**ASP.NET Compilation Overview**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/ms178466(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms178466(VS.85).aspx)

[Microsoft Visual Studio 2010/.NET Framework 4.0](http://msdn.microsoft.com/en-us/library/ms178466(VS.100).aspx)

In order for application code to service requests by users, ASP.NET must first compile the code into one or more assemblies. Assemblies are files that have the file name extension .dll. You can write ASP.NET code in many different languages, such as Visual Basic, C#, J#, and others. When the code is compiled, it is translated into a language-independent and CPU-independent representation called Microsoft Intermediate Language (MSIL). At run time, MSIL runs in the context of the .NET Framework, which translates MSIL into CPU-specific instructions for the processor on the computer running the application.

There are many benefits to compiling application code including:

* **Performance**Compiled code is much faster than scripting languages such as ECMAScript or VBScript because it is a closer representation to machine code and does not require additional parsing.
* **Security**Compiled code is more difficult to reverse engineer than non-compiled source code because it lacks the readability and abstraction of a high-level language. Additionally, there are obfuscation tools that make compiled code even more resistant to reverse engineering.
* **Stability**   Code is checked at compile time for syntax errors, type safety, and other problems. By catching these errors at build-time you can eliminate many errors in your code.
* **Interoperability**   Because MSIL code supports any .NET language, you can use assemblies that were originally written in other languages in your code. For example, if you are writing an ASP.NET Web page in C#, you can add a reference to a .dll file that was written in Visual Basic.

The ASP.NET compilation architecture includes a number of features including:

* Multiple language support.
* Automatic compilation.
* Flexible deployment.
* Extensible build system.

The following sections describe each of these features.

http://i.msdn.microsoft.com/Global/Images/clear.gif Multiple Language Support

In ASP.NET 2.0 you can use different languages such as Visual Basic and C# in the same application because ASP.NET will create multiple assemblies, one for each language. For code stored in the App\_Code folder, you can specify a subfolder for each language. For more information on the App\_Code folder, see [Shared Code Folders in ASP.NET Web Sites](http://msdn.microsoft.com/en-us/library/t990ks23.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Automatic Compilation

ASP.NET automatically compiles your application code and any dependent resources the first time a user requests a resource from the Web site. In general, ASP.NET creates an assembly for each application directory (such as App\_Code) and one for the main directory. (If files in a directory are in different programming languages, then separate assemblies will be created for each language.) You can specify which directories are compiled into single assemblies in the [Compilation](http://msdn.microsoft.com/en-us/library/system.web.configuration.systemwebsectiongroup.compilation.aspx) section of the Web.config file.

http://i.msdn.microsoft.com/Global/Images/clear.gif Flexible Deployment

Because ASP.NET compiles your Web site on first user request, you can simply copy your application's source code to the production Web server. However, ASP.NET also provides precompilation options that allow you to compile your Web site before it has been deployed, or to compile it after it has been deployed but before a user requests it. Precompilation has several advantages. It can improve the performance of your Web site on first request because there will be no lag time while ASP.NET compiles the site. Precompiling can also help you find errors that might otherwise be found only when a user requests a page. Finally, if you precompile the Web site before you deploy it, you can deploy the assemblies instead of the source code.

You can precompile a Web site using the ASP.NET compiler tool (ASPNET\_Compiler.exe). The tool that provides the following precompilation options:

* **In-place compilation**   This option performs the same compilation that occurs during dynamic compilation. Use this option to compile a Web site that has already been deployed to a production server.
* **Non-updateable full precompilation**   Use this to compile an application and then copy the compiled output to the production server. All application code, markup, and UI code is compiled into assemblies. Placeholder files such as .aspx pages still exist so that you can perform file-specific tasks such as configure permissions, but the files contain no updateable code. In order to update any page or any code you must precompile the Web site again and deploy it again.
* **Updateable precompilation**   This is similar to non-updateable full precompilation, except that UI elements such as .aspx pages and .ascx controls retain all their markup, UI code, and inline code, if any. You can update code in the file after it has been deployed; ASP.NET will detect changes to the file and recompile it. Note that code in a code-behind file (.vb or .cs file) built into assemblies during precompilation, and you therefore cannot change it without going through the precompilation and deployment steps again.

For more information, see [ASP.NET Precompilation Overview](http://msdn.microsoft.com/en-us/library/bb398860.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Extensible Build System

ASP.NET uses [BuildProvider](http://msdn.microsoft.com/en-us/library/system.web.compilation.buildprovider.aspx) classes to build items such as .aspx pages, .ascx files, and global resources. You can extend and customize the ASP.NET build system to compile custom resources by creating classes that inherit from the [BuildProvider](http://msdn.microsoft.com/en-us/library/system.web.compilation.buildprovider.aspx) class. For example, you could add a new file type and then write a [BuildProvider](http://msdn.microsoft.com/en-us/library/system.web.compilation.buildprovider.aspx) that builds that particular type.

