**Introduction to HTTP Handlers**

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 via the page handler.

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

You can create custom HTTP handlers when you want special handling that you can identify using file name extensions in your application. For example, the following scenarios would be good uses of custom HTTP handlers:

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

HTTP handlers have access to the application context, including 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(VS.85).aspx) method on the appropriate handler. The handler's **ProcessRequest** method creates a response, which is sent back to the requesting browser. As with any page request, the response goes through any HTTP modules that have subscribed to events that occur after the handler has run. For more information about the processing of Web server requests, see [ASP.NET Application Life Cycle Overview](http://msdn.microsoft.com/en-us/library/ms178473(VS.85).aspx).

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 need to start an application process that might be lengthy and the user does not need to wait until it finishes before getting a response from the server.

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

ASP.NET maps HTTP requests to HTTP handlers based on a file name extension. Each HTTP handler enables processing of individual HTTP URLs or groups of URL extensions within 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 using ASP.NET. |
| ASP.NET user control handler (\*.ascx) | The default HTTP handler for all ASP.NET user control pages. |
| 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(VS.85).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(VS.85).aspx) interface to create a synchronous handler or the [IHttpAsyncHandler](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler(VS.85).aspx) to create an asynchronous handler. Both handler interfaces require you to implement the [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable(VS.85).aspx) property and the **ProcessRequest** method. The **IsReusable** property specifies whether the [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory(VS.85).aspx) object (the object that actually calls the appropriate handler) can place your handlers in a pool and reuse them to increase performance, or whether it must create new instances every time the handler is needed. The **ProcessRequest** method is responsible for actually processing the individual HTTP requests.

**Creating a File Name Extension**

When you create a class file as your HTTP handler, you can have your handler 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 extension .rss. In order for ASP.NET to know which handler to use for your custom file name extension, the extension of the handler's class file must be mapped in IIS to ASP.NET, and in your application to your custom handler.

By default, ASP.NET maps the file name extension .ashx for custom HTTP handlers, in the same way that it maps the .aspx extension to the ASP.NET page handler. Therefore, if you create an HTTP handler class with 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 your handler, you must explicitly register the extension with IIS and ASP.NET. The advantage of not using the .ashx file name extension is that your handler is then reusable for different extension mappings. For example, in one application your custom handler might respond to requests that end in .rss, and in another application it might respond to requests that end in .feed. As another example, your handler might be mapped to both file name extensions in the same application, but might create different responses based on the extension.

**Asynchronous HTTP Handlers**

Asynchronous HTTP handlers allow you to start an external process, such as a method call to a remote server, and then continue the processing of the handler without waiting for the external process to finish. During the processing of an asynchronous HTTP handler, ASP.NET places 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 greatly improve performance, because only a limited number of threads can be executed at once. If a number of users request synchronous HTTP handlers that rely on external processes, the operating system can quickly run out of threads because many threads are blocked and waiting for an external process.

When you create an asynchronous handler, in addition to implementing the **IHttpAsyncHandler** interface, you must implement the [BeginProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.beginprocessrequest(VS.85).aspx) to initiate an asynchronous call for processing individual HTTP requests. You must also implement the [EndProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.defaulthttphandler.endprocessrequest(VS.85).aspx) method to run cleanup code when the process ends.

**Custom IHttpHandlerFactory Classes**

The **IHttpHandlerFactory** class receives requests and is responsible for forwarding a request to an appropriate HTTP handler. You can create a custom HTTP handler factory by creating a class that implements the **IHttpHandlerFactory** interface. Creating a custom handler factory can allow finer control over the processing of an HTTP request 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.

**How to: Register HTTP Handlers**

After you have created a custom HTTP handler class, you must register it in the Web.config file. This enables ASP.NET to call the HTTP handler in order to service requests for resources that have the specified file name extension.

How you register an HTTP handler depends on the version of Internet Information Services (IIS) that hosts your application. For IIS 6.0, you register the handler by using the [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) section of the Web.config file. For IIS 7.0 running in Classic mode, you register the handler in the [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) section, and you map the handler to the Aspnet\_isapi.dll file. For IIS 7.0 running in Integrated mode, you register the handler by using the **handlers** element in the **system.WebServer** section.

**To register an HTTP handler for IIS 6.0**

1. Compile the HTTP handler class and copy the resulting assembly to the Bin folder under the application's root folder.

-or-

Put the source code for the handler into the application's App\_Code folder.

For an example of an HTTP handler, see [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx).

