Lab 14.2: Building an ISAPI Application

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:
  - Adding and managing parse map entries corresponding to request methods.
  - Adding handler member functions corresponding to the parse map entries.
  - Implementing the handlers and supporting functions and classes
- Installing and testing an ISAPI application.

Prerequisites

Before attempting this lab, you should have completed Chapter 14 through the section titled "More Information on ISAPI applications."

Lab Setup

To see a demonstration for the solution to this lab, click this icon.

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.

There is no setup for this lab. The completed code for these exercises is in \Labs\C14\Lab02\Xxx, where Xxx is the exercise number.

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. Alternatively, the labs can be run on a Windows 95 machine running Personal Web Server.

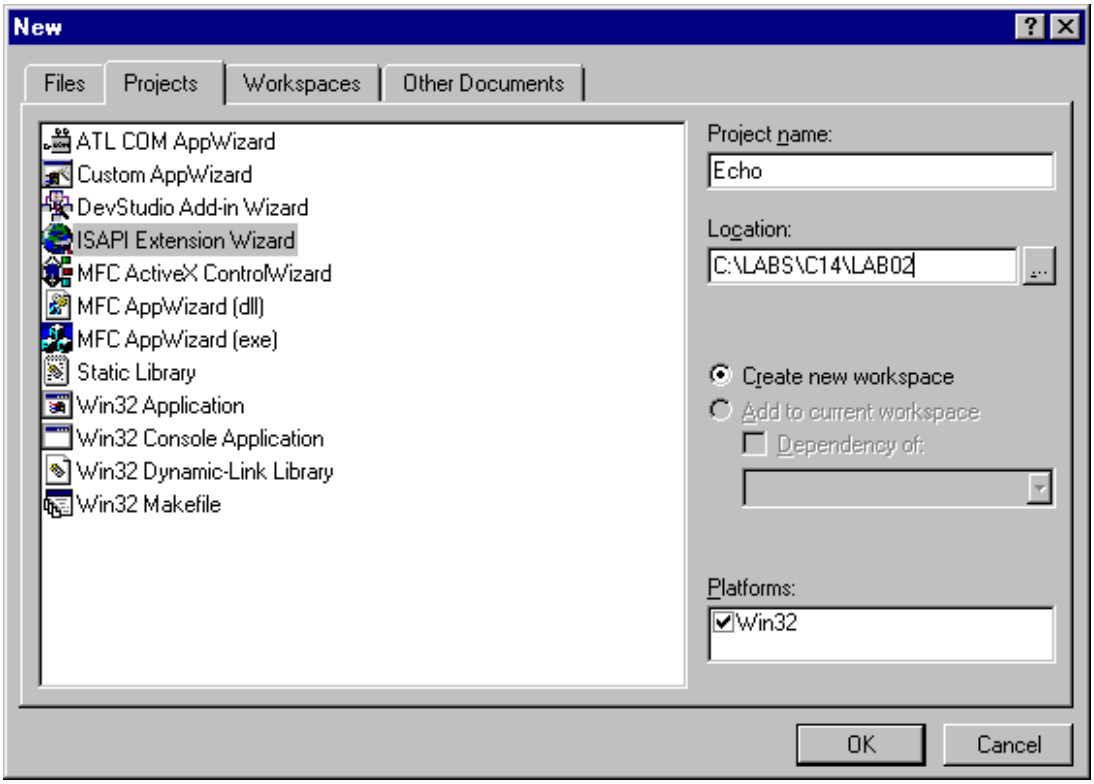
Exercise 1: Creating the Project

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.

