

26.1 Introduction: creating content dynamically

As you discovered in [Section 24.3.5](#), maintenance of a product list using HTML is a labor-intensive, mind-numbing experience—every change to price or inventory information involves searching through the HTML table cells, making the changes manually, and publishing the updated file to the web server.



Although it is possible to use the **Save as HTML** within ACCESS to automatically generate an HTML table containing product information, the resulting document is static. If a price is changed or a new product is added, the “save as” procedure must be repeated.

A better approach is to have the server create the product list dynamically, the moment it is requested. In this way, the most recent database information is used.

26.1.1 Scripting basics

Although there are a number of different ways to create documents dynamically on the server

using scripts (small programs), the process typically involves the following steps:

1. The user clicks on a link to a document. The user’s browser generates an HTTP request and sends it to the web server in the normal way (recall [Lesson 25](#)).
2. The server receives the request and retrieves the document file requested by the browser. If the file has a non-HTML extension, it is pre-processed before being transferred to the browser.
3. During preprocessing, the server checks for special non-HTML tags and programming code (e.g., VBSCRIPT, COLDFUSION, JAVA).
4. If special tags are found, the server processes them. For example, if the tags are VBSCRIPT, the VISUAL BASIC code they contain is executed by the ACTIVE SERVER PAGES (ASP) processor. If the tags contain COLDFUSION MARKUP LANGUAGE (CFML) queries, the SQL statements are executed by the COLDFUSION processor.
5. In most cases, the code within the special tags returns a result, such as a calculation or a database lookup. The server replaces the special tags with the result before the document is sent to the client.



Since all the work is done on the server, this process is called “server-side scripting”. After all the server-side processing is complete, the document sent to the browser is plain HTML. Users have no way of knowing that the document showing in their browser contains content that was generated dynamically in the instant it took the server to respond to the request.¹

26.1.2 A simple example

Consider the following VBSCRIPT embedded within standard HTML tags:

```
NL <HTML>
NL <HEAD>...</HEAD>
NL <BODY>
NL <P>The sum of 3 and 5 is:
   <%= 3+5 %></P>
NL </BODY>
NL </HTML>
```

The the code within the special script tags `<%... %>` is executed by the ASP processor and replaced by the result. Thus, the HTML that the browser receives is simply:

```
NL <HTML>
```

¹ “Instant” is a relative term. When the amount of processing done by the server is considerable (e.g., searching for a cheap flight on EXPEDIA.COM), the delay in returning a page to the user can also be considerable.

```
NL <HEAD>...</HEAD>
NL <BODY>
NL <P>The sum of 3 and 5 is: 8</P>
NL </BODY>
NL </HTML>
```



The ability to mix executable code within the structure and format of an HTML document facilitates a straightforward division of labor: The look and feel of the page can be created in plain HTML by artists and web design specialist. Then, programmers and database specialists can insert scripting tags within the HTML to enable dynamic generation of content.

26.1.3 The preprocessor

In this lesson, you will use the **ACTIVE SERVER PAGES (ASP)** functionality that is bundled with MICROSOFT’S INTERNET INFORMATION SERVER (IIS) to build some simple server-side scripts.²



To complete these exercise, you must be able to place ASP files into an IIS web server directory and access the directory with a web browser. An ASP-capable “personal web server” is included with MICROSOFT WINDOWS and is useful for

² Do not confuse ACTIVE SERVER PAGES with “Application Service Provider”—another popular Internet term that uses the same acronym.



building and testing web sites when you do not have a persistent Internet connection to a server.

Although ASP supports the use of a number of scripting languages, we are going to focus on VISUAL BASIC SCRIPTING EDITION (known as VBSCRIPT). VBSCRIPT is a simplified subset of VISUAL BASIC similar to the VBA language used in [Lesson 18](#).

26.2 Learning objectives

- create dynamic web pages using MICROSOFT'S ACTIVE SERVER PAGES technology
- understand the essentials of server-side scripting
- use VBSCRIPT within ASP pages
- understand the Response object and how is it used to send information to browsers
- understand the Request object and how is it used to receive information from browsers
- understand how server-side scripting differs from CGI programming

26.3 Exercises

An ASP file is a plain-text file, virtually identical to an HTML file. There are two important differences, however:

1. ASP files end with an .asp extension.
2. ASP files can contain scripting tags in addition to plain text and HTML tags.

