**ASP.NET Precompilation 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 2010/.NET Framework 4.0](http://msdn.microsoft.com/en-us/library/bb398860(VS.100).aspx)

ASP.NET can precompile a Web site before it is made available to users. This provides many advantages, which include faster response time, error checking, source-code protection, and efficient deployment.

You can also compile a project by using the Web application project model. All code files (standalone, code-behind, and class files) in the project are compiled into a single assembly and stored in the Bin directory. Because compilation creates a single assembly, you can specify attributes, such as assembly name and version. You can also specify the location of the output assembly if you do not want it to be in the Bin directory. For more information, see [Compiling Web Application Projects](http://msdn.microsoft.com/en-us/library/aa983464.aspx).

This topic contains:

* [Scenarios](http://msdn.microsoft.com/en-us/library/bb398860.aspx#Scenarios)
* [Precompilation Features](http://msdn.microsoft.com/en-us/library/bb398860.aspx#Features)
* [Background](http://msdn.microsoft.com/en-us/library/bb398860.aspx#Background)
* [Code Examples](http://msdn.microsoft.com/en-us/library/bb398860.aspx#CodeExamples)

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

Precompiling an ASP.NET Web site provides the following advantages:

* Faster response time for users, because pages and code files do not have to be compiled the first time that they are requested. This is especially useful on large sites that are frequently updated.
* A means to identify compile-time bugs before users see a site.
* The ability to create a compiled version of the site that can be deployed to a production server without source code.

[Back to top](http://msdn.microsoft.com/en-us/library/bb398860.aspx#Introduction)

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

By default, ASP.NET Web pages and code files are compiled dynamically when users first request a resource such as a page from the Web site. After pages and code files have been compiled the first time, the compiled resources are cached. Therefore, subsequent requests to the same page are very efficient.

ASP.NET can also precompile a whole site before it is made available to users. ASP.NET offers the following options for precompiling a site:

* Precompiling a site in place. This option is useful for existing sites where you want to enhance performance and perform error checking.
* Precompiling a site for deployment. This option creates a special output that you can deploy to a production server.

Additionally, you can precompile a site to make it read-only or updatable. The following sections provide more details about each option.

**In-Place Precompilation**

You can enhance the performance of a Web site somewhat by precompiling it. This is particularly true in sites where there are frequent changes and additions among ASP.NET Web pages and code files. In a fluid Web site, the extra time required to dynamically compile new and changed pages can affect the users' perception of the site's quality.

Precompiling a site in place effectively performs the same compilation that occurs when users request pages from the site. Therefore, the primary performance improvement is that pages do not have to be compiled for the first request.

When you perform in-place precompilation, all ASP.NET file types are compiled. (HTML files, graphics, and other non-ASP.NET static files are left as is.) The precompilation process follows the same logic that ASP.NET uses for dynamic compilation, accounting for dependencies between files. During precompilation, the compiler creates assemblies for all executable output and puts them in a special folder under the %SystemRoot%\Microsoft.NET\Framework\*version*\Temporary ASP.NET Files folder. Thereafter, ASP.NET fulfills requests for pages from the assemblies in this folder.

If you precompile the site again, only new or changed files are compiled (or those with dependencies on new or changed files). Because of this compiler optimization, it is practical to compile the site after even minor updates.

**Precompilation for Deployment**

Another use for precompiling a site is to produce an executable version of the site that can be deployed to a production server. Precompiling for deployment creates output in the form of a layout. The layout contains assemblies, configuration information, information about the site's folders, and static files (such as HTML files and graphics).

After compiling the site, you can deploy the layout to a production server by using tools such as the Windows XCopy command, FTP, Windows installation, and so on. After the layout is deployed, it functions as the site, and ASP.NET fulfills requests for pages from the assemblies in the layout.

Precompiling a site for deployment provides a measure of protection for source code and other intellectual property. For more information about how the compiler works with files during compilation for deployment, see [File Handling During Site Compilation for Deployment](http://msdn.microsoft.com/en-us/library/bb398860.aspx#FileHandlingDuringASPNETPrecompilation) later in this topic.

You can precompile for deployment in two ways: precompiling for deployment only, or precompiling for deployment and update.

**Precompiling for Deployment Only**

When you precompile for deployment only, the compiler produces assemblies from almost all ASP.NET source files that are ordinarily compiled at run time. This includes program code in pages, .cs and .vb class files, other code files, and resource files. The compiler removes all source and markup from the output. In the resulting layout, compiled files are generated for each .aspx file (with the extension .compiled) that contains pointers to the appropriate assembly for that page.