1. In the application's Web.config file, create an [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) section.

The following example shows how to register an HTTP handler that responds to requests for the SampleHandler.new resource. The handler is defined as the class SampleHandler in the assembly SampleHandlerAssembly.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05other');)

<configuration>

<system.web>

<httpHandlers>

<add verb="\*" path="SampleHandler.new"

type="SampleHandler, SampleHandlerAssembly" />

</httpHandlers>

<system.web>

</configuration>

The following example maps all HTTP requests for files that have the file name extension ".SampleFileExtension" to the SampleHandler2 class. In this case, the handler code is in the App\_Code folder, so you do not have to specify an assembly.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl06other');)

<configuration>

<system.web>

<httpHandlers>

<add verb="\*" path="\*.SampleFileExtension"

type="SampleHandler2 " />

</httpHandlers>

<system.web>

</configuration>

1. Configure IIS to forward the request for the custom file name extension to ASP.NET.

For more information, see [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

**To register an HTTP handler for IIS 7.0 running in Classic mode**

1. Compile the HTTP handler class and copy the resulting assembly to the Bin folder under the application's root folder.

-or-

Put the source code for the handler into the application's App\_Code folder.

For an example of an HTTP handler, see [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx).

1. In the application's Web.config file, create an [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) section.
2. Create a **system.webServer** section inside the [configuration](http://msdn.microsoft.com/en-us/library/ms228147.aspx) element.
3. Create a **handlers** element inside the **system.WebServer** section.

|  |
| --- |
| **46c5ddfy.alert_note(en-us,VS.90).gifNote:** |
| You must define both an [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) element and a **handlers** element. |

1. The following example shows how to register an HTTP handler that responds to requests for the SampleHandler.new resource. The handler is defined as the class SampleHandler in the assembly SampleHandlerAssembly.
2. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13other');)
3. <configuration>
4. <system.web>
5. <httpHandlers>
6. <add verb="\*" path="SampleHandler.new"
7. type="SampleHandler, SampleHandlerAssembly" />
8. </httpHandlers>
9. <system.web>
10. <system.webServer>
11. <add name=SampleHandler" verb="\*" path="SampleHandler.new"
12. Modules="IsapiModule"
13. scriptProcessor="FrameworkPath\aspnet\_isapi.dll"
14. resourceType="File" />
15. </system.webServer>
16. </configuration>
17. Replace FrameworkPath with the correct path to the Aspnet\_isapi.dll file.
18. The following example maps all HTTP requests for files that have the file name extension ".SampleFileExtension" to the SampleHandler2 class. In this case, the handler code is in the App\_Code folder, so you do not have to specify an assembly.
19. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14other');)
20. <configuration>
21. <system.web>
22. <httpHandlers>
23. <add verb="\*" path="\*.SampleFileExtension"
24. type="SampleHandler2" />
25. </httpHandlers>
26. <system.web>
27. <system.webServer>
28. <add name=SampleHandler2" verb="\*" path="\*.SampleFileExtension"
29. Modules="IsapiModule"
30. scriptProcessor="FrameworkPath\aspnet\_isapi.dll"
31. resourceType="File" />
32. </system.webServer>
33. </configuration>
34. Replace FrameworkPath with the correct path to the Aspnet\_isapi.dll file.

|  |
| --- |
| **46c5ddfy.alert_note(en-us,VS.90).gifNote:** |
| For IIS 7.0 running in Classic mode, you do not have to separately use IIS Manager to map the file name extension to the Aspnet\_isapi.dll file, as you do with IIS 6.0. You can map the extension in the Web.config file. |

**To register an HTTP handler for IIS 7.0 running in Integrated Mode**

1. Compile the HTTP handler class and copy the resulting assembly to the Bin folder under the application's root folder.

-or-

Put the source code for the handler into the application's App\_Code folder.

For an example of an HTTP handler, see [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx).

1. In the application's Web.config file, create a **handlers** element in the **system.webServer** section.

|  |
| --- |
| **46c5ddfy.alert_note(en-us,VS.90).gifNote:** |
| Handlers that are defined in the [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) element are not used. If you do not remove the [httpHandlers](http://msdn.microsoft.com/en-us/library/bya7fh0a.aspx) registrations, you must set the **validation** element’s **validateIntegratedModeConfiguration** attribute to **false** in order to avoid errors. The **validation** element is a child element of the **system.webServer** element. For more information, see "Disabling the migration error message" in [ASP.NET Integration with IIS 7.0](http://go.microsoft.com/fwlink/?LinkId=88850). |

1. The following example shows how to register an HTTP handler that responds to requests for the SampleHandler.new resource. The handler is defined as the class SampleHandler in the assembly SampleHandlerAssembly.
2. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl21other');)
3. <configuration>
4. <system.webServer>
5. <handlers>
6. <add name="SampleHandler" verb="\*"
7. path="SampleHandler.new"
8. type="SampleHandler, SampleHandlerAssembly"
9. resourceType="Unspecified" />
10. </handlers>
11. <system.webServer>
12. </configuration>

|  |
| --- |
| **46c5ddfy.alert_note(en-us,VS.90).gifNote:** |
| The **resourceType** attribute performs the same function as the Verify file exists option in IIS manager for IIS 6.0. |

1. The following example shows how to map all HTTP requests to files with the file name extension ".SampleFileExtension" to the SampleHandler2 HTTP handler class. In this case, the handler code is in the App\_Code folder, so you do not have to specify an assembly.
2. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl23other');)
3. <configuration>
4. <system.webServer>
5. <handlers>
6. <add name="SampleHandler2" verb="\*"
7. path="\*.SampleFileExtension"
8. type="SampleHandler2" />
9. resourceType="Unspecified" />
10. <handlers>
11. <system.webServer>
12. </configuration>
13. For IIS 7.0 running in Integrated mode, only the registration in the **handlers** element is required.

