ASP.NET Application Life Cycle Overview for IIS 7.0

This topic describes the application life cycle for ASP.NET applications that are running in IIS 7.0 in Integrated mode and with the .NET Framework 3.0 or later. IIS 7.0 also supports Classic mode, which behaves like ASP.NET running in IIS 6.0. For more information, see ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0.

The IIS 7.0 integrated pipeline is a unified request processing pipeline that supports both native-code and managed-code modules. Managed-code modules that implement the IHttpModule interface have access to all events in the request pipeline. For example, a managed-code module can be used for ASP.NET forms authentication for both ASP.NET Web pages (.aspx files) and HTML pages (.htm or .html files). This is true even though HTML pages are treated as static resources by IIS and ASP.NET. For more information about IIS 7.0 Integrated mode, see ASP.NET Integration with IIS7.

This topic contains the following sections:

Architectural Overview

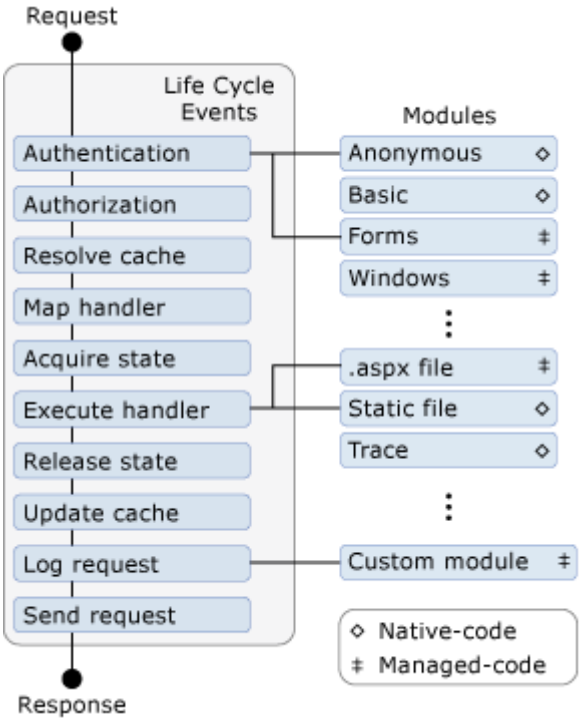
A request in IIS 7.0 Integrated mode passes through stages that are like the stages of requests for ASP.NET resources in IIS 6.0. However, in IIS 7.0, these stages include several additional application events, such as the MapRequestHandler, LogRequest, and PostLogRequest events.

The main difference in processing stages between IIS 7.0 and IIS 6.0 is in how ASP.NET is integrated with the IIS server. In IIS 6.0, there are two request processing pipelines. One pipeline is for native-code ISAPI filters and extension components. The other pipeline is for managed-code application components such as ASP.NET. In IIS 7.0, the ASP.NET runtime is integrated with the Web server so that there is one unified request processing pipeline for all requests. For ASP.NET developers, the benefits of the integrated pipeline are as follows:

- The integrated pipeline raises all the events that are exposed by the HttpApplication object, which enables existing ASP.NET HTTP modules to work in IIS 7.0 Integrated mode.
- Both native-code and managed-code modules can be configured at the Web server, Web site, or Web application level. This includes the built-in ASP.NET managed-code modules for session state, forms authentication, profiles, and role management. Furthermore, managed-code modules can be enabled or disabled for all requests, regardless of whether the request is for an ASP.NET resource like an .aspx file.
- Managed-code modules can be invoked at any stage in the pipeline. This includes before any server processing occurs for the request, after all server processing has occurred, or anywhere in between.
- You can register and enable or disable modules through an application's Web.config file.

The following illustration shows the configuration of an application's request pipeline. The example includes the following:

- The **Anonymous** native-code module and the **Forms** managed-code module (which corresponds to FormsAuthenticationModule). These modules are configured, and they are invoked during the **Authentication** stage of the request.
- The **Basic** native-code module and the **Windows** managed-code module (which corresponds to WindowsAuthenticationModule). They are shown, but they are not configured for the application.
- The **Execute handler** stage, where the handler (a module scoped to a URL) is invoked to construct the response. For .aspx files, the PageHandlerFactory handler is used to respond to the request. For static files, the native-code **StaticFileModule** module responds to the request.
- The **Trace** native-code module. This is shown, but it is not configured for the application.
- The **Custom module** managed-code class. It is invoked during the **Log request** stage.



For information about known compatibility issues with ASP.NET applications that are being migrated from earlier versions of IIS to IIS 7.0, see the "Known Differences Between Integrated Mode and Classic Mode" section of Upgrading ASP.NET Applications to IIS 7.0: Differences between IIS 7.0 Integrated Mode and Classic mode.

Life Cycle Stages

The following table lists the stages of the ASP.NET application life cycle with Integrated mode in IIS 7.0.

Stage	Description
A request is made for an application resource.	The life cycle of an ASP.NET application starts with a request sent by a browser to the Web server.