To change the Web site, which includes the layout of pages, you must change the original files, recompile the site, and redeploy the layout. The only exception is the site configuration. You can make changes to the Web.config file on the production server without having to recompile the site.

This option provides the greatest degree of protection for ASP.NET Web pages and the best performance at startup.

**Precompiling for Deployment and Update**

When you precompile for deployment and update, the compiler produces assemblies from all source code (except page code in single-file pages) and from other files that ordinarily produce assemblies, such as resource files. The compiler converts .aspx files into single files that use the compiled code-behind model and copies them to the layout.

This option enables you to make limited changes to the ASP.NET Web pages in the site after you compile them. For example, you can change the arrangement of controls, colors, fonts, and other appearance aspects of pages. You can also add controls, as long as they do not require event handlers or other code.

When the site runs the first time, ASP.NET performs additional compilation in order to create output from the markup.

|  |
| --- |
| **NoteNote:** |
| A precompiled updatable site does not allow for multiple pages to reference the same **CodeFile** class. |

**Precompilation Decision Matrix**

Use the following table to help you decide which compilation model to use. Some of the options listed in the table are described later in this topic.

|  |  |
| --- | --- |
| **Goal** | **Compilation model** |
| Quickly develop applications without worrying about compiling the code. | Use default compilation (no precompilation). |
| Improve the response time for the first page request to the Web site. | Use in-place precompilation or one of the deployment compilation options. |
| Separate source code and user interface (UI) code. | Use precompilation with updatable UI. |
| Modify the UI code without changing the source code. | Use precompilation with updatable UI. |
| Remove all source code and UI code from the production server. | Use precompilation with non-updatable UI. |
| Update the application by replacing specific assemblies. | Use precompilation with fixed names. |
| Increase the security of the application by using strong-named assemblies. | Use precompilation with signed assemblies. |

**Performing Precompilation**

You can precompile a Web site using the Aspnet\_compiler.exe tool on the command line. For more information, see [How to: Precompile ASP.NET Web Sites for Deployment](http://msdn.microsoft.com/en-us/library/ms227976.aspx) and [ASP.NET Compilation Tool (Aspnet\_compiler.exe)](http://msdn.microsoft.com/en-us/library/ms229863.aspx). Visual Studio also includes commands to precompile a Web site from the IDE.

|  |
| --- |
| **NoteNote:** |
| Precompiling a Web site compiles only that site, not any child sites. If a Web site contains a subfolder that is marked as an application in IIS, the child application is not compiled when you precompile the parent folder. |

One coding restriction when you precompile a Web site applies to sites that you intend to compile with source protection enabled. It is possible for a base page class (a code-behind class) to reference the associated page class (.aspx file) and page class members by using a fully qualified class name. However, this kind of reference will not work when you precompile the site with source protection enabled. This is because the base page class from the code-behind file is not located in the same assembly as the page class derived from the .aspx page. For more information about precompilation with source protection enabled, see [How to: Sign Assemblies for Precompiled Web Sites](http://msdn.microsoft.com/en-us/library/ms228209.aspx).

**Default Compilation**

You do not have to manually compile an ASP.NET application. By default, ASP.NET compiles the Web application the first time a page in the application is requested by a Web browser. If you make a change to a file in the application, the next time a page is requested the ASP.NET runtime determines the dependencies of the changed file. It recompiles only the files that are affected by the change.

**Advantages**

The advantages of using default compilation include the following:

* Simple to use. The ASP.NET compiler does all the work for you.
* Best compilation model to select during development when the extra steps required for precompiling a Web site slow down the development process.

**Disadvantages**

The disadvantages of using default compilation include the following:

* Can cause significant delays when the Web site is first requested.
* Requires you to store source code files on the production server.
* Makes source code and UI code available to anyone with file-system access to the Web site directory on the server.

**When to Use Default Compilation**

Use default compilation in the following situations:

* When you are developing and testing a Web site.
* For Web sites that have primarily static information.
* For Web sites that are not frequently changed.
* When you are not concerned about storing source-code files on the production server.

**In-Place Compilation**

You can precompile a site that is already on a production server. This is called in-place compilation. If you make a change to a file in the application, you can recompile the affected file with the ASP.NET compilation tool. The affected files will also be recompiled the next time that a page is requested from the application.

For more information about this compilation model, see [How to: Precompile ASP.NET Web Sites](http://msdn.microsoft.com/en-us/library/ms227972.aspx).

**Advantages**

The advantages of using in-place compilation include the following:

* First-request response time from the Web site is reduced.
* No special deployment steps are needed; the application is compiled exactly as if a page had been requested from the site.

