

Lab 10.2: Building an ISAPI Application (Echo)

Lab Overview

Objectives

After completing this lab, you will be able to:

- ♦ Use the ISAPI Extension Wizard to create an ISAPI application project with Microsoft Visual C++.
- ♦ Implement the project, including:
 - Add and manage parse map entries corresponding to request methods.
 - Add handler member functions corresponding to the parse map entries.
 - Implement the handlers and supporting functions and classes
- ♦ Install and test an ISAPI application.

Scenario

You can develop ISAPI applications to extend a Web site that uses an ISAPI-compliant Web server. These components flexibly extend the Web server by performing various programmed actions, then typically return a dynamically-created HTML page to the Web client.

Lab Setup

To develop the ISAPI component, this lab requires that you have Microsoft Visual C++ version 4.1 or later installed on your computer. To test this component, you must have administrative privileges on a computer running Windows NT and a Microsoft Web service, such as Internet Information Server or Peer Web Services.

This demonstration shows the solution to this lab.



Estimated time to complete this lab: **90 minutes**.

Exercises

The following exercises provide practice working with the concepts and techniques covered in this chapter.

Exercise 1: Creating the Project

In this exercise, you will use Microsoft Visual C++ to create the Echo ISAPI application project. You will select and edit various options in the ISAPI Extension Wizard dialog, as appropriate for the Echo application.

Exercise 2: Implementing Echo

In this exercise, you will implement the Echo ISAPI application. To accomplish this, you will manage the parse map for the project, add and implement basic versions of the handler member functions corresponding to request methods, and add and implement helper member functions.

Exercise 3: Testing Echo

In this exercise, you will install the basic version of the Echo ISAPI application on your Microsoft Web server and test it with Internet Explorer. Optionally, you will repeat the testing process with the supplied HTML page, EchoTest.htm.

Then, you will test the final version of Echo, and repeat the testing process. To accomplish this, you will need to uninstall the previous version of Echo.dll.

Exercise 1: Creating the Project

In this exercise, you will use Microsoft Visual C++ to create the Echo ISAPI application project. You will select and edit various options in the ISAPI Extension Wizard dialog, as appropriate for the Echo application.

The Echo ISAPI application echoes the request information from the client's HTTP request message. The application contains one method, `EchoRequest`, that enables the client to receive command help, HTTP header and body information, or a bad syntax error message. Echo is a simple developer's tool that compliments the `WebClient` utility.

► Create a new project for Echo

1. Start Microsoft Developer Studio. From the File menu, choose New.
2. In the New dialog, select Project Workspace, and choose OK. The New Project Workspace dialog is displayed.
3. Supply the following information in the New Project Workspace dialog:
 - Type - Select ISAPI Extension Wizard
 - Name - Enter Echo
 - Platform - Select Win32 (default)
 - Location - Enter or browse for `\<MID root directory>\Labs\Lab10.2\Echo`

Then, choose the Create button. The ISAPI Extension Wizard dialog, Step 1 of 1, is displayed.

4. In the ISAPI Extension Wizard dialog, make the following corrections and verifications:
 - The Generate A Filter Object check box should be cleared, and Generate A Server Extension Object should be selected.
 - Change the Extension Description to Echo ISAPI App.
 - Use the MFC libraries As A Shared DLL

When you are done, choose Finish.

5. A summary dialog box appears. After you review the information in it, choose OK to generate the new project.

► Use the Project Workspace to investigate Echo

1. Select the ClassView pane of the Project Workspace, if it is not already displayed. Expand all the branches of the Echo classes.
2. Double-click on the following entries to view the associated source code.
 - `CEchoExtension` – to view your `CHttpServer`-derived class declaration. Note the line towards the end of the file that declares the parse map.
 - `GetExtensionVersion` – to view the initialization member function that the Wizard generated for your ISAPI application.
 - `Default` – to view the example command handler generated by the Wizard.
 - `theExtension` – to view the single instantiation of your `CHttpServer`-derived class.
3. Select the ResourceView pane. Expand all the branches of the Echo resources.
4. Double-click on the following entries to view the associated Windows resource.
 - `String Table` has a single-string resource, `IDS_SERVER`, which is the string description used by `CEchoExtension::GetExtensionVersion` to register an extension description with the Web server. (You supplied this string when you ran the ISAPI Extension Wizard in the previous step.)
 - `VS_VERSION_INFO` is an editable program-version resource.