**Walkthrough: Creating a Synchronous HTTP Handler**

This walkthrough illustrates how to create an HTTP handler that performs synchronous processing of requests. The example handler processes requests for resources in an ASP.NET application whose URL ends with .sample.

When users request a resource whose URL ends in .sample, the Web server forwards the request to ASP.NET. ASP.NET then calls the HTTP handler, which returns a response. The response is created dynamically by the handler. There is no need for a file that has the file name extension .sample to exist.

For more information about how the ASP.NET runtime interacts with IIS 6.0, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx). For more information about ASP.NET integration with IIS 7.0, see [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

Tasks illustrated in this walkthrough include the following:

* How to create the code for an HTTP handler class. The class must implement the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method and the [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) property.
* How to register the handler in the Web.config file and map the .sample file name extension to it.
* How to map the .sample file name extension to ASP.NET In Internet Information Services (IIS).

|  |
| --- |
| **ms228090.alert_note(en-us,VS.90).gifNote:** |
| The ASP.NET Development Server will serve the request for the resource after the configuration file is changed to include a reference to the new handler. For more information about the ASP.NET Development Server, see [Web Servers in Visual Web Developer](http://msdn.microsoft.com/en-us/library/58wxa9w5.aspx). To enable IIS to serve the request, see the procedures later in this walkthrough. |

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

In order to complete this walkthrough, you will need:

* Visual Studio or Visual Web Developer.
* An ASP.NET Web site that you can run by using IIS.
* IIS 6.0 or IIS 7.0.

http://i.msdn.microsoft.com/Global/Images/clear.gif Creating a Web Site that Runs Under IIS

For this walkthrough, you must run a Web site by using IIS.

**To create a Web site that runs under IIS**

1. Open Visual Studio or Visual Web Developer.
2. In the File menu, click New Web Site.
3. In the Location list, select HTTP and then enter "http://localhost/HttpHandler" in the text box.
4. Click OK.

http://i.msdn.microsoft.com/Global/Images/clear.gif Creating a Synchronous HTTP Handler Class

**To create the custom HelloWorldHandler HTTP handler class**

1. In Solution Explorer, right-click the project, click Add ASP.NET Folder, and then click App\_Code.
2. In the App\_Code folder, create a class named HelloWorldHandler and add the following code to the class file.

Visual Basic

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18VisualBasic');)

Imports System.Web

Public Class HelloWorldHandler

Implements IHttpHandler

Public Sub ProcessRequest(ByVal context As \_

System.Web.HttpContext) Implements \_

System.Web.IHttpHandler.ProcessRequest

Dim request As HttpRequest = context.Request

Dim response As HttpResponse = context.Response

' This handler is called whenever a file ending

' in .sample is requested. A file with that extension

' does not need to exist.

response.Write("<html>")

response.Write("<body>")

response.Write("<h1>Hello from a synchronous custom HTTP handler.</h1>")

response.Write("</body>")

response.Write("</html>")

End Sub

Public ReadOnly Property IsReusable() As Boolean \_

Implements System.Web.IHttpHandler.IsReusable

Get

Return False

End Get

End Property

End Class

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl19CSharp');)

using System.Web;

public class HelloWorldHandler : IHttpHandler

{

public HelloWorldHandler()

{

}

public void ProcessRequest(HttpContext context)

{

HttpRequest Request = context.Request;

HttpResponse Response = context.Response;

// This handler is called whenever a file ending

// in .sample is requested. A file with that extension

// does not need to exist.

Response.Write("<html>");

Response.Write("<body>");

Response.Write("<h1>Hello from a synchronous custom HTTP handler.</h1>");

Response.Write("</body>");

Response.Write("</html>");

}

public bool IsReusable

{

// To enable pooling, return true here.

// This keeps the handler in memory.

get { return false; }

}

}

The code implements the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method and writes a string to the [Response](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.response.aspx) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) object.

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler in IIS 6.0

After you have created the custom HTTP handler class, you must register it in the application's Web.config file. This enables ASP.NET to find the handler when requests are made for resources whose URL ends with .sample.

There are different procedures for registering the handler, depending on whether you are working with IIS 6.0 or IIS 7.0. This section describes how to register a handler in IIS 6.0. The next section describes how to register a handler in IIS 7.0.