	In Classic mode in IIS 7.0 and in IIS 6.0, the ASP.NET request pipeline is separate from the Web server pipeline. Modules apply only to requests that are routed to the ASP.NET ISAPI extension. If the file-name extension of the requested resource type is not explicitly mapped to ASP.NET, ASP.NET functionality is not invoked for the request because the request is not processed by the ASP.NET runtime. In integrated mode in IIS 7.0, a unified pipeline handles all requests. When the integrated pipeline receives a request, the request passes through stages that are common to all requests. These stages are represented by the RequestNotification enumeration. All requests can be configured to take advantage of ASP.NET functionality, because that functionality is encapsulated in managed-code modules that have access to the request pipeline. For example, even though the .htm file-name extension is not explicitly mapped to ASP.NET, a request for an HTML page still invokes ASP.NET modules. This enables you to take advantage of ASP.NET authentication and authorization for all resources.
The unified pipeline receives the first request for the application.	When the unified pipeline receives the first request for any resource in an application, an instance of the ApplicationManager class is created, which is the application domain that the request is processed in. Application domains provide isolation between applications for global variables and enable each application to be unloaded separately. In the application domain, an instance of the HostingEnvironment class is created, which provides access to information about the application, such as the name of the folder where the application is stored. During the first request, top-level items in the application are compiled if required, which includes application code in the App_Code folder. You can include custom modules and handlers in the App_Code folder as described in Managed-code Modules in IIS 7.0 later in this topic.
Response objects are created for each request.	After the application domain has been created and the HostingEnvironment object has been instantiated, application objects such as HttpContext, HttpRequest, and HttpResponse are created and initialized. The HttpContext class contains objects that are specific to the current application request, such as the HttpRequest and HttpResponse objects. The HttpRequest object contains information about the current request, which includes cookies and browser information. The HttpResponse object contains the response that is sent to the client, which includes all the rendered output and cookies. The following are some key differences between IIS 6.0 and IIS 7.0 running in Integrated mode and with the .NET Framework 3.0 or later: • The SubStatusCode property of the HttpResponse object is available for setting codes that are useful for failed-request tracing. For more information, see Troubleshooting Failed Requests Using Failed Request Tracing in IIS 7.0. • The Headers property of the HttpResponse object provides access to response headers for the response. • Two properties of the HttpContext object, IsPostNotification and CurrentNotification, are used when one event handler handles several HttpApplication events. • The Headers and ServerVariables property of the HttpRequest object are write-enabled.
An HttpApplication object is assigned to the request	After all application objects have been initialized, the application is started by creating an instance of the HttpApplication 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 class. It then uses the derived class to represent the application. **Note**

	The first time that an ASP.NET page or process is requested in an application, a new instance of the HttpApplication class is created. However, to maximize performance, HttpApplication instances might be reused for multiple requests. Which ASP.NET modules are loaded (such as the SessionStateModule) depends on the managed-code modules that the application inherits from a parent application. It also depends on which modules are configured in the configuration section of the application's Web.config file. Modules are added or removed in the application's Web.config **modules** element in the **system.webServer** section. For more information, see How to: Configure the <system.webServer> Section for IIS 7.0.
The request is processed by the HttpApplication pipeline.	The following tasks are performed by the HttpApplication class while the request is being processed. The events are useful for page developers who want to run code when key request pipeline events are raised. They are also useful if you are developing a custom module and you want the module to be invoked for all requests to the pipeline. Custom modules implement the IHttpModule interface. In Integrated mode in IIS 7.0, you must register event handlers in a module's Init method. 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 and Script Exploits Overview. 2. Perform URL mapping, if any URLs have been configured in the UrlMappingsSection section of the Web.config file. 3. Raise the BeginRequest event. 4. Raise the AuthenticateRequest event. 5. Raise the PostAuthenticateRequest event. 6. Raise the AuthorizeRequest event. 7. Raise the PostAuthorizeRequest event. 8. Raise the ResolveRequestCache event. 9. Raise the PostResolveRequestCache event. 10. Raise the MapRequestHandler event. An appropriate handler is selected based on the file-name extension of the requested resource. The handler can be a native-code module such as the IIS 7.0 **StaticFileModule** or a managed-code module such as the PageHandlerFactory class (which handles .aspx files). 11. Raise the PostMapRequestHandler event. 12. Raise the AcquireRequestState event. 13. Raise the PostAcquireRequestState event. 14. Raise the PreRequestHandlerExecute event. 15. Call the ProcessRequest method (or the asynchronous version IHttpAsyncHandler.BeginProcessRequest) of the appropriate IHttpHandler class for the request. For example, if the request is for a page, the current page instance handles the request. 16. Raise the PostRequestHandlerExecute event. 17. Raise the ReleaseRequestState event. 18. Raise the PostReleaseRequestState event. 19. Perform response filtering if the Filter property is defined. 20. Raise the UpdateRequestCache event. 21. Raise the PostUpdateRequestCache event. 22. Raise the LogRequest event. 23. Raise the PostLogRequest event. 24. Raise the EndRequest event. 25. Raise the PreSendRequestHeaders event. 26. Raise the PreSendRequestContent event.