26.3.1 A simple example

In this section, you will create a simple ASP file and review some basic programming constructs such as looping and branching.

➔ Use a text editor to create a new file. Add the basic HTML elements to the document (header, body, title)



To save time, you might want to create a file called **skeleton.html** that contains the core HTML tags. You can cut and paste from this file whenever you need to create a new document.

➔ In the body, enter the following ASP and HTML code:

```
NL <HTML>
NL <HEAD>...</HEAD>
NL <BODY>
NL <P>The sum of 3 and 5 is:
   <%= 3+5 %></P>
```



```
NL </BODY>
NL </HTML>
```

➔ Save the file as **ASPTest.asp**.



Remember to use the asp extension instead of html for files containing scripting language commands. If the asp extension is not used, the web server will not preprocess the ASP tags and VBSCRIPT code will simply be transferred to the browser as plain text.

Recall that in [Lesson 24](#), you could preview HTML-only documents by opening the files without going through a web server: you simply used **File** → **Open** in your browser or double-clicked on the file. In contrast, ASP files *must* be accessed through the server in order for the VBSCRIPT code to be executed. Hence the name “server-side scripting”.

➔ Copy your local copy of **ASPTest.asp** to the web server you are using for the remainder of the project.



From this point forward, the “publish” step will be implied. Ever time you make a change to your local copy of a file, you must publish the change to the web server. Of course, if you are running a

local web server, you can omit the publishing step entirely.

➔ Type the URL of the **ASPTest.asp** document into the address bar of your browser and verify that the script executed properly. This is shown in [Figure 26.1](#).



The URL should be of the form: **http://<server name>/<directory>/ASPTest.asp**.

26.3.2 Using the Response object

Programming in the ASP environment requires an understanding of ASP’s built-in objects. In this section, you will take a brief look at the **Response** object.

26.3.2.1 The Write method

➔ Edit the script in **ASPTest.asp** to read:

```
NL <HTML>
NL <HEAD>...</HEAD>
NL <BODY>
NL <P>The sum of 3 and 5 is:
  <% Response.Write(3+5)%></P>
NL </BODY>
NL </HTML>
```



Ensure that you *do not* include the equals sign after the opening **<%** tag.

FIGURE 26.1: Use NOTEPAD to create an ASP document containing a very short VBSCRIPT expression.

1 Embed a short VBSCRIPT expression within the body of the document.

```

ASPTest.asp - Notepad
File Edit Search Help
<HTML>
<HEAD>
<TITLE>ASP Test</TITLE>
</HEAD>
<BODY>
<P>The sum of 3 and 5 is: <%= 5+3 %></P>
</BODY>
</HTML>
  
```

2 Publish the ASP file to your web server and open it in a browser.

ASP Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search
Address /142.58.96.177/OrderEntry/Utility/ASPTest.asp Go
The sum of 3 and 5 is: 8

When preprocessed on the server, the script is evaluated and replaced with the result of the expression.

3 Select View → Source (or your browser's equivalent) to view the plain HTML sent by the web server.

```

ASPTest[1] - Notepad
File Edit Search Help
<HTML>
<HEAD>
<TITLE>ASP Test</TITLE>
</HEAD>
<BODY>
<P>The sum of 3 and 5 is: 8</P>
</BODY>
</HTML>
  
```

The **Response** object provides a means of controlling the web server's HTTP response to the client. For example, the **write()** method writes a string of text directly to the browser.

`<% Response.Write("Hello") %>` are equivalent.

? The equals sign you used in [Section 26.3.1](#) is shorthand for the **write()** method. In other words, `<%= "Hello" %>` and

You use the **Response.Write()** method whenever you need to dynamically evaluate an expression and output the result to the browser. For example, you can combine VBSCRIPT looping



constructs with HTML to generate a variable-length list of items.