**To register the handler in IIS 6.0**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl27other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl27other');)

<configuration>

<system.web>

<httpHandlers>

<add verb="\*" path="\*.sample"

type="HelloWorldHandler"/>

</httpHandlers>

</system.web>

</configuration>

The configuration element registers the custom handler by class name, and it maps the .sample file name extension to that handler.

Register an application extension mapping for the .sample file name extension by using IIS Manager. For more information, see [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler in IIS 7.0

In IIS 7.0, an application can run in either Classic or Integrated mode. In Classic mode, requests are processed much the same way as they are in IIS 6.0. In Integrated mode, IIS 7.0 manages requests by using a pipeline that enables it to share requests, modules, and other features with ASP.NET.

For IIS 7.0, the handler registration requires either registering the handler in the Web.config file or in IIS Manager. Because of the centralized administration in IIS 7.0, changes in an application's Web.config file are reflected in IIS Manager interface for the application and vice versa. In the following procedures, the handlers are registered in the Web.config file.

There are different procedures for registering the handler for IIS 7.0 running in Classic mode and running in Integrated mode. Follow the procedure for the IIS mode that you are using.

**To register the handler in IIS 7.0 running in Classic mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

|  |
| --- |
| **ms228090.alert_note(en-us,VS.90).gifNote:** |
| Substitute the correct path of the aspnet\_isapi.dll file. The .dll file is in the folder where the .NET Framework is installed. By default this is C:\WINDOWS\Microsoft.NET\Framework\version. |

1. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl37other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl37other');)
2. <configuration>
3. <system.web>
4. <httpHandlers>
5. <add verb="\*" path="\*.sample"
6. type="HelloWorldHandler"/>
7. </httpHandlers>
8. </system.web>
9. <system.webServer>
10. <handlers>
11. <add verb="\*" path="\*.sample"
12. name="HelloWorldHandler"
13. type="HelloWorldHandler"
14. modules="IsapiModule"/>
15. scriptProcessor="%path%\aspnet\_isapi.dll"
16. </handlers>
17. </system.webServer>
18. </configuration>
19. The configuration element registers the custom handler by class name and maps the .sample file name extension to that handler.

|  |
| --- |
| **ms228090.alert_note(en-us,VS.90).gifNote:** |
| Because you are registering a custom file name extension, you register the handler in both the **handlers** section and the **httpHandlers** section. In Classic mode, for backward compatibility, the handler is specified as an ISAPI module by using the **modules** attribute. The path of the ASP.NET ISAPI dll is specified by using the **scriptProcessor** attribute. The **name** attribute is required in the **handlers** section. |

**To register the handler in IIS 7.0 running in Integrated mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39other');)

<configuration>

<system.webServer>

<handlers>

<add verb="\*" path="\*.sample"

name="HelloWorldHandler"

type="HelloWorldHandler"/>

</handlers>

</system.webServer>

</configuration>

The configuration element registers the custom handler by class name and maps the .sample file name extension to that handler.

|  |
| --- |
| **ms228090.alert_note(en-us,VS.90).gifNote:** |
| The registration is done in the **handlers** section, but not in the **httpHandlers** section. The **name** attribute is required. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Testing the Custom HTTP Handler

After you have created and registered the custom HTTP handler, you can test it.

**To test your custom HTTP handler**

1. In the browser, request a page from the Web application.
2. In the browser, enter a URL that ends in **.sample**. For example, enter the following URL:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl44other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl44other');)

http://localhost/HttpHandler/test.sample

The text defined in the HelloWorldHandler class is displayed.

**Walkthrough: Creating an Asynchronous HTTP Handler**

This walkthrough illustrates how to create an asynchronous HTTP handler. Asynchronous HTTP handlers enable you to start an external process (such as a method call to a remote server) while the handler continues processing. The handler can continue without waiting for the external process to finish.

During processing of an asynchronous HTTP handler, 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 improve performance, because only a limited number of threads can be executing at the same time. If many users request synchronous HTTP handlers that rely on external processes, the operating system can quickly run out of threads because many threads are blocked and waiting for an external process.

The example in this walkthrough demonstrates an asynchronous HTTP handler that processes requests for files that have the file name extension .SampleAsync in an ASP.NET application. The example shows the code for the handler, and then how to map the .SampleAsync extension to the handler in ASP.NET. The example also shows how to map the .SampleAsync extension to ASP.NET in IIS, so that IIS forwards requests that end in .SampleAsync to ASP.NET.

For more information about how the ASP.NET runtime interacts with IIS 6.0, see [ASP.NET Application Life Cycle Overview for IIS 5.0 and 6.0](http://msdn.microsoft.com/en-us/library/ms178473.aspx). For more information about ASP.NET integration with IIS 7.0, see [ASP.NET Application Life Cycle Overview for IIS 7.0](http://msdn.microsoft.com/en-us/library/bb470252.aspx).

