**Introduction to HTTP Modules**

An HTTP module is an assembly that is called on every request 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 therefore give you the opportunity to examine incoming requests and take action based on the request. They also give you the opportunity to examine the outbound response and modify it.

ASP.NET HTTP modules are similar to ISAPI filters in that they run for all requests. However, they are written in managed code and are fully integrated with the life cycle of an ASP.NET application.

Typical uses for HTTP modules include:

* Security. Because you can examine incoming requests, your HTTP module can perform custom authentication or other security checks before the requested page, XML Web service, or handler is called.
* Statistics and logging. Because HTTP modules are called on every request, you can gather request statistics and logging information in a centralized module, rather than in individual pages.
* Custom headers or footers. Because you can modify the outbound response, you can inject content such as custom header information into every page or XML Web service response.

ASP.NET uses modules to implement various application features, including 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](http://msdn.microsoft.com/en-us/library/ms178473(VS.85).aspx).

|  |
| --- |
| **NoteNote** |
| HTTP modules differ from HTTP handlers. HTTP modules are called for all requests and responses, whereas HTTP handlers run only in response to specific requests. For more information, see [Introduction to HTTP Handlers](http://msdn.microsoft.com/en-us/library/ms227675(VS.85).aspx). |

**How HTTP Modules Work**

You register a custom HTTP modules in your application's Web.config file. When ASP.NET creates an instance of the [HttpApplication](http://msdn.microsoft.com/en-us/library/system.web.httpapplication(VS.85).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(VS.85).aspx) method is called and the module initializes itself. For more information, see [ASP.NET Application Life Cycle Overview](http://msdn.microsoft.com/en-us/library/ms178473(VS.85).aspx).

In a module's **Init** method, you can subscribe to various application events such as [BeginRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.beginrequest(VS.85).aspx) or [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest(VS.85).aspx) by binding the events to methods you have created in the module. When these events are raised, the appropriate method in your module is called and the module can perform whatever logic is required, such as an authentication check 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(VS.85).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 your module includes an authentication check, the module might check for credentials and redirect to a login or error page if the credentials were not correct. Otherwise, when the module's event handler has finished running, ASP.NET calls the next process in the pipeline, which 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 in that they are encapsulated and can be created once and used in many different applications. By adding them to the Global Assembly Cache (GAC) 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(VS.85).aspx).

However, the advantage to using the Global.asax file is that you can place code in other registered module events such as **Session\_Start** and **Session\_End** methods. 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 need to create code that depends on application events and you either wish to reuse the module in other applications or you don't want to place complex code in the Global.asax file. You should place code in the Global.asax file whenever you need to create code that depends on application events and you do not need to reuse it across applications, or when you need to subscribe to events such as **Session\_Start** that are not available to modules.

**Creating an HTTP Module**

You can create a custom HTTP module by creating a class that implements the [IHttpModule](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule(VS.85).aspx) interface and then registering it in the Web.config file. The general process for writing an HTTP module is:

* Create a class that implements the **IHttpModule** [frlrfSystemWebIHttpModuleClassTopic](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule(VS.85).aspx) interface.
* Write a handler for the **Init**[frlrfSystemWebIHttpModuleClassInitTopic](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.init(VS.85).aspx) method. Your init method should initialize your module and subscribe to any application events you need. For example, you might subscribe to the **EndRequest** event if you want to append something to responses, or you might subscribe to the [AuthenticateRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.authenticaterequest(VS.85).aspx) event if you wish to perform custom authentication logic. For more information on application events, see [ASP.NET Application Life Cycle Overview](http://msdn.microsoft.com/en-us/library/ms178473(VS.85).aspx).
* Write code for the events you have subscribed to.
* Optionally implement the [Dispose](http://msdn.microsoft.com/en-us/library/system.web.ihttpmodule.dispose(VS.85).aspx) method if your module requires cleanup.
* Register the module in the Web.config file.

**How to: Create Custom HTTP Modules**

The custom HTTP module described in this topic illustrates the basic functionality of an HTTP module. The module is called in response to two events: the [BeginRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.beginrequest(VS.85).aspx) event and the [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.httpapplication.endrequest(VS.85).aspx) event. This enables the module to run before and after a page request is processed. In this case, the module adds a message to the requested ASP.NET Web page at the beginning of any HTTP request and another message after the request has been processed.

|  |
| --- |
| **NoteNote** |
| The **BeginRequest** and **EndRequest** events are only two of the events that occur during page processing. For more information about the events raised while processing a page, see [Server Event Handling in ASP.NET Web Pages](http://msdn.microsoft.com/en-us/library/xax2hw3x(VS.85).aspx). |

Each handler is written as a private method of the module. When the registered events are raised, ASP.NET calls the appropriate handler method in the module, which writes information to the ASP.NET Web page.