5. In the FileView pane, open Echo.cpp. Toward the top of the file, locate the parse map implementation. Note that the Wizard has placed an entry for the example handler, Default, into the map. This handles the invocation of the ISAPI extension when no method is supplied, because it uses the special macro `DEFAULT_PARSE_COMMAND`.
6. Open the file Echo.def. Note that it only exports the two entry points that every ISAPI application must have: the global C functions `HttpExtensionProc` and `GetExtensionVersion`.

➤ **Build the Echo project**

1. Build the application, targeting Win32 Debug. Echo should compile and link cleanly.
2. Close all the source windows except the ones for Echo.h and Echo.cpp. You will be editing these files in the subsequent exercises of this lab.

Exercise 2: Implementing Echo

In this exercise, you will implement the base Echo ISAPI application. To accomplish this, you will manage the parse map for the project, add and implement basic versions of the handler member functions corresponding to request methods, and add and implement helper member functions.

➤ **Edit the `CEchoExtension` class declaration**

To complete the Echo ISAPI application, you will add a new command handler, named `EchoRequest`, and three helper functions, which will be declared in a protected-implementation section.

1. Open the file Echo.h, if it is not already open. Locate the `Default` member function.
2. Immediately after the `Default` member function, add a new handler, named `EchoRequest`, that returns nothing, and takes two arguments: a pointer to the `CHttpServerContext` object and a `LPCTSTR`.
3. Just before the declaration of the parse map, add a comment that denotes an Implementation section. Afterward, add a Protected access specifier.
4. In this new section, add four new helper member functions named `EchoBody`, `EchoHead`, `BadSyntax`, and `EchoHelp`. All of these functions have the same signature: they return nothing and take a single argument, a pointer to the `CHttpServerContext` object.
5. Save your work in the file Echo.h.

➤ **Edit the parse map**

1. Open the file Echo.cpp, if it is not already open. Locate the parse map implementation near the top of the file.
2. Add parse map entries for the command handler, `EchoRequest`. The first entry identifies the new command handler and specifies its arguments:

```
ON_PARSE_COMMAND(EchoRequest, CEchoExtension, ITS_PSTR)
```

The second entry names the argument of the associated ISAPI invocation method and supplies a default text value of `FULL`.

```
ON_PARSE_COMMAND_PARAMS("option=full")
```

If the server's name is `WebSvr`, these two parse map entries specify an ISAPI method that can be invoked through one of the following URLs:

```
http://WebSvr/Scripts/Echo.dll?EchoRequest
http://WebSvr/Scripts/Echo.dll?EchoRequest&full
http://WebSvr/Scripts/Echo.dll?EchoRequest&option=full
```

➤ Implement the default handler

The default implementation for an ISAPI application, if present, usually performs the standard processing routine. However, since Echo is a developer's tool, you will code it to return a help page.

1. Locate the implementation of `CEchoExtension::Default`.
2. Replace the statements in this function that begin with `*pCtxt` with the following lines:

```
*pCtxt << "<P><H1>Help for Echo.dll</H1><P><HR>";
EchoHelp(pCtxt);
```

`EchoHelp` is a helper member function that you will add in a later step.

➤ Implement the EchoRequest command handler

In MFC, one of the initial responsibilities of the command handler is to distinguish the passed argument(s), if any, and determine appropriate processing based on those arguments.

1. Copy the implementation of `CEchoExtension::Default` and paste it immediately after the original.
2. Rename the new function `CEchoExtension::EchoRequest`. Add a second argument to the signature, `LPCTSTR pstrOption`.

