From the August 2002 issue of MSDN Magazine

msdn magazine

ASP.NET

Intercept, Monitor, and Modify Web Requests with HTTP Filters in ISAPI and ASP.NET

Panos Kougiouris

This article assumes you're familiar with C, C#, IIS, and ISAPI

Level of Difficulty 1 2 3

Download the code for this article: HTTPFilters.exe (1,169 KB)

SUMMARY There can be many reasons to reroute incoming Web requests. For instance, sometimes it's necessary to redirect a browser to a page based on user criteria without passing long lists of parameters in the URL. In the past, the only way to intercept such page requests and send them elsewhere was with ISAPI. Now, in ASP.NET, the IHttpModule interface provides notification of server requests, and lets you easily reroute them based on criteria other than browser type or version. Here the author demonstrates the use of IHttpModule for interception and explains the use of ISAPI filters for anyone who isn't yet using ASP.NET.

ASP.NET has many advantages over ASP. Strongly typed languages, the code-behind feature, and Web controls are just a few. One of my favorite features of ASP.NET is the ability to filter HTTP calls easily. This enables a Web developer to intercept every call and Web server response and inject code before completing the request. Securing a site and logging throughput are examples of situations in which HTTP filters are helpful.

Before ASP.NET, the only way to filter HTTP calls was through ISAPI filters, which are difficult to write. Deployment is complicated since the Microsoft® Internet Information Services (IIS) metabase needs to be modified. And development and deployment of filters requires stopping and starting the IIS process. In most cases, ASP.NET resolves all of these problems.

In this article I will present a solution to a common URL rewrite problem using HTTP filtering. First I will demonstrate how to solve the problem in ASP 3.0; then I will solve the same problem using ASP.NET.

Rewriting URLs

Assume a company runs a fictional Web-based payroll site named http://www.bestPayroll.com, which is used by several customer companies. All the companies use the same Web application, but the application is customized for each. The key requirement is that each customer land directly in his or her own view of the site. The ASP programmers at the payroll company decided to give each customer company a URL that contains a parameter with the company name. For instance, the URL for the fictitional MSDN® Magazine report page http://www.bestPayroll.com/report.asp?company=msdnmag.

When the developers showed the proposed solution to the company's business analyst, she didn't like it. The experienced analyst, with several dot-com ventures under her belt, recalled that in her last venture the switch from ASP to ASP.NET was rather painful. Everybody in engineering had assured her that there would be no visible change for the customer since it was just a change in technology. Yet at the last minute she discovered that all the URLs had changed from .asp to .aspx and the customers were complaining that their bookmarks stopped working. So she's not interested in more .asp, .aspx, .jsp, or other techie jargon. In addition, she wants to get rid of the question mark at the end of the URL that the developers recommended. She suggests the developers use a dif- ferent syntax altogether for the URL: http://www.yourPayroll.com/msdnmag/report.

After considering several ugly and not very scalable solutions using one directory for each company, the developers realize that HTTP filtering is the best solution. As the request comes in, the virtual path can contain a string that looks like /msdnmag/report, but a filter can convert this to /report.asp or

/report.aspx. The only remaining issue is how to communicate the company name to the template each time a URL is requested.

Multiple Directories and Default Documents

One way to solve this problem in both ASP and ASP.NET is to take advantage of IIS default documents. A Web site or virtual directory can specify a number of file names to be used as default documents. When a request arrives at the server, the server searches the directory to find whether a file with a default document name exists. If it does exist, it returns that document.

IIS uses default.htm, default.asp, and default.aspx as default document names. Using the IIS metabase's inheritance feature, the list can be modified for the whole server, a site inside the server, or even a virtual directory. A user can modify the list from either the IIS Microsoft Management Console (MMC) GUI or programmatically using the IIS Administration APIs.

Using default documents means that for every page you display, you have to create a folder and a page with the default name. For instance, for the hypothetical www.bestPayroll.com scenario you would have to create a folder called /report and then add a default.aspx page inside that folder.