### To create a custom HTTP module class

1. If your Web site does not already have an App\_Code folder, create one under the root of the site.
2. Create a class file named HelloWorldModule.vb (for Visual Basic) or HelloWorldModule.cs (for C#) in the App\_Code directory.

|  |
| --- |
| **NoteNote** |
| Alternatively, you can compile the HelloWorldModule class into a library and place the resulting .dll file in the Web application's Bin directory. |

1. Add the following code to your class file:

Visual Basic

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04VisualBasic');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl04VisualBasic');)

Imports Microsoft.VisualBasic

Public Class HelloWorldModule

Implements IHttpModule

Public ReadOnly Property ModuleName() As [String]

Get

Return "HelloWorldModule"

End Get

End Property

' In the Init function, register for HttpApplication

' events by adding your handlers.

Public Sub Init(ByVal application As HttpApplication) \_

Implements IHttpModule.Init

AddHandler application.BeginRequest, \_

AddressOf Me.Application\_BeginRequest

AddHandler application.EndRequest, \_

AddressOf Me.Application\_EndRequest

End Sub

Private Sub Application\_BeginRequest(ByVal source As Object, \_

ByVal e As EventArgs)

' Create HttpApplication and HttpContext objects to access

' request and response properties.

Dim application As HttpApplication = CType(source, \_

HttpApplication)

Dim context As HttpContext = application.Context

context.Response.Write \_

("<h1><font color=red>HelloWorldModule: " & \_

"Beginning of Request</font></h1><hr>")

End Sub

Private Sub Application\_EndRequest(ByVal source As Object, \_

ByVal e As EventArgs)

Dim application As HttpApplication = CType(source, \_

HttpApplication)

Dim context As HttpContext = application.Context

context.Response.Write \_

("<hr><h1><font color=red>HelloWorldModule: " & \_

"End of Request</font></h1>")

End Sub

Public Sub Dispose() Implements IHttpModule.Dispose

End Sub

End Class

C#

[[http://i.msdn.microsoft.com/Global/Images/clear.gif](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05CSharp');)Copy Code](javascript:CopyCode('ctl00_rs1_mainContentContainer_ctl05CSharp');)

public class HelloWorldModule : IHttpModule

{

public HelloWorldModule()

{

}

public String ModuleName

{

get { return "HelloWorldModule"; }

}

// In the Init function, register for HttpApplication

// events by adding your handlers.

public void Init(HttpApplication application)

{

application.BeginRequest +=

(new EventHandler(this.Application\_BeginRequest));

application.EndRequest +=

(new EventHandler(this.Application\_EndRequest));

}

private void Application\_BeginRequest(Object source,

EventArgs e)

{

// Create HttpApplication and HttpContext objects to access

// request and response properties.

HttpApplication application = (HttpApplication)source;

HttpContext context = application.Context;

context.Response.Write("<h1><font color=red>

HelloWorldModule: Beginning of Request

</font></h1><hr>");

}

private void Application\_EndRequest(Object source, EventArgs e)

{

HttpApplication application = (HttpApplication)source;

HttpContext context = application.Context;

context.Response.Write("<hr><h1><font color=red>

HelloWorldModule: End of Request</font></h1>");

}

public void Dispose()

{

}

}

# **Registering the HTTP Module**

When you have finished creating the HelloWorldModule class, you can register the module by creating an entry in the Web.config file.

### To register the module in the Web.config file

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

[[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>

<httpModules>

<add name="HelloWorldModule" type="HelloWorldModule"/>

</httpModules>

</system.web>

</configuration>

The code registers the module with the class name and the module name of HelloWorldModule.

# **Testing the Custom HTTP Module**

After you have created and registered your custom HTTP module you can test it.

### To test the custom HTTP module

1. Create a Default.aspx page in your application.
2. Request the Default.aspx page in a browser.

The HTTP module appends a string to the beginning and end of the response. The module will automatically run on any request to a file whose extension is assigned to ASP.NET.