Tasks illustrated in this walkthrough include the following:

* How to create the code for an HTTP handler class. The class must implement the [ProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.processrequest.aspx) method and the [IsReusable](http://msdn.microsoft.com/en-us/library/system.web.ihttphandler.isreusable.aspx) property.
* How to register the handler in the Web.config file and map the .SampleAsync file name extension to it.
* How to map the .sample file name extension to ASP.NET in IIS.

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

In order to complete this walkthrough, you will need:

* Visual Studio or Visual Web Developer.
* An ASP.NET Web site that you can run by using IIS.
* IIS 6.0 or IIS 7.0.

http://i.msdn.microsoft.com/Global/Images/clear.gif Creating an Asynchronous HTTP Handler Class

To begin, you will create a class that implements the asynchronous handler.

**To create the HelloWorldAsyncHandler HTTP handler class**

1. If the Web site that you are working with does not already have an App\_Code folder, create one under the root of the site.
2. In the App\_Code directory, create a class named HelloWorldAsyncHandler and add the following code to the class file:

Visual Basic

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl17VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl17VisualBasic');)

Imports Microsoft.VisualBasic

Imports System.Web

Imports System.Threading

Public Class HelloWorldAsyncHandler

Implements IHttpAsyncHandler

Public ReadOnly Property IsReusable() As Boolean Implements System.Web.IHttpHandler.IsReusable

Get

Return False

End Get

End Property

Public Function BeginProcessRequest( \_

ByVal context As System.Web.HttpContext, \_

ByVal cb As System.AsyncCallback, \_

ByVal extraData As Object) \_

As System.IAsyncResult \_

Implements System.Web.IHttpAsyncHandler.BeginProcessRequest

context.Response.Write("<p>Begin IsThreadPoolThread is " \_

& Thread.CurrentThread.IsThreadPoolThread & "</p>" & vbCrLf)

Dim asynch As New AsynchOperation(cb, context, extraData)

asynch.StartAsyncWork()

Return asynch

End Function

Public Sub EndProcessRequest(ByVal result As \_

System.IAsyncResult) \_

Implements System.Web.IHttpAsyncHandler.EndProcessRequest

End Sub

Public Sub ProcessRequest(ByVal context \_

As System.Web.HttpContext) \_

Implements System.Web.IHttpHandler.ProcessRequest

Throw New InvalidOperationException()

End Sub

End Class

Class AsynchOperation

Implements IAsyncResult

Private \_completed As Boolean

Private \_state As [Object]

Private \_callback As AsyncCallback

Private \_context As HttpContext

ReadOnly Property IsCompleted() As Boolean \_

Implements IAsyncResult.IsCompleted

Get

Return \_completed

End Get

End Property

ReadOnly Property AsyncWaitHandle() As WaitHandle \_

Implements IAsyncResult.AsyncWaitHandle

Get

Return Nothing

End Get

End Property

ReadOnly Property AsyncState() As [Object] \_

Implements IAsyncResult.AsyncState

Get

Return \_state

End Get

End Property

ReadOnly Property CompletedSynchronously() As Boolean \_

Implements IAsyncResult.CompletedSynchronously

Get

Return False

End Get

End Property

Public Sub New(ByVal callback As AsyncCallback, \_

ByVal context As HttpContext, \_

ByVal state As [Object])

\_callback = callback

\_context = context

\_state = state

\_completed = False

End Sub

Public Sub StartAsyncWork()

ThreadPool.QueueUserWorkItem(New WaitCallback(AddressOf StartAsyncTask), Nothing)

End Sub

Private Sub StartAsyncTask(ByVal workItemState As [Object])

\_context.Response.Write("<p>Completion IsThreadPoolThread is " & Thread.CurrentThread.IsThreadPoolThread & "</p>" & vbCrLf)

\_context.Response.Write("Hello World from Async Handler!")

\_completed = True

\_callback(Me)

End Sub 'StartAsyncTask

End Class 'AsynchOperation

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl18CSharp');)

using System;

using System.Web;

using System.Threading;

class HelloWorldAsyncHandler : IHttpAsyncHandler

{

public bool IsReusable { get { return false; } }

public HelloWorldAsyncHandler()

{

}

public IAsyncResult BeginProcessRequest(HttpContext context, AsyncCallback cb, Object extraData)

{

context.Response.Write("<p>Begin IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

AsynchOperation asynch = new AsynchOperation(cb, context, extraData);

asynch.StartAsyncWork();

return asynch;

}

public void EndProcessRequest(IAsyncResult result)

{

}

public void ProcessRequest(HttpContext context)

{

throw new InvalidOperationException();

}

}

class AsynchOperation : IAsyncResult

{

private bool \_completed;

private Object \_state;

private AsyncCallback \_callback;

private HttpContext \_context;

bool IAsyncResult.IsCompleted { get { return \_completed; } }

WaitHandle IAsyncResult.AsyncWaitHandle { get { return null; } }

Object IAsyncResult.AsyncState { get { return \_state; } }

bool IAsyncResult.CompletedSynchronously { get { return false; } }

public AsynchOperation(AsyncCallback callback, HttpContext context, Object state)

