This page is specific to

Microsoft Visual Studio 2010/.NET Framework 4

**HTTP Handlers and HTTP Modules Overview**

An ASP.NET HTTP handler is the process (frequently referred to as the "endpoint") that runs in response to a request made to an ASP.NET Web application. The most common handler is an ASP.NET page handler that processes .aspx files. When users request an .aspx file, the request is processed by the page through the page handler. You can create your own HTTP handlers that render custom output to the browser.

An HTTP module is an assembly that is called on every request that is made to your application. HTTP modules are called as part of the ASP.NET request pipeline and have access to life-cycle events throughout the request. HTTP modules let you examine incoming and outgoing requests and take action based on the request.

This topic contains:

* [Scenarios](http://msdn.microsoft.com/en-us/library/bb398986.aspx#Scenarios)
* [HTTP Handlers and HTTP Modules Features](http://msdn.microsoft.com/en-us/library/bb398986.aspx#Features)
* [Background](http://msdn.microsoft.com/en-us/library/bb398986.aspx#Background)
* [Code Examples](http://msdn.microsoft.com/en-us/library/bb398986.aspx#CodeExamples)
* [Class Reference](http://msdn.microsoft.com/en-us/library/bb398986.aspx#ClassReference)

http://i.msdn.microsoft.com/Global/Images/clear.gifScenarios

Typical uses for custom HTTP handlers include the following:

* **RSS feeds** To create an RSS feed for a Web site, you can create a handler that emits RSS-formatted XML. You can then bind a file name extension such as .rss to the custom handler. When users send a request to your site that ends in .rss, ASP.NET calls your handler to process the request.
* **Image server** If you want a Web application to serve images in a variety of sizes, you can write a custom handler to resize images and then send them to the user as the handler's response.

Typical uses for HTTP modules include the following:

* **Security** Because you can examine incoming requests, an HTTP module can perform custom authentication or other security checks before the requested page, XML Web service, or handler is called. In Internet Information Services (IIS) 7.0 running in Integrated mode, you can extend forms authentication to all content types in an application.
* **Statistics and logging** Because HTTP modules are called on every request, you can gather request statistics and log information in a centralized module, instead of in individual pages.
* **Custom headers or footers** Because you can modify the outgoing response, you can insert content such as custom header information into every page or XML Web service response.

http://i.msdn.microsoft.com/Global/Images/clear.gifFeatures

HTTP handler and module features include the following:

* The [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) and [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx) interfaces are the starting point for developing handlers and modules.
* The [IHttpAsyncHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.aspx) interface is the starting point for developing asynchronous handlers.
* Custom handler and module source code can be put in the App\_Code folder of an application, or it can be compiled and put in the Bin folder of an application.
* Handlers and modules developed for use in IIS 6.0 can be used in IIS 7.0 with little or no change. For more information, see [Moving an ASP.NET Application from IIS 6.0 to IIS 7.0](http://msdn.microsoft.com/en-us/library/bb515251.aspx).
* Modules can subscribe to a variety of request-pipeline notifications. Modules can receive notification of events of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) object.
* In IIS 7.0, the request pipeline is integrated with the Web server request pipeline. HTTP modules can be used for any request to the Web server, not just ASP.NET requests.

http://i.msdn.microsoft.com/Global/Images/clear.gifBackground

**HTTP Handlers**

An ASP.NET HTTP handler is the process that runs in response to a request that is made to an ASP.NET Web application. The most common handler is an ASP.NET page handler that processes .aspx files. When users request an .aspx file, the request is processed by the page handler.

The ASP.NET page handler is only one type of handler. ASP.NET includes several other built-in handlers such as the Web service handler for .asmx files.

**Built-in HTTP Handlers in ASP.NET**

ASP.NET maps HTTP requests to HTTP handlers based on a file name extension. Each HTTP handler can process individual HTTP URLs or groups of URL extensions in an application. ASP.NET includes several built-in HTTP handlers, as listed in the following table.

|  |  |
| --- | --- |
| **Handler** | **Description** |
| ASP.NET page handler (\*.aspx) | The default HTTP handler for all ASP.NET pages. |
| Web service handler (\*.asmx) | The default HTTP handler for Web service pages created as .asmx files in ASP.NET. |
| Generic Web handler (\*.ashx) | The default HTTP handler for all Web handlers that do not have a UI and that include the [@ WebHandler](http://msdn.microsoft.com/en-us/library/ms366713.aspx) directive. |
| Trace handler (trace.axd) | A handler that displays current page trace information. For details, see [How to: View ASP.NET Trace Information with the Trace Viewer](http://msdn.microsoft.com/en-us/library/wwh16c6c.aspx). |

**Creating a Custom HTTP Handler**

To create a custom HTTP handler, you create a class that implements the [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) interface to create a synchronous handler. Alternatively, you can implement [IHttpAsyncHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.aspx) to create an asynchronous handler. Both handler interfaces require that you implement the [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) property and the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method. The [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) property specifies whether the [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) object (the object that actually calls the appropriate handler) can put the handler in a pool and reuse it to increase performance. If the handler cannot be pooled, the factory must create a new instance of the handler every time that the handler is needed.

