**ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0**

**.NET Framework 4**

[Other Versions](javascript:;)

http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ms178473(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms178473(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ms178473(d=printer,v=vs.80).aspx)

This topic outlines the life cycle of ASP.NET applications, listing important life-cycle events and describing how code that you write can fit into the application life cycle. The information in this topic applies to IIS 5.0 and IIS 6.0. For information about the ASP.NET application life cycle in IIS 7.0, see [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

Within ASP.NET, several processing steps must occur for an ASP.NET application to be initialized and process requests. Additionally, ASP.NET is only one piece of the Web server architecture that services requests made by browsers. It is important for you to understand the application life cycle so that you can write code at the appropriate life cycle stage for the effect you intend.

[Application Life Cycle in General](javascript:void(0))

The following table describes the stages of the ASP.NET application life cycle.

|  |  |
| --- | --- |
| **Stage** | **Description** |
| User requests an application resource from the Web server. | The life cycle of an ASP.NET application starts with a request sent by a browser to the Web server (for ASP.NET applications, typically IIS). ASP.NET is an ISAPI extension under the Web server. When a Web server receives a request, it examines the file-name extension of the requested file, determines which ISAPI extension should handle the request, and then passes the request to the appropriate ISAPI extension. ASP.NET handles file name extensions that have been mapped to it, such as .aspx, .ascx, .ashx, and .asmx.  Note**Note**  If a file name extension has not been mapped to ASP.NET, ASP.NET will not receive the request. This is important to understand for applications that use ASP.NET authentication. For example, because .htm files are typically not mapped to ASP.NET, ASP.NET will not perform authentication or authorization checks on requests for .htm files. Therefore, even if a file contains only static content, if you want ASP.NET to check authentication, create the file using a file name extension mapped to ASP.NET, such as .aspx.  Note**Note**  If you create a custom handler to service a particular file name extension, you must map the extension to ASP.NET in IIS and also register the handler in your application's Web.config file. For more information, see [HTTP Handlers and HTTP Modules Overview](http://msdn.microsoft.com/en-us/library/bb398986.aspx). |
| ASP.NET receives the first request for the application. | When ASP.NET receives the first request for any resource in an application, a class named [ApplicationManager](http://msdn.microsoft.com/en-us/library/system.web.hosting.applicationmanager.aspx) creates an application domain. Application domains provide isolation between applications for global variables and allow each application to be unloaded separately. Within an application domain, an instance of the class named [HostingEnvironment](http://msdn.microsoft.com/en-us/library/system.web.hosting.hostingenvironment.aspx) is created, which provides access to information about the application such as the name of the folder where the application is stored.  The following diagram illustrates this relationship:  Application Topgraphy Overview Graphic  ASP.NET also compiles the top-level items in the application if required, including application code in the App\_Code folder. For more information, see "Compilation Life Cycle" later in this topic. |
| ASP.NET core objects are created for each request. | After the application domain has been created and the [HostingEnvironment](http://msdn.microsoft.com/en-us/library/system.web.hosting.hostingenvironment.aspx) object instantiated, ASP.NET creates and initializes core objects such as [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx), [HttpRequest](http://msdn.microsoft.com/en-us/library/system.web.httprequest.aspx), and [HttpResponse](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.aspx). The [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) class contains objects that are specific to the current application request, such as the [HttpRequest](http://msdn.microsoft.com/en-us/library/system.web.httprequest.aspx) and [HttpResponse](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.aspx) objects. The [HttpRequest](http://msdn.microsoft.com/en-us/library/system.web.httprequest.aspx) object contains information about the current request, including cookies and browser information. The [HttpResponse](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.aspx) object contains the response that is sent to the client, including all rendered output and cookies. |
| An [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) object is assigned to the request | After all core application objects have been initialized, the application is started by creating an instance of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class. If the application has a Global.asax file, ASP.NET instead creates an instance of the Global.asax class that is derived from the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class and uses the derived class to represent the application.  Note**Note**  The first time an ASP.NET page or process is requested in an application, a new instance of [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) is created. However, to maximize performance, [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) instances might be reused for multiple requests.  When an instance of [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) is created, any configured modules are also created. For instance, if the application is configured to do so, ASP.NET creates a [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) module. After all configured modules are created, the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class's [Init](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.init.aspx) method is called.  The following diagram illustrates this relationship:  Application Environment Graphic |
| The request is processed by the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) pipeline. | The following events are executed by the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class while the request is processed. The events are of particular interest to developers who want to extend the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class.   1. Validate the request, which examines the information sent by the browser and determines whether it contains potentially malicious markup. For more information, see [ValidateRequest](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.validaterequest.aspx) and [Script Exploits Overview](http://msdn.microsoft.com/en-us/library/w1sw53ds.aspx). 2. Perform URL mapping, if any URLs have been configured in the [UrlMappingsSection](http://msdn.microsoft.com/en-us/library/system.web.configuration.urlmappingssection.aspx) section of the Web.config file. 3. Raise the [BeginRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.beginrequest.aspx) event. 4. Raise the [AuthenticateRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.authenticaterequest.aspx) event. 5. Raise the [PostAuthenticateRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postauthenticaterequest.aspx) event. 6. Raise the [AuthorizeRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.authorizerequest.aspx) event. 7. Raise the [PostAuthorizeRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postauthorizerequest.aspx) event. 8. Raise the [ResolveRequestCache](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.resolverequestcache.aspx) event. 9. Raise the [PostResolveRequestCache](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postresolverequestcache.aspx) event. 10. Based on the file name extension of the requested resource (mapped in the application's configuration file), select a class that implements [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) to process the request. If the request is for an object (page) derived from the [Page](http://msdn.microsoft.com/en-us/library/system.web.ui.page.aspx) class and the page needs to be compiled, ASP.NET compiles the page before creating an instance of it. 11. Raise the [PostMapRequestHandler](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postmaprequesthandler.aspx) event. 12. Raise the [AcquireRequestState](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.acquirerequeststate.aspx) event. 13. Raise the [PostAcquireRequestState](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postacquirerequeststate.aspx) event. 14. Raise the [PreRequestHandlerExecute](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.prerequesthandlerexecute.aspx) event. 15. Call the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method (or the asynchronous version IHttpAsyncHandler.BeginProcessRequest) of the appropriate [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) class for the request. For example, if the request is for a page, the current page instance handles the request. 16. Raise the [PostRequestHandlerExecute](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postrequesthandlerexecute.aspx) event. 17. Raise the [ReleaseRequestState](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.releaserequeststate.aspx) event. 18. Raise the [PostReleaseRequestState](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postreleaserequeststate.aspx) event. 19. Perform response filtering if the [Filter](http://msdn.microsoft.com/en-us/library/system.web.httpresponse.filter.aspx) property is defined. 20. Raise the [UpdateRequestCache](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.updaterequestcache.aspx) event. 21. Raise the [PostUpdateRequestCache](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.postupdaterequestcache.aspx) event. 22. Raise the [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest.aspx) event. 23. Raise the [PreSendRequestHeaders](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.presendrequestheaders.aspx) event. 24. Raise the [PreSendRequestContent](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.presendrequestcontent.aspx) event. |