{

\_callback = callback;

\_context = context;

\_state = state;

\_completed = false;

}

public void StartAsyncWork()

{

ThreadPool.QueueUserWorkItem(new WaitCallback(StartAsyncTask), null);

}

private void StartAsyncTask(Object workItemState)

{

\_context.Response.Write("<p>Completion IsThreadPoolThread is " + Thread.CurrentThread.IsThreadPoolThread + "</p>\r\n");

\_context.Response.Write("Hello World from Async Handler!");

\_completed = true;

\_callback(this);

}

}

The code implements the [BeginProcessRequest](http://msdn.microsoft.com/en-us/library/system.web.ihttpasynchandler.beginprocessrequest.aspx) method. The method writes a string to the [Response](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.response.aspx) property of the current [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) object, creates a new instance of the AsyncOperation class, and calls the StartAsyncWork method. The StartAsyncWork method then adds the StartAsyncTask delegate to the [ThreadPool](http://msdn.microsoft.com/en-us/library/system.threading.threadpool.aspx) object. When a thread becomes available, the StartAsyncTask method is called, which writes out another string to the [Response](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.response.aspx) property. The task then finishes by invoking the [AsyncCallback](http://msdn.microsoft.com/en-us/library/system.asynccallback.aspx) delegate.

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler in IIS 6.0

After you have created the custom HTTP handler class, you must register it in the application's Web.config file. This enables ASP.NET to find the handler when requests are made for resources whose URL ends with .SampleAsync.

There are different procedures for registering the handler, depending on whether you are working with IIS 6.0 or IIS 7.0. This section describes how to register a handler in IIS 6.0. The next section describes how to register a handler in IIS 7.0.

**To register the handler in IIS 6.0**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted markup to the Web.config file:

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl29other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl29other');)

<configuration>

<system.web>

<httpHandlers>

<add verb="\*" path="\*.SampleAsync"

type="HelloWorldAsyncHandler"/>

</httpHandlers>

</system.web>

</configuration>

The configuration element registers the HelloWorldAsyncHandler handler as the handler for requests that end with .SampleAsync.

1. Register an application extension mapping for the .SampleAsync file name extension by using IIS Manager. For more information, see [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler in IIS 7.0

In IIS 7.0, an application can run in either Classic or Integrated mode. In Classic mode, requests are processed much the same way as they are in IIS 6.0. In Integrated mode, IIS 7.0 manages requests by using a pipeline that enables it to share requests, modules, and other features with ASP.NET.

For IIS 7.0, the handler registration requires registering the handler either in the Web.config file or in IIS Manager. Because administration is centralized in IIS 7.0, changes in an application's Web.config file are reflected in the IIS Manager interface for the application and vice versa. In the procedures that follow, the handlers are registered in the Web.config file.

There are different procedures for registering the handler for IIS 7.0 running in Classic mode and running in Integrated mode. Follow the procedure for the IIS mode that you are using.

**To register the handler in IIS 7.0 running in Classic mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

|  |
| --- |
| **ms227433.alert_note(en-us,VS.90).gifNote:** |
| Substitute the correct path for the aspnet\_isapi.dll file. The .dll file is in the folder where the .NET Framework is installed. By default this is C:\WINDOWS\Microsoft.NET\Framework\version. |

1. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl39other');)
2. <configuration>
3. <system.web>
4. <httpHandlers>
5. <add verb="\*" path="\*.SampleAsync"
6. type="HelloWorldAsyncHandler"/>
7. </httpHandlers>
8. </system.web>
9. <system.webServer>
10. <handlers>
11. <add verb="\*" path="\*.SampleAsync"
12. name="HelloWorldAsyncHandler"
13. type="HelloWorldAsyncHandler"
14. modules="IsapiModule"/>
15. scriptProcessor="%path%\aspnet\_isapi.dll"
16. </handlers>
17. </system.webServer>
18. </configuration>
19. The configuration element registers the custom handler by class name and maps the .SampleAsync file name extension to that handler.

|  |
| --- |
| **ms227433.alert_note(en-us,VS.90).gifNote:** |
| Because you are registering a custom file name extension, you register the handler in both the **handlers** section and the **httpHandlers** section. In Classic mode, for backward compatibility, the handler is specified as an ISAPI module by using the **modules** attribute. The path of the ASP.NET ISAPI dll is specified by using the **scriptProcessor** attribute. The **name** attribute is required in the **handlers** section. |

**To register the handler in IIS 7.0 running in Integrated mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl41other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl41other');)

<configuration>

<system.webServer>

<handlers>

<add verb="\*" path="\*.SampleAsync"

name="HelloWorldAsyncHandler"

type="HelloWorldAsyncHandler"/>

</handlers>

</system.webServer>

</configuration>

The configuration element registers the custom handler by class name and maps the .SampleAsync file name extension to that handler.

|  |
| --- |
| **ms227433.alert_note(en-us,VS.90).gifNote:** |
| The registration is in the **handlers** section, but not in the **httpHandlers** section. The **name** attribute is required. |

http://i.msdn.microsoft.com/Global/Images/clear.gif Testing the Custom HTTP Handler

After you have created and registered the custom HTTP handler, you can test it.