The [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method is responsible for processing individual HTTP requests. In this method, you write the code that produces the output for the handler.

HTTP handlers have access to the application context. This includes the requesting user's identity (if known), application state, and session information. When an HTTP handler is requested, ASP.NET calls the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method of the appropriate handler. The code that you write in the handler's [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method creates a response, which is sent back to the requesting browser.

**Mapping a File Name Extension**

When you create a class file as your HTTP handler, the handler can respond to any file name extension that is not already mapped in IIS and in ASP.NET. For example, if you are creating an HTTP handler for generating an RSS feed, you can map your handler to the .rss file name extension. In order for ASP.NET to know which handler to use for your custom file name extension, in IIS you must map the extension to ASP.NET. Then in the application, you must map the extension to the custom handler.

By default, ASP.NET maps the file name extension .ashx to an HTTP handler. If you add the [@ WebHandler](http://msdn.microsoft.com/en-us/library/ms366713.aspx) directive to a class file, ASP.NET automatically maps the .ashx file name extension to the default HTTP handler. This is similar to the way ASP.NET maps the .aspx file name extension to the ASP.NET page handler when the [@ Page](http://msdn.microsoft.com/en-us/library/ydy4x04a.aspx) directive is used. Therefore, if you create an HTTP handler class that has the file name extension .ashx, the handler is automatically registered with IIS and ASP.NET.

If you want to create a custom file name extension for a handler, you must explicitly register the extension with IIS and ASP.NET. The advantage of not using the .ashx file name extension is that the handler is then reusable for different extension mappings. For example, in one application the custom handler might respond to requests that end in .rss. In another application, it might respond to requests that end in .feed. As another example, the handler might be mapped to both file name extensions in the same application, but might create different responses based on the extension.

The process for registering a handler's custom file name extension is different in IIS 7.0 and in earlier versions of IIS. For more information, see [How to: Register HTTP Handlers](http://msdn.microsoft.com/en-us/library/46c5ddfy.aspx) and [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

**Asynchronous and Synchronous HTTP Handlers**

An HTTP handler can be either synchronous or asynchronous. A synchronous handler does not return until it finishes processing the HTTP request for which it is called. An asynchronous handler runs a process independently of sending a response to the user. Asynchronous handlers are useful when you must start an application process that might be lengthy and the user does not have to wait until it finishes before receiving a response from the server.

Asynchronous HTTP handlers enable you to start an external process, such as a method call to a remote server. The handler can then continue processing without waiting for the external process to finish. While an asynchronous HTTP handler is processing, ASP.NET puts the thread that would ordinarily be used for the external process back into the thread pool until the handler receives a callback from the external process. This can prevent thread blocking and can improve performance because only a limited number of threads can execute at the same time. If many users request synchronous HTTP handlers that rely on external processes, the operating system can run out of threads, because many threads are blocked and are waiting for an external process.

When you create an asynchronous handler, you must implement the [IHttpAsyncHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.aspx) interface. You must also implement the [BeginProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.beginprocessrequest.aspx) method in order to initiate an asynchronous call that processes individual HTTP requests. In addition, you must implement the [EndProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.defaulthttphandler.endprocessrequest.aspx) method to run cleanup code when the process ends.

**Custom IHttpHandlerFactory Classes**

The [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) class receives requests and is responsible for forwarding a request to the appropriate HTTP handler. You can create a custom HTTP handler factory by creating a class that implements the [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) interface. A custom handler factory can give you finer control over how HTTP requests are processed by creating different handlers based on run-time conditions. For example, with a custom HTTP handler factory, you can instantiate one HTTP handler for a file type if the HTTP request method is PUT, and another if the method is GET.

To register a custom extension for a handler factory, follow the steps for registering a custom extension for a handler. For an example of creating and registering a handler factory, see [Walkthrough: Creating and Registering HTTP Handler Factories](http://msdn.microsoft.com/en-us/library/ms227439.aspx).

**HTTP Modules**

An HTTP module is an assembly that is called on every request that is made to your application. HTTP modules are called as part of the request pipeline and have access to life-cycle events throughout the request. HTTP modules therefore let you examine incoming requests and take action based on the request. They also let you examine the outgoing response and modify it.