When I first conceived this solution I felt I might be abusing IIS. After all, this feature appeared to be in place only for the root page of a Web site, and here I was trying to use it for every page. Is there any hidden penalty? Well, if you look at the log of your Web server after you implement this solution you'll see something like the following entries. Let's say you tried the URL http://bestpayroll.com/report:

```
23:49:55 192.168.27.1 GET /report/ 302
23:49:55 192.168.27.1 GET /report/Default.aspx 200
```

The problem is that there are two entries in the log file for each request. What happened? When the first request arrives at the Web server, the server responds to the client with the 302 message. Status messages in the 300s belong to the Redirection class. They inform the browser that the resource has moved and redirect the browser to the new location. The 302 status code tells the browser that the resource has moved temporarily. Along with the 302 status, the server sends the new URL. The HTTP payload shows the detail:

MSDN Magazine Blog

MSDN Magazine June Issue Preview
Monday morning the June issue of MSDN Magazine will go live on our Web site. Here's what you can expect to find in the magazine. Windows 8 figures pr... More...
Friday, May 31

MSDN Magazine May Issue Preview
Tomorrow afternoon the May issue of MSDN Magazine will go live on our Web site. Here's what you can look forward to in the upcoming issue. Craig Shoe... More...
Tuesday, Apr 30

More MSDN Magazine Blog entries >

Current Issue

Browse All MSDN Magazines

Subscribe to MSDN Flash newsletter

Receive the MSDN Flash e-mail newsletter every other week, with news and information

personalized to your interests and areas of focus.

```
HTTP/1.1 302 Object Moved
Location: http://192.168.27.130/bestpayroll/report/
Server: Microsoft-IIS/5.1
Content-Type: text/html
Content-Length: 164
```

The browser then tries the new URL, the one with the additional slash at the end. The Web server responds with the default document. It would be better to avoid this redirection for every page. I knew there must be a better solution.

Using an HTTP Filter in ASP.NET

A better solution is to intercept the incoming URL early on and rewrite it. Before ASP.NET, the only way to intercept HTTP calls was through ISAPI. ASP.NET introduces a new set of APIs around the IHttpModule interface. Objects that implement this interface can ask the Web application to be notified whenever an HTTP request is being processed.

Here's how it works. The developer authors a class that implements the IHttpModule interface. The IHttpModule interface defines only two methods:

```
interface IHTTPModule {
   void Init(HTTPApplication context);
   void Dispose();
}
```

The Init method is called once—when an application is initialized. After it's initialized, the HTTPApplication is passed as an argument to the Init method. It is expected that the method will attach handlers to one or more events that it cares about. The Dispose method is called once—when the application terminates. You should keep in mind that the application terminates and reinitializes not only when the application or IIS restarts, but also when the Web.config configuration file or a dependent assembly is modified in any way.

The code in Figure 1 is a minimal HTTP module. It is invoked when a request first arrives. It logs an event with the Windows® system logging facility and returns.

To activate the filter on a Web app, you need to compile the file into an assembly and make the assembly accessible to the app. Then you need to modify the file Web.config to add the module:

```
<configuration>
   <system.web>
      •••
      <httpModules>

   <add type="Panos.Test.HTTPModule,PanosTest" name="test" />
      </httpModules>
      •••
   </system.web>
•••
</configuration>
```

In addition to the BeginRequest message, the ASP.NET runtime provides a number of other events on the application object. An ASP.NET HTTP filter can set up handlers to any of these events. Figure 2 lists all the events and the order in which they are invoked by the runtime as a request progresses.

In case of multiple modules, the same events are fired in the order the HttpModules appear in the Web configuration file. For instance, assume an application has two modules, A and B, and both handle the BeginRequest and AuthenticateRequest events. Also assume that module A is listed first in the HttpModules section of the configuration file. Then the four events will be handled in the following order:

- Module A handles BeginRequest
- Module B handles BeginRequest
- Module A handles AuthenticateRequest
- Module B handles AuthenticateRequest

Rewriting URLs with ASP.NET HttpModules

When I started researching HTTP filtering in ASP.NET, I was certain that there would be a way to solve the URL rewriting problem using ASP.NET HTTP filtering. It turns out that generally this is not possible. This is not really a limitation in the common language runtime (CLR) or ASP.NET; it's due to an limitation in IIS 5.0.

