompile your entire application with a single command that you can issue through a command line. One type of compilation is referred to as *in-place precompilation*. In order to precompile your entire ASP.NET application, you must use the `aspnet_compiler.exe` tool that now comes with ASP.NET 2.0. You navigate to the tool using the Command window. Open the Command window and navigate to `C:\Windows\Microsoft.NET\Framework\v2.0.xxxxx\`. When you are there, you can work with the `aspnet_compiler` tool. You can also get to this tool directly by pulling up the Visual Studio 2005 Command Prompt. Choose `Start→All Programs→Microsoft Visual Studio 2005→Visual Studio Tools→Visual Studio 2005 Command Prompt`.

After you get the command prompt, you use the `aspnet_compiler.exe` tool to perform an in-place precompilation using the following command:

```
aspnet_compiler -p "C:\Inetpub\wwwroot\WROX" -v none
```

You then get a message stating that the precompilation is successful. The other great thing about this precompilation capability is that you can also use it to find errors on any of the ASP.NET pages in your application. Because it hits each and every page, if one of the pages contains an error that won't be triggered until runtime, you get notification of the error immediately as you employ this precompilation method.

The next precompilation option is commonly referred to as *precompilation for deployment*. This is an outstanding new addition to ASP.NET that enables you to compile your application down to some DLLs, which can then be deployed to customers, partners, or elsewhere for your own use. Not only are minimal steps required to do this, but after your application is compiled, you only have to move around the DLL and some placeholder files for the site to work. This means that your Web site code is completely removed and placed in the DLL when deployed.

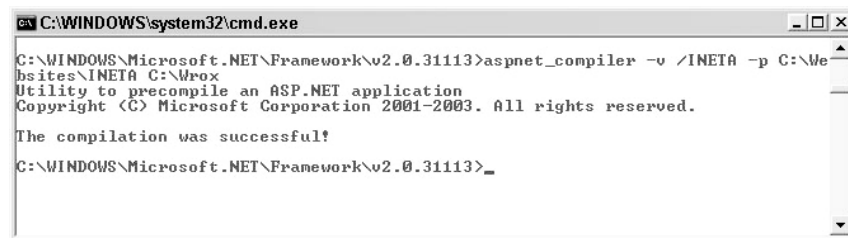
Before you do, however, create a folder in your root drive called, for example, `Wrox`. This folder is the one to which you will direct the compiler output. When it is in place, you can return to the compiler tool and give the following command:

```
aspnet_compiler -v [Application Name] -p [Physical Location] [Target]
```

So, if you have an application called `INETA` located at `C:\Websites\INETA`, you use the following commands:

```
aspnet_compiler -v /INETA -p C:\Websites\INETA C:\Wrox
```

Press the Enter key, and the compiler either tells you that it has a problem with one of the command parameters or that it was successful (shown in Figure 3-13). If it was successful, you can see the output placed in the target directory.



```
C:\WINDOWS\system32\cmd.exe
C:\WINDOWS\Microsoft.NET\Framework\v2.0.31113>aspnet_compiler -v /INETA -p C:\We
bsites\INETA C:\Wrox
Utility to precompile an ASP.NET application
Copyright (C) Microsoft Corporation 2001-2003. All rights reserved.

The compilation was successful!
C:\WINDOWS\Microsoft.NET\Framework\v2.0.31113>
```

Figure 3-13

In the example just shown, `-v` is a command for the virtual path of the application, which is provided by using `/INETA`. The next command is `-p`, which is pointing to the physical path of the application. In this case, it is `C:\Websites\INETA`. Finally, the last bit, `C:\Wrox`, is the location of the compiler output. The following table describes the possible commands for the `aspnet_compiler.exe` tool.

| Command | Description |
|--------------------------|---|
| <code>-m</code> | Specifies the full IIS metabase path of the application. If you use the <code>-m</code> command, you cannot use the <code>-v</code> or <code>-p</code> command. |
| <code>-v</code> | Specifies the virtual path of the application to be compiled. If you also use the <code>-p</code> command, the physical path is used to find the location of the application. |
| <code>-p</code> | Specifies the physical path of the application to be compiled. If this is not specified, the IIS metabase is used to find the application. |
| <code>[targetDir]</code> | Specifies the target directory where the compiled files should be placed. If this is not specified, the output files are placed in the application directory. |

After compiling the application, you can go to `C:\Wrox` to see the output. Here, you see all the files and the file structures that were in the original application. But if you look at the content of one of the files, notice that the file is simply a placeholder. In the actual file, you find the following comment:

```
This is a marker file generated by the precompilation tool
and should not be deleted!
```

In fact, you find a `Code.dll` file in the `bin` folder where all the page code is located. Because it is in a DLL file, it provides great code obfuscation as well. From here on, all you do is move these files to another server using FTP or Windows Explorer, and you can run the entire Web application from these files. When you have an update to the application, you simply provide a new set of compiled files. A sample output is displayed in Figure 3-14.

Chapter 3

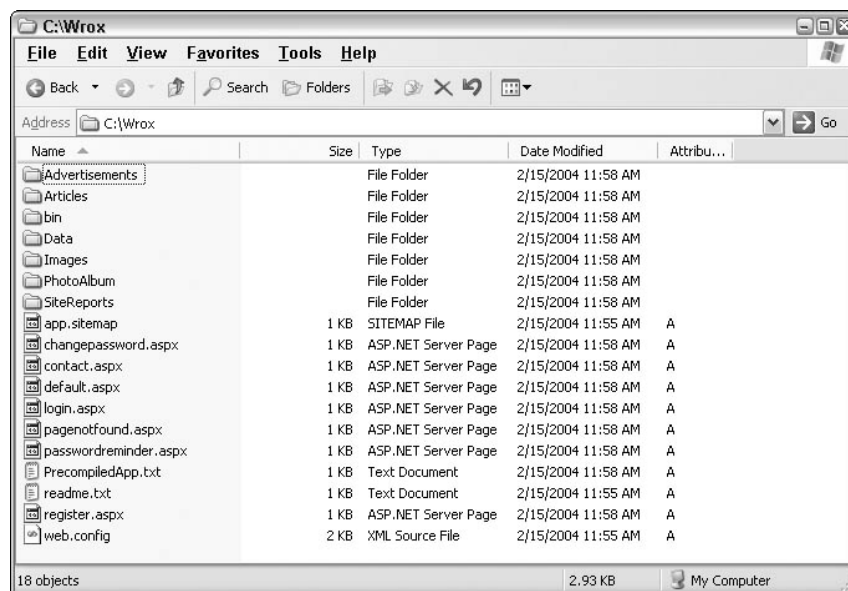


Figure 3-14

Note that this compilation process doesn't compile *every* type of Web file. In fact, it compiles only the ASP.NET-specific file types and leaves out of the compilation process the following types of files:

- HTML files
- XML files
- XSD files
- web.config files
- Text files

You can't do much to get around this, except in the case of the HTML files and the text files. For these file types, just change the file extension of these file types to `.aspx`; they are then compiled into the `Code.dll` like all the other ASP.NET files.

Build Providers

As you review the various ASP.NET folders, note that one of the more interesting folders is the `\App_Code` folder. You can simply drop code files, XSD files, and even WSDL files directly into the folder for automatic compilation. When you drop a class file into the `\App_Code` folder, the class can automatically be utilized by a running application. Before ASP.NET 2.0, if you wanted to deploy a custom component, you had to precompile the component before being able to utilize it within your application. Now ASP.NET simply takes care of all the work that you once had to do. You don't need to perform any compilation routine.