In IIS 6.0, the ASP.NET request pipeline is separate from the Web server request pipeline. In IIS 7.0, the ASP.NET request pipeline and the Web server request pipeline can be integrated into a common request pipeline. In IIS 7.0, this is referred to as *Integrated* mode. The unified pipeline has several benefits for ASP.NET developers. For example, it lets managed-code modules receive pipeline notifications for all requests, even if the requests are not for ASP.NET resources. However, if you want, you can run IIS 7.0 in *Classic* mode, which emulates ASP.NET running in IIS 6.0. For more information, see [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

ASP.NET HTTP modules are like ISAPI filters because they are invoked for all requests. However, they are written in managed code and are fully integrated with the life cycle of an ASP.NET application. You can put custom module source code in the App\_Code folder of your application, or you can put compiled custom modules as assemblies in the Bin folder of an application.

ASP.NET uses modules to implement various application features, which includes forms authentication, caching, session state, and client script services. In each case, when those services are enabled, the module is called as part of a request and performs tasks that are outside the scope of any single page request. Modules can consume application events and can raise events that can be handled in the Global.asax file. For more information about application events, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx) and [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

|  |
| --- |
| **NoteNote** |
| HTTP modules differ from HTTP handlers. An HTTP handler returns a response to a request that is identified by a file name extension or family of file name extensions. In contrast, an HTTP module is invoked for all requests and responses. It subscribes to event notifications in the request pipeline and lets you run code in registered event handlers. The tasks that a module is used for are general to an application and to all requests for resources in the application. |

**How HTTP Modules Work**

Modules must be registered to receive notifications from the request pipeline. The most common way to register an HTTP module is in the application's Web.config file. In IIS 7.0, the unified request pipeline also enables you to register a module in other ways, which includes through IIS Manager and through the Appcmd.exe command-line tool. For more information, see [Configuring Handler Mappings in IIS 7.0](http://go.microsoft.com/fwlink/?LinkId=92338) and [Start Appcmd.exe](http://go.microsoft.com/fwlink/?LinkId=93419).

When ASP.NET creates an instance of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.aspx) class that represents your application, instances of any modules that have been registered are created. When a module is created, its [Init](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init.aspx) method is called and the module initializes itself. For more information, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx) and ASP.NET [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

In a module's [Init](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init.aspx) method, you can subscribe to various application events such as [BeginRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.beginrequest.aspx) or [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest.aspx) by binding the events to methods in the module.

|  |
| --- |
| **NoteNote** |
| For modules that operate in the IIS 7.0 integrated pipeline, you should register event handlers in the [Init](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init.aspx) method. |

When application events are raised, the appropriate method in your module is called. The method can perform whatever logic is required, such as checking authentication or logging request information. During event handling, the module has access to the [Context](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.context.aspx) property of the current request. This enables you to redirect the request to an alternative page, modify the request, or perform any other request manipulation. For example, if the module checks authentication, the module might redirect to a login or error page if the credentials are not correct. Otherwise, when the module's event handler has finished running, ASP.NET calls the next process in the pipeline. This might be another module or might be the appropriate HTTP handler (such as an .aspx file) for the request.

**HTTP Modules versus Global.asax Files**

You can implement much of the functionality of a module in the application's Global.asax file, which enables you to respond to application events. However, modules have an advantage over the Global.asax file because they are encapsulated and can be created one time and used in many different applications. By adding them to the global assembly cache and registering them in the Machine.config file, you can reuse them across applications. For more information, see [Global Assembly Cache](http://msdn.microsoft.com/en-us/library/yf1d93sz.aspx).

|  |
| --- |
| **NoteNote** |
| In IIS 7.0, the integrated pipeline enables managed modules to subscribe to pipeline notifications for all requests, not just requests for ASP.NET resources. Event handlers in the Global.asax file are invoked only for notifications during requests for resources in the application. Custom modules in Integrated mode can be explicitly scoped to receive event notifications only for requests to the application. Otherwise, custom modules receive event notification for all requests to the application. If the **precondition** attribute of the **add** element of the **modules** section is set to "managedHandler", the module is scoped to the application. |

The advantage of using the Global.asax file is that you can handle other registered events such as **Session\_Start** and **Session\_End**. In addition, the Global.asax file enables you to instantiate global objects that are available throughout the application.

You should use a module whenever you must create code that depends on application events, and when the following conditions are true:

* You want to re-use the module in other applications.
* You want to avoid putting complex code in the Global.asax file.
* The module applies to all requests in the pipeline (IIS 7.0 Integrated mode only).

You should add code in the Global.asax file whenever you must create code that depends on application events and you do not have to reuse it across applications. You can also use the Global.asax file when you have to subscribe to events that are not available to modules, such as **Session\_Start**.