IIS 5.0, like previous versions of IIS, relies on file extensions for extensibility. Take a look at **Figure 3**. This view of the IIS metabase is what links a particular extension with a runtime system. If you take a close look at the mappings you'll see that all the ASP extensions are associated with asp.dll, while all the ASP.NET extensions are handled by aspnet_isapi.dll. In fact, if you write an ASP.NET handler that handles files with the extension .foo you need to make sure that you change your Web.config file to map .foo files to your handler, and you also must configure IIS to have ASP.NET handle files with the extension .foo.

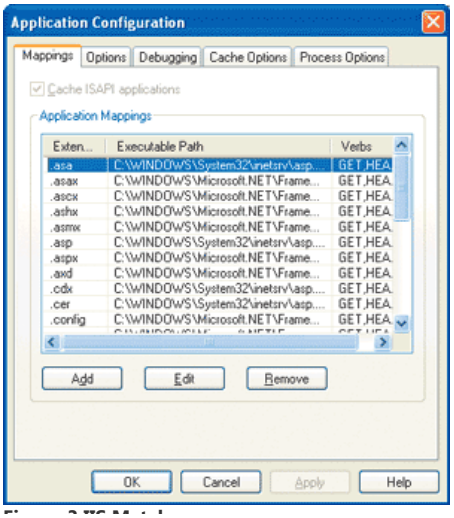

Figure 3 IIS Metabase

What does IIS do with paths that have no extension, like /bestpayroll/msdnmag/report? IIS checks to see if the virtual path maps to an existing physical directory. If it does, it looks up the list of the default documents to find a match with a name in the list of the default documents. If it finds one, it redirects the client to this file as I explained earlier. Otherwise it simply reports an HTTP 404—File not found error.

Now armed with all this knowledge, the Web developer should be able to get back to his business analyst and explain to her that in order to complete the project they really need a file extension such as /msdnmag/report.bestpayroll instead of /msdnmag/report.

If the idea is accepted, then ASP.NET can help with the IHttpHandler interface. Objects that implement this interface can handle HTTP requests directly. In fact, a Web form is compiled to a class that inherits from the code-behind class, which in turn inherits from the System.Web.UI.Page class that, itself in turn, implements the IHttpHandler interface.

The key IHttpHandler method to implement is ProcessRequest. This method takes an argument of type HttpContext. The argument provides access to all the useful context objects like Request, Response, and so on. In my case, I want the HttpHandler to parse the virtual path and redirect the request to another page. I prefer to do the redirection on the server side to avoid a round-trip back to the client. I use the Server.Transfer method, which is similar to its ASP equivalent, to accomplish this.

The only other point I need to mention is how the HttpHandler class communicates the company name to the ASP.NET form. The HttpHandler finds the name by parsing the virtual path of the URL. In my example, I used the HttpContext.Items collection. The ASP.NET page can collect the company name using the <%=Context.Items["company"]%> expression. Figure 4 shows the code that implements the redirection handler.

After you implement the class and package it into an assembly, you need to configure the Web site to invoke this handler whenever a request arrives. You do this in two places. In your Web.config file, you need to add an entry that associates URLs containing the .bestPayroll extension with your handler. Here's how I would do this for my particular example:

```
<configuration>
  <system.web>
    •••
    <httpHandlers>
     •••
      <add verb="*" path="*.bestpayroll"
        type="Panos.Test.HttpHandler,Panos.Test" />
    </httpHandlers>
  <.system.web>
</configuration>
```

In addition, you need to set IIS to forward requests for files with the .bestpayroll extension to the ASP.NET runtime. You can do this either programmatically or through the IIS administration UI, as shown in **Figure 5**. For the the .NET runtime, the DLL is C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705\aspnet_isapi.dll. The ASP.NET runtime will vary in different versions of the .NET Framework and the safest way to find the right one is to see the DLL associated with .aspx files. Hopefully, in a future version of IIS that is more aware of .NET, you will not have to perform this last step at all.