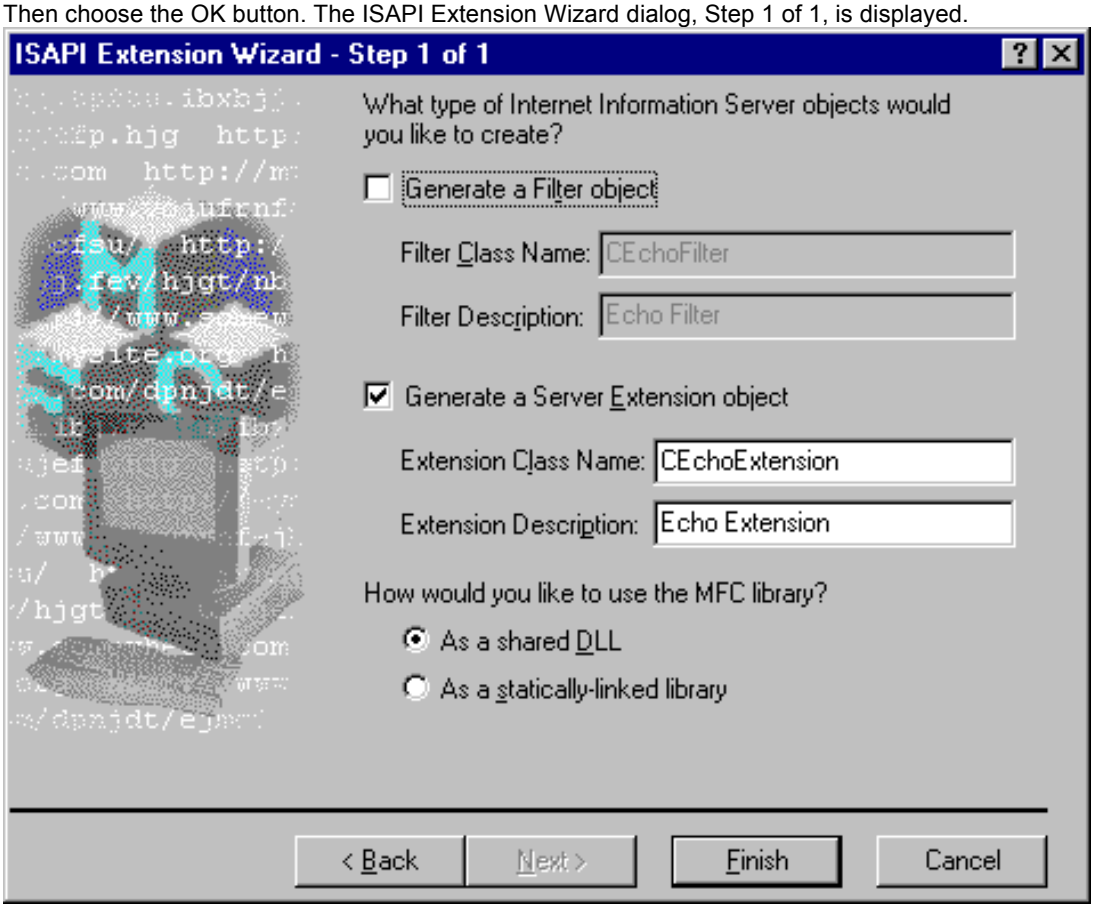
The Echo ISAPI application echoes the request information from the client's HTTP request message. The application contains one method, EchoRequest, that enables the client to receive command help, HTTP header and body information, or a bad syntax error message. Echo is a simple developer's tool that complements 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 the Projects tab.
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 /Labs/C14/Lab02.