**Disadvantages**

The disadvantages of using in-place compilation include the following:

* All source code for the application must be stored on the production server.
* Makes source code and UI code available to anyone with access to the Web site directory.

**When to Use In-Place Compilation**

Use in-place compilation in the following situations:

* You make frequent changes to pages on the Web site.
* You are not concerned about storing source-code files on the production server.
* You want to improve the response time for first page requests.

**Precompiling with Updatable UI**

By using the **-u** switch of the ASP.NET compilation tool, you can compile source code (.cs, .vb files, and .resource files) to a DLL. You can leave the UI markup in the .aspx files available for updating. After you deploy the Web site to a production server, you can make changes to the .aspx code without recompiling the whole Web site.

For more information about this compilation method, see [How to: Precompile ASP.NET Web Sites for Deployment](http://msdn.microsoft.com/en-us/library/ms227976.aspx).

**Advantages**

The advantages of precompiling a Web site that has updatable UI include the following:

* First-request response time from the Web site is reduced.
* User interface designers can modify the appearance and behavior of a Web site without requiring the whole Web site to be recompiled.
* There is a measure of protection for intellectual property in the application's source code. It is protected from casual observation by anyone who has file-system access to the Web site directory.

**Disadvantages**

The disadvantages of precompiling a Web site that has updatable UI include the following:

* Requires a separate compilation step before deployment to production servers.
* Intellectual property in the application's UI (.aspx files) is available to anyone with access to the Web site directory.
* Multiple pages cannot reference the same **CodeFile** class, which is the associated code file for a page that use the code-behind model.

**When to Precompile with Updateable UI**

Precompile an application to have updatable UI in the following situations:

* UI designers are working separately from source-code developers.
* The program source code contains intellectual property that you want to protect from casual observation.
* You do not want to store program source code on the production server.

**Precompiling with Non-Updatable UI**

The ASP.NET compilation tool can compile all the source code for an application into DLLs that are deployed in the application's Bin directory. This includes UI files, such as .aspx and .ascx files.

For more information about this compilation method, see [How to: Precompile ASP.NET Web Sites for Deployment](http://msdn.microsoft.com/en-us/library/ms227976.aspx).

**Advantages**

The advantages of precompiling with non-updatable UI include the following:

* First-request response time from the Web site is reduced.
* There is a measure of protection for intellectual property in the application's source code and UI code. It is protected from casual observation by anyone who has access to the Web site directory.

**Disadvantages**

The disadvantages of precompiling with non-updatable UI include the following:

* Requires a separate compilation step before deployment to production servers.
* Even minor changes to the application's UI require the whole Web site to be recompiled.

**When to Precompile with Non-Updatable UI**

Precompile the Web site to have non-updatable UI in the following situations:

* UI code contains intellectual property that you want to protect from casual observation.
* You do not have to change the UI often.
* You want to have only compiled DLLs on the production server.

**Precompiling to Fixed-Name Assemblies**

The ASP.NET compilation tool uses random names for the assemblies that are generated during compilation. The name of the assembly changes every time that the application is recompiled.

Because the assembly names change, you must redeploy the whole application to service one assembly. However, by using the **-fixednames** switch with the ASP.NET compilation tool, you can create one assembly for each page in the application. The name of the assembly will not change on subsequent compilations. Therefore, you can create service releases of the application that replace only the changed assemblies.

Because the **-fixednames** switch creates an individual assembly for each page, you should limit the number of pages in the application.

For more information about this precompilation method, see [How to: Generate Fixed Names with the ASP.NET Compilation Tool](http://msdn.microsoft.com/en-us/library/ms228040.aspx).

**Advantages**

The advantages of precompiling to fixed-name assemblies include the following:

* The name of individual assemblies does not change from compilation to compilation. This enables you to replace specific assemblies without redeploying the whole application.
* Minor updates to the application can be more targeted.

**Disadvantages**

The disadvantages of precompiling to fixed-name assemblies include the following:

* One assembly is created for each page in the application. This can create many assemblies for sites that have many pages.

**When to Precompile to Fixed-Name Assemblies**

Precompile the Web site to fixed-name assemblies in the following situation:

* You must service Web applications without replacing the whole application.

**Precompiling to Signed Assemblies**

You can use the ASP.NET compilation tool to create strong-named assemblies that can be deployed to the server's Global Assembly Cache (GAC) or to the Bin directory of the application. A signed assembly makes it more difficult for malicious users to replace the application's assemblies with malicious code.

For more information about this compilation method, see [How to: Sign Assemblies for Precompiled Web Sites](http://msdn.microsoft.com/en-us/library/ms228209.aspx).