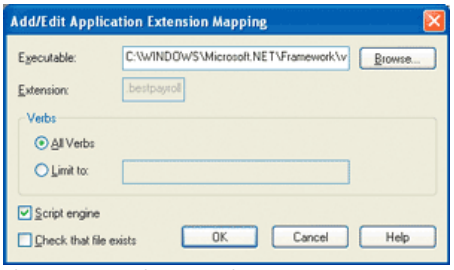

Figure 5 Extension Mapping

For a more detailed description of how to implement and configure an HTTP handler, please see HOW TO: Create an ASP.NET HTTP Handler Using Visual C# .NET.

URL Rewriting using ISAPI Filters

If you've reached this point, chances are you want the general solution, one that maps any arbitrary URL to your implementation ASP or ASPX files without the limitation of the file extension I've discussed. The only way I know to do this is outside the .NET Framework, using plain old ISAPI filters. You see, ISAPI filters are HTTP handlers native to IIS and they are given a chance to run very early, before any framework code in ASP.NET.

Conceptually, ISAPI filters are very similar to the ASP.NET HTTPModules I discussed earlier. A major difference is that ISAPI filters must be written in C or C++. There is plenty of literature on ISAPI filters so I will only touch on them briefly here (see ISAPI, and Tips and Tricks for ISAPI Programming for more information).

In practice, an ISAPI filter is a Windows DLL that provides the following two entry points:

```
DWORD WINAPI HttpFilterProc(
  PHTTP_FILTER_CONTEXT pfc,
  DWORD notificationType,
  LPVOID pvNotification
);

BOOL WINAPI GetFilterVersion(
  PHTTP_FILTER_VERSION pVer
);
```

The IIS runtime calls the GetFilterVersion method when the filter is initialized. The filter should use this opportunity to notify IIS about the events it cares about. Then, when HTTP requests arrive, IIS calls the HttpFilterProc once for every event for which the filter asked to be notified. If you want to make things a little bit easier you can use the MFC class CHttpFilter, which conveniently wraps the raw C API and routes the events to one or more virtual methods that you implement. The rest of the discussion will assume that you use the MFC-provided wrapper (see Writing Interactive Web Apps is a Piece of Cake with the New ISAPI Classes in MFC 4.1).

ISAPI Classes in MFC 4.1).

Figure 6 shows the C++ code that rewrites the URL and stores part of the virtual path in a pseudoheader. The event that you need to catch in order to rewrite the URL is OnPreprocHeaders. The trick here is that you can use the special header URL to read the original URL. Then, after rewriting the URL, you can use the SetHeader call to set the special header URL, forcing IIS to call your file. The code shown in Figure 6 changes a URL like /bestpayroll/msdnmag/ report2 to /bestpayroll/report2.aspx. Notice how the company name has been removed and the .aspx suffix has been added. Most of the ugly code deals with extracting the company name and then piecing together the rest of the URL for the request.

The last remaining task is to communicate either the original URL or the company name to the ASP.NET form. There is no clean way to communicate from an ISAPI filter to either ASP or ASP.NET. What I can do is add a new header. A header called companyname can be read from ASP.NET using the Request.ServerVariables collections, as shown here:

```
<form id="test" method="post" runat="server">
    The company is
    <b><%=Request.ServerVariables["HTTP_COMPANYNAME"]%></b>
</form>
```

Here I hardcoded the logic that rewrites URLs into the C++ code. In more realistic scenarios the logic should reside in a configuration file that is parsed by the filter during initialization.

Conclusion

Using the simple problem of rewriting a URL, I presented the different ways to filter HTTP calls. ASP.NET adds new ways to filter HTTP calls that take advantage of the CLR, and this is great in most cases. However, you might still need to do some C++ programming when you have to filter calls early on, before the ASP or ASP.NET runtime have a chance to run. I hope that future versions of IIS will allow administrators to designate the ISAPI extension that will handle a request based on the prefix of the virtual path.

For related articles see:
Active Server Pages+: ASP+ Improves Web App Deployment, Scalability, Security, and Reliability
HOW TO: Create an ASP.NET HTTP Module Using Visual C# .NET
For background information see:
Taking the Splash: Diving into ISAPI Programming
ISAPI Extensions: Filters
ASP.NET Settings Schema

Panos Kougiouris *has architected several Internet applications using Windows DNA, .NET, and other technologies. He can be reached at panos@acm.org.*