**To test your custom HTTP handler**

* Browse to the application and enter a URL in the browser that ends in .SampleAsync.

The text defined in the HelloWorldAsyncHandler class is displayed.

**Walkthrough: Creating and Registering HTTP Handler Factories**

The [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) interface creates and manages HTTP handlers for processing requests. Therefore, you can create a class that implements the [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) interface, and then use that class as an HTTP handler.

Creating handlers in this manner can give you finer control over the processing of an HTTP request. It lets you map a URL to an HTTP handler factory that creates different handlers based on a set of conditions. For example, with an HTTP handler factory you can create a limited number of HTTP handler objects that access expensive or limited resources, such as database connections. You can then reuse those handler objects in future requests.

In this walkthrough, you will create an HTTP handler factory that creates two handlers for resources that are identified with the extension .sample. One handler works with resources during an HTTP **GET** request, and the other handler works with HTTP **POST** requests. The first handler is an instance of the handler described in [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx). The second handler is an instance of the handler described in [Walkthrough: Creating an Asynchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms227433.aspx).

Tasks illustrated in this walkthrough include the following:

* How to create the code for an HTTP handler factory class.
* How to register the handler factory in the Web.config file and map the .sample file name extension to it.
* How to map the .sample file name extension to ASP.NET In Internet Information Services (IIS).

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

In order to complete this walkthrough, you will need:

* Visual Studio or Visual Web Developer.
* An ASP.NET Web site that you can run by using IIS.
* IIS 6.0 or IIS 7.0.

http://i.msdn.microsoft.com/Global/Images/clear.gif Creating a Custom HTTP Handler Factory

To begin, you will create a handler factory class.

**To create an HTTP handler factory class**

1. If the ASP.NET Web site does not already have an App\_Code folder, create one under the root of the site.
2. In the App\_Code directory, create a class named HelloWorldHandler.
3. Add the following code to the class file.

Visual Basic

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl13VisualBasic');)

Imports System

Imports System.Web

Class HandlerFactory

Implements IHttpHandlerFactory

Public Function GetHandler(ByVal context As HttpContext, \_

ByVal requestType As String, ByVal url As [String],\_

ByVal pathTranslated As [String]) As IHttpHandler \_

Implements IHttpHandlerFactory.GetHandler

Dim handlerToReturn As IHttpHandler

Dim requestType as String = \_

context.Request.RequestType.ToLower()

If "get" = requestType Then

handlerToReturn = New HelloWorldHandler()

Else

If "post" = requestType Then

handlerToReturn = New HelloWorldAsyncHandler()

Else

handlerToReturn = Nothing

End If

End If

Return handlerToReturn

End Function

Public Sub ReleaseHandler(ByVal handler As IHttpHandler) \_

Implements IHttpHandlerFactory.ReleaseHandler

End Sub

Public ReadOnly Property IsReusable() As Boolean

Get

Return False

End Get

End Property

End Class

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl14CSharp');)

using System;

using System.Web;

class HandlerFactory : IHttpHandlerFactory

{

public IHttpHandler GetHandler(HttpContext context,

string requestType, String url, String pathTranslated)

{

IHttpHandler handlerToReturn;

if ("get" == context.Request.RequestType.ToLower())

{

handlerToReturn = new HelloWorldHandler();

}

else if ("post" == context.Request.RequestType.ToLower())

{

handlerToReturn = new HelloWorldAsyncHandler();

}

else

{

handlerToReturn = null;

}

return handlerToReturn;

}

public void ReleaseHandler(IHttpHandler handler)

{

}

public bool IsReusable

{

get

{

return false;

}

}

}

The code implements the [GetHandler](http://msdn.microsoft.com/en-us/library/system.windows.forms.design.eventhandlerservice.gethandler.aspx) method of the [IHttpHandlerFactory](http://msdn.microsoft.com/en-us/library/system.web.ihttphandlerfactory.aspx) interface. If the request is a **GET** request, the method returns the synchronous handler interface. If the request is a **POST** request, it returns the asynchronous handler interface.

http://i.msdn.microsoft.com/Global/Images/clear.gif Creating Custom HTTP Handlers

The custom HTTP handler factory returns either the synchronous handler described in [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx), or the asynchronous handler described in [Walkthrough: Creating an Asynchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms227433.aspx). You must create both the synchronous HelloWorldHandler class and the asynchronous HelloWorldAsyncHandler class in order for the custom HTTP handler factory to be able to return those handlers.