**Advantages**

The advantages of precompiling to signed assemblies include the following:

* Signed assemblies increase the security of the application by making it more difficult for the assemblies to be replaced by malicious code.

**Disadvantages**

The disadvantages of precompiling to signed assemblies include the following:

* Key management in shared development environments can be complex.
* Assemblies must have the [AllowPartiallyTrustedCallersAttribute](http://msdn.microsoft.com/en-us/library/system.security.allowpartiallytrustedcallersattribute.aspx) attribute to be called by the ASP.NET runtime.

**When to Precompile to Signed Assemblies**

Precompile the Web site to signed assemblies in the following situations:

* Users have access to the application directory or GAC, and they can replace the application's assemblies.
* You want to limit the ability of third parties to replace the assemblies generated by the application code.

**Writing Precompilation Output**

When the precompilation process is finished, the resulting output is written to a folder that you specify. You can write the output to any folder that is available to you in the file system, by using File Transfer Protocol (FTP), or by using HTTP. You must have appropriate permissions to be able to write to the target site.

|  |
| --- |
| **NoteNote:** |
| The publishing process deploys only the files from the Web site's folders and subfolders. It does not deploy the Machine.config file. Therefore, the configuration of the target Web server might be different than it is on your computer, which might affect the behavior of the application. |

You can specify a target folder on a staging server or production server, or you can write the output to a folder on the local computer. If you specify a folder on a production server, you can precompile and deploy in a single step. If you write the output to a folder that is not part of a Web site, you can copy the output to the server in a separate step.

|  |
| --- |
| **NoteNote:** |
| If you open a precompiled Web site using Visual Studio, you will not be able to build the Web site. Build options will be turned off. To change the site, it is recommended that you edit the files in the original Web site, precompile the site, and then publish it again. |

The output of the compilation process includes the compiled assemblies for any code or pages. If you select the option to enable the precompiled site to be updated, any code-behind classes for .aspx, .asmx, and .ashx files are compiled into assemblies. However, the .aspx, .asmx, and .ashx files themselves are copied as-is to the target folder so that you can change their layout after you deploy the site. For precompiled sites that can be updated, the code in single-file pages is not compiled into an assembly. Instead, it is deployed as source code.

Static files are not compiled. Instead, they are copied as-is to the output folder. Static files include graphics, .htm or .html files, text files, and so on. For more information, see [File Handling During Site Compilation for Deployment](http://msdn.microsoft.com/en-us/library/bb398860.aspx#FileHandlingDuringASPNETPrecompilation) later in this topic.

If an error occurs during precompilation, it is reported to you in the **Output** window and in the **Error List** window. Errors during precompilation will prevent the site from being compiled and published.

**File Handling During ASP.NET Precompilation**

When you precompile a site for deployment, ASP.NET creates a layout, which is a structure that contains the compiler output. This section describes how files are handled during precompilation and describes the layout structure and contents.

You can precompile both source code (any file that produces an assembly, which includes program code and resources) and markup (.aspx files), or only source code.

**Compiled Files**

The precompilation process performs actions on various types of files in an ASP.NET Web application. Files are treated differently depending on whether the application is being precompiled for deployment only or if it is being precompiled for deployment and update.

|  |
| --- |
| **NoteNote:** |
| Precompiling a site only for deployment or for deployment and update does not preserve file access control lists (ACLs) on target files and subdirectories. For example, if you have previously precompiled a site and deployed it to a target location, changed a file's ACL, and then precompiled and deployed the site again, the ACL change will be lost. |

The following table describes different file types and the actions taken on them if the application is being precompiled for deployment only.

|  |  |  |
| --- | --- | --- |
| **File types** | **Precompilation action** | **Output location** |
| .aspx, ascx, .master | Generates assemblies and a .compiled file that points to the assembly. The original file is left in place as a placeholder for fulfilling requests. | Assemblies and .compiled files are written to the Bin folder. Pages (.aspx files, with content stripped) are left in their original locations. |
| .asmx, .ashx | Generates assemblies. The original file is left in place as a placeholder for fulfilling requests. | Bin folder |
| Files in the App\_Code folder | Generates one or more assemblies (depending on Web.config settings).  Note**Note:**  Static content in the App\_Code folder is not copied to the target folder. | Bin folder |
| .cs or .vb files not in the App\_Code folder | Compiles with the page or resource that depends on it. | Bin folder |
| Existing .dll files in Bin folder. | Copies files as is. | Bin folder |
| Resources (.resx) files | For .resx files found in the App\_LocalResources or App\_GlobalResources folders, generates an assembly or assemblies and a culture structure. | Bin folder |
| Files in the App\_Themes folder and subfolders | Generates assemblies in target and generates .compiled files that point to the assemblies. | Bin |
| Static files (.htm, .html, .js, graphics files, and so on) | Copies files as is. | Same structure as in source. |
| Browser definition files | Copies files as is.  Note**Note:**  Browser information is inherited from machine-level configuration files, and may therefore behave differently on the target server. | App\_Browsers |
| Dependent projects | Generates the output of the dependent project into an assembly. | Bin folder |
| Web.config files | Copies files as is. | Same structure as in source. |
| Global.asax files | Generates an assembly. | Bin folder |