Note

The MapRequestHandler, LogRequest, and PostLogRequest events are supported only if the application is running in Integrated mode in IIS 7.0 and with the .NET Framework 3.0 or later.

Using the Global.asax File

The Global.asax file is used in Integrated mode in IIS 7.0 much as it is used in ASP.NET in IIS 6.0. For more information, see the "Life Cycle Events and Global.asax File" section in ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0.

One difference is that you can add handlers for the MapRequestHandler, LogRequest, and PostLogRequest events. These events are supported for applications that run in Integrated mode in IIS 7.0 and with the .NET Framework 3.0 or later.

You can provide application event handlers in the Global.asax file to add code that executes for all requests that are handled by ASP.NET, such as requests for .aspx and .axd pages. However, handler code in the Global.asax file is not called for requests for non-ASP.NET resources, such as static files. To run managed code that runs for all resources, create a custom module that implements the IHttpModule interface. The custom module will run for all requests to resources in the application, even if the resource handler is not an ASP.NET handler.

Managed-code Modules in IIS 7.0

The ASP.NET managed-code modules that can be configured and loaded in IIS 7.0 include the following:

- FormsAuthenticationModule
- ProfileModule
- RoleManagerModule
- SessionStateModule

To configure IIS 7.0 managed-code modules you can use one of the following methods:

- Use IIS Manager. For more information, see How to: Open IIS Manager.
- Use the IIS 7.0 command-line tool (Appcmd.exe). For more information, see IIS 7.0 Command-Line Tool.
- Edit the IIS 7.0 XML-based configuration store. For more information, see IIS 7.0: IIS 7.0 Configuration Store.

When an ASP.NET managed-code module such as the FormsAuthenticationModule module is configured to load in IIS 7.0, it has access to all events in the request pipeline. This means that all requests pass through the managed-code module. For the FormsAuthenticationModule class, it means that static content can be protected by using forms authentication, even though the content is not handled by an ASP.NET handler.

Developing Custom Managed-code Modules

The ASP.NET application life cycle can be extended with modules that implement the IHttpModule interface. Modules that implement the IHttpModule interface are managed-code modules. The integrated pipeline of ASP.NET and IIS 7.0 is also extensible through native-code modules, which are not discussed in this topic. For more information about native-code modules and about how to configure modules generally, see IIS Module Overview.

You can define a managed-code module as a class file in the application's App_Code folder. You can also create the module as a class library project, compile it, and add it to application's Bin folder. After you have created the custom module, you must register it with IIS 7.0. You can use one of the methods described for managing IIS 7.0 managed-code modules. For example, you can edit an application's Web.config file to register the managed-code module for just that application. For an example of registering a module, see Walkthrough: Creating and Registering a Custom HTTP Module.

If a module is defined an application's App_Code or Bin folder and it is registered in the application's Web.config file, the module is invoked only for that application. To register the module in the application's Web.config file, you work with the **modules** element in the **system.webServer** section. For more information, see How to: Configure the <system.webServer> Section for IIS 7.0. Changes made by using IIS Manager or the Appcmd.exe tool will make changes to the application's Web.config file.

Managed-code modules can also be registered in the **modules** element of the IIS 7.0 configuration store (the ApplicationHost.config file). Modules registered in the ApplicationHost.config file have global scope because they are registered for all Web applications hosted by IIS 7.0. Similarly, native-code modules that are defined in the **globalModules** element of the ApplicationHost.config file have global scope. If a global module is not needed for a Web application, you can disable it.

Example

The following example shows a custom module that handles the LogRequest and PostLogRequest events. Event handlers are registered in the Init method of the module.

C#

```
using System;
using System.Data;
using System.Web;
using System.Web.Security;
using System.Web.UI;

// Module that demonstrates one event handler for several events.
namespace Samples
{
    public class ModuleExample : IHttpModule
    {
        public ModuleExample()
        {
            // Constructor
        }
        public void Init(HttpApplication app)
        {
            app.LogRequest += new EventHandler(App_Handler);
            app.PostLogRequest += new EventHandler(App_Handler);
        }
        public void Dispose()
        {
        }
```

```
        // One handler for both the LogRequest and the PostLogRequest events.
        public void App_Handler(object source, EventArgs e)
        {
            HttpApplication app = (HttpApplication)source;
            HttpContext context = app.Context;

            if (context.CurrentNotification == RequestNotification.LogRequest)
            {
                if (!context.IsPostNotification)
                {
                    // Put code here that is invoked when the LogRequest event is
raised.
                }
                else
                {
                    // PostLogRequest
                    // Put code here that runs after the LogRequest event
completes.
                }
            }

        }
    }
}
```

The following example shows how to register the module in the application's Web.config file. Add the **system.webServer** configuration section inside the configuration section.

```
<system.webServer>
  <modules>
    <add name="ModuleExample" type="Samples.ModuleExample"/>
  </modules>
</system.webServer>
```

For an additional example that shows how to create and register a custom module, see Walkthrough: Creating and Registering a Custom HTTP Module.

See Also

Concepts
ASP.NET Page Life Cycle Overview
ASP.NET Overview
ASP.NET Compilation Overview
Other Resources
ASP.NET and IIS Configuration