[Life Cycle Events and the Global.asax file](javascript:void(0))

During the application life cycle, the application raises events that you can handle and calls particular methods that you can override. To handle application events or methods, you can create a file named Global.asax in the root directory of your application.

If you create a Global.asax file, ASP.NET compiles it into a class derived from the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class, and then uses the derived class to represent the application.

An instance of [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) processes only one request at a time. This simplifies application event handling because you do not need to lock non-static members in the application class when you access them. This also allows you to store request-specific data in non-static members of the application class. For example, you can define a property in the Global.asax file and assign it a request-specific value.

ASP.NET automatically binds application events to handlers in the Global.asax file using the naming convention Application\_event, such as Application\_BeginRequest. This is similar to the way that ASP.NET page methods are automatically bound to events, such as the page's Page\_Load event. For details, see [ASP.NET Page Life Cycle Overview](http://msdn.microsoft.com/en-us/library/ms178472.aspx).

The Application\_Start and Application\_End methods are special methods that do not represent [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) events. ASP.NET calls them once for the lifetime of the application domain, not for each [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) instance.

The following table lists some of the events and methods that are used during the application life cycle. There are many more events than those listed, but they are not commonly used.

|  |  |
| --- | --- |
| **Event or method** | **Description** |
| Application\_Start | Called when the first resource (such as a page) in an ASP.NET application is requested. The Application\_Start method is called only one time during the life cycle of an application. You can use this method to perform startup tasks such as loading data into the cache and initializing static values.  You should set only static data during application start. Do not set any instance data because it will be available only to the first instance of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class that is created. |
| Application\_event | Raised at the appropriate time in the application life cycle, as listed in the application life cycle table earlier in this topic.  Application\_Error can be raised at any phase in the application life cycle.  Application\_EndRequest is the only event that is guaranteed to be raised in every request, because a request can be short-circuited. For example, if two modules handle the Application\_BeginRequest event and the first one throws an exception, the Application\_BeginRequest event will not be called for the second module. However, the Application\_EndRequest method is always called to allow the application to clean up resources. |
| [Init](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.init.aspx) | Called once for every instance of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class after all modules have been created. |
| [Dispose](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.dispose.aspx) | Called before the application instance is destroyed. You can use this method to manually release any unmanaged resources. For more information, see [Cleaning Up Unmanaged Resources](http://msdn.microsoft.com/en-us/library/498928w2.aspx). |
| Application\_End | Called once per lifetime of the application before the application is unloaded. |