3. Edit the output line as shown:

```
*pCtxt << "<P><H1>Echo.dll</H1><P><HR>";
```

4. Replace the call to `EchoHelp` with a nested if-else statement that does a case-insensitive comparison (`_stricmp`) of the handler's second argument and of these expected values:

String Match	Action(s)
"header"	Call <code>EchoHead</code> , passing along the pointer to the server context
"body"	Call <code>EchoBody</code> , passing along the pointer to the server context
"full"	Call <code>EchoHead</code> , and then <code>EchoBody</code>
<i>other</i>	Call <code>BadSyntax</code> , passing along the pointer to the server context

`EchoHead`, `EchoBody`, and `BadSyntax` are helper member functions that you will add during later steps.

➤ Implement the EchoHead helper function

Note: Because the Web server parses the HTTP header by the time an ISAPI application is invoked, the application cannot actually echo the original header request. Instead, it can only return pieces of the original header.

1. After the `EchoRequest` function, add a comment separator with the label `Implementation Member Functions`.
2. Add the function header and empty body for the `EchoHead` function. Copy the declaration from the file `Echo.h` to get started.
3. Inside the body, declare two local variables: an array of 1000 characters named `pstrBuffer`, and a `DWORD` variable named `dwSize`, initialized to the size of the array.

4. Write an appropriate HTML header line to the HTML stream:

```
*pCtxt << "<P><H3>Request Header Information</H3>";
```

5. Write a line that displays the request method. One way to obtain the request method is to search the extension control block member of the server context object for this information:

```
*pCtxt->m_pECB->lpszMethod
```

See the Help topics `CHttpServerContext Class Members` and `EXTENSION_CONTROL_BLOCK` for more information.

6. Write a line that displays the translated resource path, `lpszPathTranslated`.

- Next, use `CHttpServerContext::GetServerVariable` to obtain the query string of the request URL. Use `pstrBuffer` and the address of `dwSize` for the last two arguments. In the next statement, write this query string using `CHttpServerContext::`.

This member function represents another way of accessing client-request information.

- On the next line, reset the value of `dwSize` to the size of `pstrBuffer`, and assign the first character of `pstrBuffer` to a value of zero (the Null character).
- Use `CHttpServerContext::GetServerVariable` to obtain the remote address of the client. In the next statement, enter this information:

```
pCtxt->GetServerVariable("REMOTE_ADDR", pstrBuffer, &dwSize);
*pCtxt << "<BR>Client IP address: " << pstrBuffer;
```

➤ Implement the EchoBody helper function

- Copy the function implementation for `EchoHead`. Rename this version to `EchoBody`.
- Edit the HTML header line, as shown:

```
"<P><H3>Request Body Information</H3><BR>";
```

- Increase the size of the `pstrBuffer` array to be 4000.
- Call the function `CHttpServerContext::ReadClient` to copy the first 4000 bytes of the HTTP request body to the array `pstrBuffer`. Store the size of the body in the variable `dwSize`.
- Write the size of the HTTP body to the HTML stream. Before writing it, cast `dwSize` to be a long integer.
- If the read operation was successful, write this information to the HTML stream:

```
if (b == TRUE)
    *pCtxt << pstrBuffer;
```

➤ Implement the BadSyntax and EchoHelp helper functions

These two functions simply write textual information. `BadSyntax` displays an error message and then help information; `EchoHelp` displays only the help information.

- Implement `BadSyntax`.
 - Write a syntax error message to the HTML stream.
 - Call the `EchoHelp` function, passing along the server context.
- Implement `EchoHelp` to write syntax information to the HTML stream.

To save time, you may want to copy the implementations of these two functions from the source code in the `\Labs\Lab10.2\Solution` subdirectory of the *Mastering Internet Development* CD-ROM.

➤ Build the Echo DLL

Build the Echo project and correct any coding errors. For assistance, refer to the solution code in the subdirectory `\Labs\Lab10.2\Solution` of the *Mastering Internet Development* CD-ROM.

Exercise 3: Testing Echo

In this exercise, you will install the basic version of the Echo ISAPI application on your Microsoft Web server and test it with Internet Explorer. Optionally, you will repeat the testing process with the supplied HTML page, `EchoTest.htm`.