**Understanding ASP.NET Dynamic Compilation**

This page is specific to

**Microsoft Visual Studio 2008/.NET Framework 3.5**

Other versions are also available for the following:

[Microsoft Visual Studio 2005/.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/ms366723(VS.80).aspx)

[.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms366723(VS.85).aspx)

[Microsoft Visual Studio 2010/.NET Framework 4.0](http://msdn.microsoft.com/en-us/library/ms366723(VS.100).aspx)

In order for your Web application to service requests, ASP.NET must first parse and compile the code of your Web application into one or more assemblies. When the code is compiled, it is translated into a language-independent and CPU-independent representation called Microsoft Intermediate Language (MSIL). At run time, MSIL runs in the context of the .NET Framework, which translates MSIL into CPU-specific instructions for the processor on the computer running the application.

ASP.NET dynamic compilation enables you to modify your source code without having to explicitly compile your code before you deploy your Web application. If you modify a source file, ASP.NET automatically recompiles the file and updates all linked resources. The IIS server does not have to be restarted for the changes to take effect unless the <processModel> section has been changed.

You can extend the ASP.NET build system by creating custom build providers for new file types that are called during compilation.

http://i.msdn.microsoft.com/Global/Images/clear.gif Compiling on First Request

By default, ASP.NET Web pages and code files are compiled dynamically when users first request a resource, such as an ASP.NET page (.aspx file), from a Web site. After pages and code files have been compiled the first time, the compiled resources are cached, so that subsequent requests to the same page are extremely efficient.

ASP.NET supports the dynamic compilation of ASP.NET pages (.aspx files), ASP.NET Web services (.asmx files), ASP.NET HTTP handlers (.ashx files) and ASP.NET application files (Global.asax), as well as other files, such as source code and class files. For more information about ASP.NET file types, see [Web Site File Types](http://msdn.microsoft.com/en-us/library/2wawkw1c.aspx). For more information about the ASP.NET compilation process, see the "Compilation Life Cycle" section of [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Recompiling on Change

Any changes to a dynamically compiled file will automatically invalidate the file's cached compiled assembly and trigger recompilation of all affected resources. The next time a request to the code is made, ASP.NET recognizes that the code has changed and recompiles the affected resources of the Web application. This system enables you to quickly develop applications with a minimum of compilation processing overhead. (Note that depending on the change to the resources, the result can range from recompiling a single page to recompiling the whole Web site.)

http://i.msdn.microsoft.com/Global/Images/clear.gif Compilation Dependencies

When the first request is made to an application, ASP.NET compiles files in a specific order. The first items to be compiled are referred to as the top-level items. After the first request, the top-level items are recompiled only if a dependency changes.

Top-level items include the App\_GlobalResources folder, the App\_WebResources folder, profile properties, the App\_Code folder, and the Global.asax file. After the top-level items are compiled, ASP.NET compiles additional items. These items include the App\_LocalResources folder, individual ASP.NET pages (.aspx files), ASP.NET user controls (.ascx files), ASP.NET HTTP Handlers (.ashx files), and ASP.NET HTTP modules (.asmx files), as well as themes, master pages, and other source files.

For more information, see [ASP.NET Web Site Layout](http://msdn.microsoft.com/en-us/library/ex526337.aspx) and the "Compilation Life Cycle" section of [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Compilation Output

When your code is compiled, the resulting assemblies are cached in a folder on the server. This folder requires appropriate permissions so that your code compiles and runs correctly. You can configure both the compilation folder location and the permissions under which your code compiles and operates.

**Compilation Folder Location**

By default, when you compile a Web application the compiled code is placed in the Temporary ASP.NET Files folder. This folder is a subdirectory of the location where you installed the .NET framework. Typically, the location is the following:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl00');" \o "Copy Code)

%SystemRoot%\Microsoft.NET\Framework\versionNumber\Temporary ASP.NET Files

**Compilation Folder Required Permissions**

The .NET installation process creates the Temporary ASP.NET Files folder and assigns access permissions to the ASP.NET local user account, which has the high-trust permissions needed to access your compiled code. If you modify your configuration or account settings, you must make sure that the account you use has high-trust permissions to the Temporary ASP.NET Files folder. For additional details, see [How to: Run the Worker Process Under a User Account](http://msdn.microsoft.com/en-us/library/bakfs900.aspx).