[Compilation Life Cycle](javascript:void(0))

When the first request is made to an application, ASP.NET compiles application items 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. The following table describes the order in which ASP.NET top-level items are compiled.

|  |  |
| --- | --- |
| **Item** | **Description** |
| App\_GlobalResources | The application's global resources are compiled and a resource assembly is built. Any assemblies in the application's Bin folder are linked to the resource assembly. |
| App\_WebResources | Proxy types for Web services are created and compiled. The resulting Web references assembly is linked to the resource assembly if it exists. |
| Profile properties defined in the Web.config file | If profile properties are defined in the application's Web.config file, an assembly is generated that contains a profile object. |
| App\_Code | Source code files are built and one or more assemblies are created. All code assemblies and the profile assembly are linked to the resources and Web references assemblies if any. |
| Global.asax | The application object is compiled and linked to all of the previously generated assemblies. |

After the application's top level items have been compiled, ASP.NET compiles folders, pages, and other items as needed. The following table describes the order in which ASP.NET folders and items are compiled.

|  |  |
| --- | --- |
| **Item** | **Description** |
| App\_LocalResources | If the folder containing the requested item contains an App\_LocalResources folder, the contents of the local resources folder are compiled and linked to the global resources assembly. |
| Individual Web pages (.aspx files), user controls (.ascx files), HTTP handlers (.ashx files), and HTTP modules (.asmx files) | Compiled as needed and linked to the local resources assembly and the top-level assemblies. |
| Themes, master pages, other source files | Skin files for individual themes, master pages, and other source code files referenced by pages are compiled when the referencing page is compiled. |

Compiled assemblies are cached on the server and reused on subsequent requests and are preserved across application restarts as long as the source code is unchanged.

Because the application is compiled on the first request, the initial request to an application can take significantly longer than subsequent requests. You can precompile your application to reduce the time required for the first request. For more information, see [How to: Precompile ASP.NET Web Site Projects](http://msdn.microsoft.com/en-us/library/ms227972.aspx).

**Application Restarts**

Modifying the source code of your Web application will cause ASP.NET to recompile source files into assemblies. When you modify the top-level items in your application, all other assemblies in the application that reference the top-level assemblies are recompiled as well.

In addition, modifying, adding, or deleting certain types of files within the application's known folders will cause the application to restart. The following actions will cause an application restart:

* Adding, modifying, or deleting assemblies from the application's Bin folder.
* Adding, modifying, or deleting localization resources from the App\_GlobalResources or App\_LocalResources folders.
* Adding, modifying, or deleting the application's Global.asax file.
* Adding, modifying, or deleting source code files in the App\_Code directory.
* Adding, modifying, or deleting Profile configuration.
* Adding, modifying, or deleting Web service references in the App\_WebReferences directory.
* Adding, modifying, or deleting the application's Web.config file.

When an application restart is required, ASP.NET will serve all pending requests from the existing application domain and the old assemblies before restarting the application domain and loading the new assemblies.

[HTTP Modules](javascript:void(0))

The ASP.NET application life cycle is extensible through [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx) classes. ASP.NET includes several classes that implement [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx), such as the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) class. You can also create your own classes that implement [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx).

If you add modules to your application, the modules themselves can raise events. The application can subscribe to in these events in the Global.asax file by using the convention modulename\_eventname. For example, to handle the [Authenticate](http://msdn.microsoft.com/en-us/library/system.web.security.formsauthenticationmodule.authenticate.aspx) event raised by a [FormsAuthenticationModule](http://msdn.microsoft.com/en-us/library/system.web.security.formsauthenticationmodule.aspx) object, you can create a handler named FormsAuthentication\_Authenticate.

The [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) class is enabled by default in ASP.NET. All session events are automatically wired up as Session\_event, such as Session\_Start. The [Start](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.start.aspx) event is raised each time a new session is created. For more information, see [ASP.NET Session State Overview](http://msdn.microsoft.com/en-us/library/ms178581.aspx).