Next, you will test the final, fuller version of Echo, and repeat the testing process. To accomplish this, you will first need to uninstall the previous version of `Echo.dll`.

➤ Install the Echo ISAPI application on a Microsoft Web server

The first time you install an ISAPI application, installation is a simple matter of copying the DLL to the appropriate directory (see Step 2). However, once a client invokes a method from the application, Microsoft Web servers will load and retain the corresponding DLL. Therefore, the next time you update the DLL, you will need to perform all of the following steps.

1. To stop the Web service, forcing it to unload an ISAPI application, use one of the following techniques:
 - Reboot Windows to initialize the Web server.
 - Use the Internet Service Manager to stop the Web service.
 - Use the Web-based Service Administrator to stop the Web service.
 - Use the Services applet of the Control Panel to stop the Web service.
 - From the command line, issue the command **net stop W3Svc**. (Use **net start W3Svc** to restart the Web publishing service.)

Alternatively, you can force Microsoft Web servers to always reload extension DLLs each time they are used. To do this, adjust the registry setting to a value of zero at HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Services/W3SVC/Parameters/CacheExtensions. This should only be used for debugging, because it degrades the performance of the Web server.

2. Copy the file Echo.dll to the \Scripts subdirectory of your Microsoft Web server.
3. Restart the Web service, using one of the techniques listed in Step 1.

➤ Test Echo with Internet Explorer

1. To test the default handling of the Echo ISAPI application, enter the following into the address box of Internet Explorer:

```
http://<server-name>/Scripts/echo.dll?
```

This invokes the Default method of Echo, which returns the help page.

2. To test the full option of the EchoRequest method of the Echo ISAPI application, use one of the following URLs:

```
http://<server-name>/Scripts/echo.dll?EchoRequest
http://<server-name>/Scripts/echo.dll?EchoRequest&Full
http://<server-name>/Scripts/echo.dll?EchoRequest&option=Full
```

The first URL works because FULL was entered as the default value for EchoRequest's argument. This was set in the ON_PARSE_COMMAND_PARAMS macro of Echo's parse map.

Note that capitalization of the argument value Full is not a consideration, because a case-insensitive comparison was used in Echo.

3. To test the header option of the EchoRequest method, use one of the following URLs:

```
http://<server-name>/Scripts/echo.dll?EchoRequest&Header
http://<server-name>/Scripts/echo.dll?EchoRequest&option=Header
```

4. To test the body option of the EchoRequest method, use one of the following URLs:

```
http://<server-name>/Scripts/echo.dll?EchoRequest&Body
http://<server-name>/Scripts/echo.dll?EchoRequest&option=Body
```

These requests will always result in an empty body.

5. To test the bad syntax option of the EchoRequest method, use a bad argument, such as in the following URL:

```
http:// <server-name>/Scripts/echo.dll?EchoRequest&Foo7
```

➤ **Optional: Test Echo with a sample HTML page**

1. From the *Mastering Internet Development* CD-ROM, copy the file \Labs\Lab10.2\EchoTest.htm to your server's \WWWRoot subdirectory.
2. In Internet Explorer, enter the following URL:

`http://<server-name>/EchoTest.htm`

The Echo Test page should be displayed.

3. Click on the first button, Request Header, and note the page that is returned. Then, click the Back button on the Internet Explorer toolbar to return to the Echo Test page.
4. Repeat this process with the remaining buttons.
5. View the source code for the Echo Test page, EchoTest.htm.

➤ **Install and test the full version of Echo**

The fuller, final version of the Echo ISAPI application is just an enhancement of the version you have just coded. It produces better formatted output, and includes more information about the HTTP request header fields.

1. From the *Mastering Internet Development* CD-ROM, install the final version of the Echo ISAPI application (found in the subdirectory \Labs\Lab10.2\Final\Debug\) to your Web server's \Scripts subdirectory.

For more information, see the first section of this exercise.

2. As you did for the original, base version, test this ISAPI application as outlined in the previous sections of this exercise.
3. View the source code of the file Echo.cpp, found in the \Labs\Lab10.2\Final\ directory of this CD-ROM.