**Compilation Folder Configurability**

ASP.NET creates a discrete subfolder under the Temporary ASP.NET File folder for each application. You can configure the root location using the tempDirectory attribute of the compilation section of the configuration file. This optional attribute enables you to specify the directory to use for temporary file storage during compilation. The default is an empty string (""). In the case of an empty string, and if the current process has the required access permissions, the files are stored in the following directory:

[Copy Code](javascript:CopyCode('ctl00_MTCS_main_ctl35_ctl00_ctl02');" \o "Copy Code)

%FrameworkInstallLocation%\Temporary ASP.NET Files

For more information, see [compilation Element (ASP.NET Settings Schema)](http://msdn.microsoft.com/en-us/library/s10awwz0.aspx) and the [TempDirectory](http://msdn.microsoft.com/en-us/library/system.web.configuration.compilationsection.tempdirectory.aspx) property of the [CompilationSection](http://msdn.microsoft.com/en-us/library/system.web.configuration.compilationsection.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Multiple Language Support

ASP.NET 2.0 supports multiple programming languages in the same Web application. In the App\_Code directory, you can specify a subfolder for each language, such as C# and Visual Basic. ASP.NET will create a separate assembly for each subfolder. For more information, see [Shared Code Folders in ASP.NET Web Sites](http://msdn.microsoft.com/en-us/library/t990ks23.aspx) and [Walkthrough: Develop Web Sites Using Multiple Programming Languages](http://msdn.microsoft.com/en-us/library/ms366714.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Optimizing Dynamic Compilation

By default, when any change is made to a *top-level* file in a Web site, the whole site is recompiled. Top-level files include the global.asax file and all files in the bin and App\_Code folders. It is safest to recompile everything when one of these files changes because other files in the site, such as .aspx and .ascx files, may reference the objects created by code in top-level files.

While recompiling everything works fine for most applications, it could cause a very large application to be unavailable for long periods of time even when minor changes have been made to it. If the application is large enough, it could be unavailable for five to ten minutes or more after a change is made.

If you want to be able to change top-level files without causing the whole site to be recompiled, you can set the optimizeCompilations attribute of the [compilation](http://msdn.microsoft.com/en-us/library/s10awwz0.aspx) element in the Web.config file to true.

|  |
| --- |
| **NoteNote:** |
| You must install a [hotfix](http://go.microsoft.com/fwlink/?LinkId=148964) to make the optimizeCompilations attribute available in the .NET Framework version 3.5. |

If optimizeCompilations is true, when you change a top-level file only the affected files are recompiled. This saves time but can cause run-time errors depending on the type of changes you make to a top-level file.

The following kinds of changes are generally safe:

* Changing a method implementation. Because the signature is not changed, pages compiled against the old version can call the method without throwing an exception.
* Adding new methods or properties. Because these did not previously exist, no already-compiled pages will reference them, and no exceptions will be thrown.
* Adding a CLR attribute to an existing member. This is a typical Dynamic Data scenario where you add attributes like DisplayName to properties. Because CLR attributes are discovered at runtime through reflection, existing pages do not have to be recompiled.

The following kinds of changes may cause run-time exceptions:

* Renaming or deleting methods or properties. If the affected member is referenced by an already-compiled page, an exception will be thrown.
* Changing the signature of a method, or the type of a property. If the affected member is referenced by an already-compiled page, an exception will be thrown. Some signature changes would not cause compile or run-time errors if the whole site is recompiled. For example, the code Response.Write(ClassA.MethodA() in an .aspx page will compile and run fine whether MethodA returns an int or a short. But if the .aspx page is already compiled and you change the return type of MethodA from int to short without recompiling, a runtime exception will be thrown because the compiled code expects the int signature.

If you want to use the optimizeCompilations attribute to minimize dynamic compilation time, you should carefully review each change you make to top-level files in your site, and if a particular change is not safe, temporarily remove the optimizeCompilations attribute or set it to false.

http://i.msdn.microsoft.com/Global/Images/clear.gif Disadvantages of Dynamic Compilation

There are certain capabilities that dynamic compilation does not offer. Dynamic compilation can mean slower initial response time for users, because pages and code files must be compiled the first time they are requested. This may be an issue particularly on large sites that are updated frequently. Dynamic compilation does not offer a means to identify compile-time bugs before users access a site. Also, dynamic compilation does not provide the ability to create a compiled version of the site that can be deployed to a production server without source code. If any of these issues are concerns for your Web applications, you can precompile your Web site. For details information, see [ASP.NET Precompilation Overview](http://msdn.microsoft.com/en-us/library/bb398860.aspx).