**To create the HelloWorldHandler and the HelloWorldAsyncHandler classes**

1. In the Web site's App\_Code directory, create a class named HelloWorldHandler.
2. Add the code from [Walkthrough: Creating a Synchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms228090.aspx) to the class file.
3. In the Web site's App\_Code directory, create a class named HelloWorldAsyncHandler.
4. Add the code from [Walkthrough: Creating an Asynchronous HTTP Handler](http://msdn.microsoft.com/en-us/library/ms227433.aspx) to the class file.

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler Factory in IIS 6.0

After you have created the custom HTTP handler factory class, you must register it in the application's Web.config file. This enables ASP.NET to use the handler factory class to service requests made to resources with the .sample file name extension.

There are different procedures for registering the handler, depending on whether you are working with IIS 6.0 or IIS 7.0. This section describes how to register a handler in IIS 6.0. The next section describes how to register a handler in IIS 7.0.

**To register the handler factory in IIS 6.0**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl31other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl31other');)

<configuration>

<system.web>

<httpHandlers>

<add verb="GET,POST" path="\*.sample"

type="HandlerFactory" />

</httpHandlers>

</system.web>

</configuration>

The code registers the handler factory with the class name and the handler name of HandlerFactory.

1. Register an application extension mapping for the .sample file name extension by using IIS Manager. For more information, see [How to: Configure an HTTP Handler Extension in IIS](http://msdn.microsoft.com/en-us/library/bb515343.aspx).

http://i.msdn.microsoft.com/Global/Images/clear.gif Registering the Custom HTTP Handler Factory in IIS 7.0

In IIS 7.0, an application can run in either Classic or Integrated mode. In Classic mode, requests are processed much the same way as they are in IIS 6.0. In Integrated mode, IIS 7.0 manages requests by using a pipeline that enables it to share requests, modules, and other features with ASP.NET.

For IIS 7.0, you register the handler factory either in the Web.config file or in IIS Manager. Because administration is centralized in IIS 7.0, changes in an application's Web.config file are reflected in the IIS Manager interface for the application and vice versa. In the procedures that follow, the handlers are registered in the Web.config file.

**To register the handler factory in IIS 7.0 running in Classic mode**

1. If the Web site does not already have a Web.config file, create one.
2. Add the following highlighted element to the Web.config file.

|  |
| --- |
| **ms227439.alert_note(en-us,VS.90).gifNote:** |
| Substitute the correct path for the aspnet\_isapi.dll file. The .dll file is in the folder where the .NET Framework is installed. By default this is C:\WINDOWS\Microsoft.NET\Framework\version. |

1. [[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl42other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl42other');)
2. <configuration>
3. <system.webServer>
4. <handlers>
5. <add verb="GET,POST" path="\*.sample"
6. name="HandlerFactory"
7. type="HandlerFactory"
8. modules="IsapiModule"/>
9. scriptProcessor="%path%\aspnet\_isapi.dll"
10. </handlers>
11. </system.webServer>
12. </configuration>
13. The configuration element registers the custom handler factory by class name and maps the .sample file name extension to that handler.

|  |
| --- |
| **ms227439.alert_note(en-us,VS.90).gifNote:** |
| Because you are registering a custom file name extension, you register the handler in both the **handlers** section and the **httpHandlers** section. In Classic mode, for backward compatibility, the handler is specified as an ISAPI module by using the **modules** attribute. The path of the ASP.NET ISAPI dll is specified by using the **scriptProcessor** attribute. The **name** attribute is required in the **handlers** section. |

**To register the handler factory in IIS 7.0 running in Integrated mode**

1. If the Web site does not already have a Web.config file, create one under the root of the site.
2. Add the following highlighted element to the Web.config file.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl44other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl44other');)

<configuration>

<system.webServer>

<handlers>

<add verb="GET,POST" path="\*.sample"

name="HandlerFactory"

type="HandlerFactory"/>

</handlers>

</system.webServer>

</configuration>

The configuration element registers the custom handler factory by class name and maps the .sample file name extension to that handler.

|  |
| --- |
| **ms227439.alert_note(en-us,VS.90).gifNote:** |
| The registration is done in the **handlers** section, but not in the **httpHandlers** section. The **name** attribute is required. |

**Testing the Custom HTTP Handler Factory**

After you have created and registered your custom HTTP handler factory, you can test it.

**To test the custom HTTP handler factory**

1. Create an HTML page (with a file name extension of .htm) in your application.
2. In the **body** section of the page, add the following markup.

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl46other');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl46other');)

<form action="Sample.sample" method="get">

<input type="submit" value="Submit to Sample.sample via Get" />

</form>

<br />

<form action="Sample.sample" method="post">

<input type="submit" value="Submit to Sample.sample via Post" />

</form>

1. Request the HTML page in a browser.
2. Click one of the buttons.

When you click the first button, the HTTP handler factory responds to the request by creating and calling a synchronous HTTP handler. When you click the second button, the HTTP handler factory responds to the request by creating and calling an asynchronous HTTP handler.