The following table describes the different file types and the actions taken on them if the application is being precompiled for deployment and update.

|  |  |  |
| --- | --- | --- |
| **File types** | **Precompilation action** | **Output location** |
| .aspx, ascx, .master | Generates assemblies for files that have code-behind class files. Generates a .compiled file that points to the assembly. Single-file versions of these files are copied as is to the target. | Assemblies and .compiled files are written to the Bin folder. |
| .asmx, .ashx | Copies files as is without compilation. | Same structure as in source. |
| Files in the App\_Code folder | Generates one or more assemblies (depending on Web.config settings).  Note**Note:**  Static content in the App\_Code folder is not copied to the target folder. | Bin folder |
| .cs or .vb files not in App\_Code folder | Compiles with the page or resource that depends on it. | Bin folder |
| Existing .dll files in Bin folder. | Copies files as is. | Bin folder |
| Resources (.resx) files | For .resx files in the App\_GlobalResources folders, generates an assembly or assemblies and a culture structure.  For .resx files in the App\_LocalResources folders, copies files as is to the App\_LocalResources folder of the output location. | Assemblies are put in the Bin folder |
| Files in the App\_Themes folder and subfolders | Copies files as is. | Same structure as in source. |
| Static files (.htm, .html, .js, graphics files, and so on) | Copies files as is. | Same structure as in source. |
| Browser definition files | Copies files as is.  Note**Note:**  Browser information is inherited from machine-level configuration files, and may therefore behave differently on the target server. | App\_Browsers |
| Dependent projects | Generates the output of the dependent project into an assembly. | Bin folder |
| Web.config files | Copies files as is. | Same structure as in source. |
| Global.asax files | Generates an assembly. | Bin folder |

**.compiled Files**

For executable files in an ASP.NET Web application, the compiler adds the .compiled file name extension to the name of assemblies and files. The assembly name is generated by the compiler. The .compiled file does not contain executable code. Instead, it contains only the information that ASP.NET must have to find the appropriate assembly.

After the precompiled application is deployed, ASP.NET uses the assemblies in the Bin folder to process requests. The precompilation output includes .aspx or .asmx files as placeholders for pages. The placeholder files contain no code. They exist only to invoke ASP.NET for a specific page request. The placeholder files also provide a way that file permissions can be set to restrict access to the pages.

**Updating Precompiled Web Sites**

After you deploy a precompiled Web site, you can make limited changes to the files in the site. The following table describes the effect of different types of changes.

|  |  |  |
| --- | --- | --- |
| **File type** | **Changes permitted (deployment only)** | **Changes permitted (deployment and update)** |
| Static files (.htm, .html, .js, graphics files, and so on) | Static files can be changed, removed, or added. If an ASP.NET Web page refers to pages or page elements that are changed or removed, errors might occur. | Static files can be changed, removed, or added. If an ASP.NET Web page refers to pages or page elements that are changed or removed, errors might occur. |
| .aspx file | No changes are permitted to existing pages. No new .aspx files can be added. | You can change the layout of .aspx files and add elements that do not require code, such as HTML elements and ASP.NET server controls that have no event handlers. You can also add new .aspx files, which are compiled on first request. |
| .skin files | Changes and new .skin files are ignored. | Changes and new .skin files are permitted. |
| Web.config files | Changes are permitted that affect the compilation of .aspx files. Compilation options for debugging or batching are ignored.  No changes to profile properties or provider elements are permitted. | Changes are permitted if they do not affect site or page compilation. This includes compiler settings, trust levels, and globalization. Changes that affect compilation or that change behavior in compiled pages are ignored or might generate errors. Other changes are permitted. |
| Browser definitions | Changes and new files are permitted. | Changes and new files are permitted. |
| Assemblies compiled from resource (.resx) files | New resource assembly files can be added for both global and local resources. | New resource assembly files can be added for both global and local resources. |