**Creating an HTTP Module**

The general process for writing an HTTP module is as follows:

* Create a class that implements the [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.aspx) interface.
* Write a handler for the [Init](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init.aspx) method. The initialization method should initialize your module and subscribe to any application events you need. For example, if you want to append something to responses, you might subscribe to the [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest.aspx) event. If you want to perform custom authentication logic, you might subscribe to the [AuthenticateRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.authenticaterequest.aspx) event. For more information about application events, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx).
* Write code for the events that you have subscribed to.
* Optionally implement the [Dispose](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.dispose.aspx) method if the module requires cleanup.
* Register the module in the Web.config file. For more information, see [Walkthrough: Creating and Registering a Custom HTTP Module](http://msdn.microsoft.com/en-us/library/ms227673.aspx)

For information about how to move a module from an application that is running in IIS 6.0 or earlier versions to an application that runs on in IIS 7.0, see [Moving an ASP.NET Application from IIS 6.0 to IIS 7.0](http://msdn.microsoft.com/en-us/library/bb515251.aspx).

**IHttpHandler Interface**

Defines the contract that ASP.NET implements to synchronously process HTTP Web requests using custom HTTP handlers.

**Namespace:** [System.Web](http://msdn.microsoft.com/en-us/library/system.web.aspx)  
**Assembly:** System.Web (in System.Web.dll)

Syntax

Visual Basic

Public Interface IHttpHandler

C#

public interface IHttpHandler

Visual C++

public interface class IHttpHandler

F#

type IHttpHandler = interface end

The **IHttpHandler** type exposes the following members.

Properties

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public property | [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) | Gets a value indicating whether another request can use the **IHttpHandler** instance. |

Methods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public method | [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) | Enables processing of HTTP Web requests by a custom **HttpHandler** that implements the **IHttpHandler** interface. |

Remarks

You can write custom HTTP handlers to process specific, predefined types of HTTP requests in any Common Language Specification (CLS) compliant language. Executable code defined in the **HttpHandler** classes, rather than conventional ASP or ASP.NET Web pages, responds to these specific requests. HTTP handlers give you a means of interacting with the low-level request and response services of the IIS Web server and provide functionality much like ISAPI extensions but with a simpler programming model.

If your handler will access session state values, it must implement the [IRequiresSessionState](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.irequiressessionstate.aspx) interface (a marker interface with no methods).

**IHttpModule Interface**

Provides module initialization and disposal events to the implementing class.

**Namespace:** [System.Web](http://msdn.microsoft.com/en-us/library/system.web.aspx)  
**Assembly:** System.Web (in System.Web.dll)

Syntax

Visual Basic

Public Interface IHttpModule

C#

public interface IHttpModule

Visual C++

public interface class IHttpModule

F#

type IHttpModule = interface end

The **IHttpModule** type exposes the following members.

Methods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public method | [Dispose](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.dispose.aspx) | Disposes of the resources (other than memory) used by the module that implements **IHttpModule**. |
| Public method | [Init](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init.aspx) | Initializes a module and prepares it to handle requests. |

.NET Framework Class Library

**IHttpAsyncHandler Interface**

Defines the contract that HTTP asynchronous handler objects must implement.

**Namespace:** [System.Web](http://msdn.microsoft.com/en-us/library/system.web.aspx)  
**Assembly:** System.Web (in System.Web.dll)

Syntax

Visual Basic

Public Interface IHttpAsyncHandler \_

Inherits IHttpHandler

C#

public interface IHttpAsyncHandler : IHttpHandler

Visual C++

public interface class IHttpAsyncHandler : IHttpHandler

F#

type IHttpAsyncHandler =

interface

interface IHttpHandler

end

The **IHttpAsyncHandler** type exposes the following members.

Properties

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public property | [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) | Gets a value indicating whether another request can use the [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) instance. (Inherited from [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx).) |

[Top](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.aspx#mainBody)

Methods

|  |  |  |
| --- | --- | --- |
|  | **Name** | **Description** |
| Public method | [BeginProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.beginprocessrequest.aspx) | Initiates an asynchronous call to the HTTP handler. |
| Public method | [EndProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.endprocessrequest.aspx) | Provides an asynchronous process End method when the process ends. |
| Public method | [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) | Enables processing of HTTP Web requests by a custom **HttpHandler** that implements the [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx) interface. (Inherited from [IHttpHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.aspx).) |

Remarks

An **IHttpAsyncHandler** class can be associated with a file name extension or a particular URL by a configuration file, in the **httpHandlers** configuration section. The ASP.NET infrastructure will then instantiate and call the handler when the corresponding request is received. Alternatively, the handler can be defined in an .ashx file and when the corresponding request is received for the .ashx file the hander will be executed.