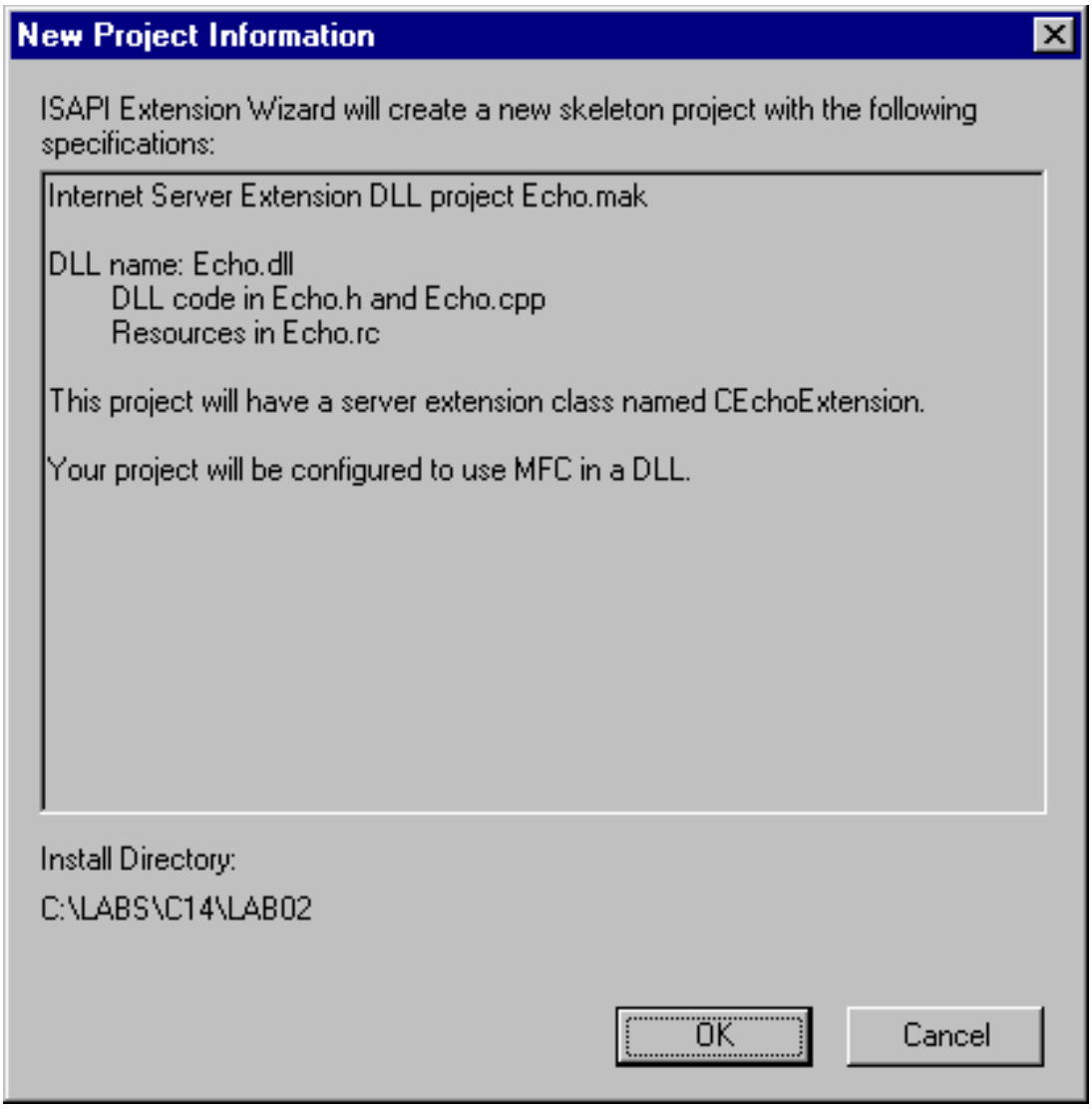


Then choose the OK button. The ISAPI Extension Wizard dialog, Step 1 of 1, is displayed.



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 the following entries to view the associated source code.
   - **CEchoExtension** – to view your CHttpServer-derived class declaration. Note the line toward 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 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 exports only 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 edit these files in the subsequent exercises of this lab.

The completed code for this exercise is in /Labs/C14/Lab02/Ex01.

Exercise 2: Implementing the ISAPI Application

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. In Class View, right-click **CEchoExtension** and add the following public member function.

```
void EchoRequest(CHttpServerContext *pCtxt, LPCTSTR pstrOption)
```

2. Use the method described in Step 1 to add the following protected functions to **CEchoExtension**.

```
void EchoHelp(CHttpServerContext * pCtxt)
void BadSyntax(CHttpServerContext * pCtxt)
void EchoHead(CHttpServerContext * pCtxt)
void EchoBody(CHttpServerContext * pCtxt)
```

➢ **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 contents of the member function CEchoExtension::Default and paste them into CEchoExtension::Request.
2. Edit the output line as shown:

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

3. 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 EchoHead, passing along the pointer to the server context
"body"	Call EchoBody, passing along the pointer to the server context
"full"	Call EchoHead, and then EchoBody
other	Call BadSyntax, passing along the pointer to the server context

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

4. The completed code for the **EchoRequest** function looks like this:

```
void CEchoExtension::EchoRequest(CHttpServerContext *pCtxt, LPCTSTR
pstrOption)
{
    StartContent(pCtxt);
    WriteTitle(pCtxt);

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

    if ( _stricmp(pstrOption, "full") == 0)
    {
        EchoHead(pCtxt);
        *pCtxt << "<BR>";
        EchoBody(pCtxt);
    }
    else if ( _stricmp(pstrOption, "header") == 0)
    {
        EchoHead(pCtxt);
    }
    else if ( _stricmp(pstrOption, "body") == 0)
    {
        EchoBody(pCtxt);
    }
    else
        BadSyntax(pCtxt);

    EndContent(pCtxt);
}
```

➢ **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. In the **EchoHead** function, declare two local variables: an array of 1,000 characters named pstrBuffer, and a DWORD variable named dwSize, initialized to the size of the array.
2. Write an appropriate HTML header line to the HTML stream:

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

3. 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.
4. Write a line that displays the translated resource path, lpszPathTranslated.
5. 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.
6. 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).
7. 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;
   ```

8. The completed code for the **EchoHead** function should look like this:

   ```
   void CEchoExtension::EchoHead(CHttpServerContext * pCtxt)
   {
       char pstrBuffer[1000];
       DWORD dwSize = sizeof(pstrBuffer);

       *pCtxt << "<P><H3>Request Header Information</H3>";
       *pCtxt << "<BR>Request method: " << pCtxt->m_pECB->lpszMethod;
       *pCtxt << "<BR>Tranlsated path: " << pCtxt->m_pECB->lpszPathTranslated;

       pCtxt->GetServerVariable("QUERY_STRING", pstrBuffer, &dwSize);
       *pCtxt << "<BR>Query String: " << pstrBuffer;

       dwSize = sizeof(pstrBuffer);
       pstrBuffer[0] = 0;
       pCtxt->GetServerVariable("REMOTE_ADDR", pstrBuffer, &dwSize);
       *pCtxt << "<BR>Client IP address: " << pstrBuffer;
   }
   ```

➢ **Implement the EchoBody helper function**

1. Copy the code from the **EchoHead** function and insert it into **EchoBody**.
2. Edit the HTML header line, as shown:

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

3. Increase the size of the pstrBuffer array to 4,000.
4. Call the function **CHttpServerContext::ReadClient** to copy the first 4,000 bytes of the HTTP request body to the array pstrBuffer. Store the size of the body in the variable dwSize.

5. Write the size of the HTTP body to the HTML stream. Before writing it, cast dwSize to be a long integer.
6. If the read operation was successful, write this information to the HTML stream:

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

7. The completed code for the **EchoBody** function should look like this:

```
void CEchoExtension::EchoBody(CHttpServerContext * pCtxt)
{
    char pstrBuffer[4000];
    DWORD dwSize = sizeof(pstrBuffer);

    *pCtxt << "<P><H3>Request Body Information</H3><BR>";

    BOOL b = pCtxt->ReadClient(pstrBuffer, &dwSize);
    *pCtxt << "Body is " << (long int)dwSize << " bytes <BR>" ;
    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.

1. Implement **BadSyntax**.
   a. Write a syntax error message to the HTML stream.
   b. Call the **EchoHelp** function, passing along the server context.
2. Implement **EchoHelp** to write syntax information to the HTML stream.
3. The completed code for the **BadSyntax** and **EchoHelp** functions should look like this:

```
void CEchoExtension::BadSyntax(CHttpServerContext* pCtxt)
{
    *pCtxt << "<H3>Bad Syntax Error!</H3><P>";
    *pCtxt << "Proper syntax is as follows:";
    EchoHelp(pCtxt);
}

void CEchoExtension::EchoHelp(CHttpServerContext* pCtxt)
{
    *pCtxt << "<P><I>Supported method:</I>";
    *pCtxt << "<BR><B><TT>EchoRequest</TT></B> - currently the only method
supported."
           << " Dynamically creates an HTML page containing HTTP request
information.";
    *pCtxt << "<P><I>Supported arguments:</I>";
    *pCtxt << "<BR><B><TT>Full</TT></B> - echo back the entire HTTP request
message.";
    *pCtxt << "<BR><B><TT>Body</TT></B> - echo back only the HTTP request
message body.";
    *pCtxt << "<BR><B><TT>Header</TT></B> - echo back only the HTTP request
message head.";
}
```

To save time, you may want to copy the implementations of these two functions from the source code in the \Lab\C14\Lab02\Ex02.

➢ **Build the Echo DLL**

Build the Echo project and correct any coding errors. The completed code for this exercise is in \Labs\C14\Lab02\Ex02.

Exercise 3: Testing the ISAPI Application

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 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.)
   - If you are running Windows 95 and Personal Web Server, open the Personal Web Server applet from the Control Panel. Select the Startup tab and choose the Stop button. If the Personal Web Server icon appears in the taskbar for Windows 95, you can right-click the icon and select Properties instead of going through the Control Panel.

   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 be used only 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. Copy the file \Labs\C14\Lab02\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 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**

A more sophisticated version of the Echo ISAPI application is located in \Labs\C14\Lab02\Ex03. It produces better formatted output, and includes more information about the HTTP request header fields.

1. Build this version of Echo and copy Echo.Dll to your Web server's \Scripts subdirectory. For more information, see the first section of this exercise.
2. As you did for the original version, test this ISAPI application as outlined in the previous sections of this exercise.
3. View the source code of the file Echo.Cpp.

The completed code for this exercise is in \Labs\C14\Lab02\Ex03.