➔ Add the following to the ASP document:

```
NL <BODY>
NL <P>The sum of 3 and 5 is:
  <% Response.Write(3+5)%></P>
NL <UL>List:
NL <% For i = 1 To 5 %>
NL <LI>This is item:
  <% Response.Write(i) %></LI>
NL <% Next %>
NL </UL>
NL </BODY>
```

Notice how it is possible to intermix scripting code and HTML. For example, a **For... Next** statement is normally treated as a single long statement. However, it is possible to suspend the statement after the **For** part (using a closing tag `%>`), enter some HTML, and pick up the statement again (using an opening tag `<%`) for the **Then** part. In this way, the HTML within the loop is written to the browser every time the loop executes.

➔ If necessary, transfer the modified file to the web server and verify the results, as shown in [Figure 26.2](#).

26.3.2.2 The Redirect method

Another useful **Response** object method is **Redirect**. The **Redirect** method is used to immediately send the user's browser to a different URL.

➔ Enter the following VBSCRIPT code at the top of your document, *before* the `<HTML>` tag:

```
NL <% Response.Redirect
  "MyFirst.html" %>
NL <HTML>
NL ...
NL </HTML>
```

➔ Save the change and view **ASPTest.asp** in the browser. It should immediately transfer you to the URL passed as a parameter to the **Redirect** method.



If you do not have a page called **MyFirst.html** (created in [Section 24.3.2.2](#)), the redirect will fail and will return an HTTP 404 (page not found) error.



The **Redirect** method fails if it is executed after anything has been written to the HTTP response. As such, it must be located at the top of the document before any HTML tags, including `<HTML>`.



FIGURE 26.2: Use a VBSCRIPT looping construct and the `Response.Write` method to create a variable-length list.

1 Enclose the first half of the `For... Next` statement in script tags.

2 Write the list tags in normal HTML but use the `Response.Write` method to include the value of `i`.

```
ASPTest.asp - Notepad
File Edit Search Help
<HTML>
<HEAD>
<TITLE>ASP Test</TITLE>
</HEAD>
<BODY>
<P>The sum of 3 and 5 is: <%= 5+3 %></P>
<UL>List:
<% For i = 1 To 5 %>
  <LI>This is item:
  <% Response.Write(i) %></LI>
<% Next %>
</BODY>
</HTML>
```

? The shorthand version of the `Write` method can be used instead: `<%= i %>`.

3 Enclose the last half of the `For... Next` statement in script tags.

```
ASP Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home
Address http://localhost/ASPTutorial/ASPTest.asp Go
The sum of 3 and 5 is: 8
List:
  This is item: 1
  This is item: 2
  This is item: 3
  This is item: 4
  This is item: 5
Done Local intranet
```

? Everything within the `For... Next` statement is written to the browser when the loop repeats.

A simple redirect such as this is of little use. However, when combined with the conditional branching construct in VBSCRIPT (the `If... Then` statement), the redirect feature can be used to control access to pages and the flow of a web-based application.

➔ Modify the redirect statement so that it is conditional on the value of a query string, `LI` (short for "log in"):

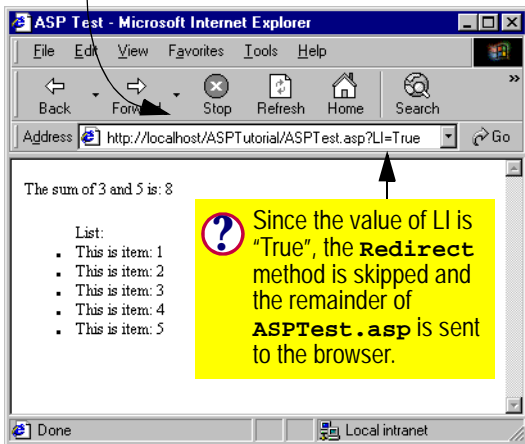
```
NL <% If
Request.QueryString.Item("LI") <>
"True" Then
NL Response.Redirect "MyFirst.html"
NL End If %>
NL <HTML>
NL ...
NL </HTML>
```



- Add the following query to the end of the URL in your web browser's address field:
NL `http://... /ASPTTest.asp?LI=True`
- Press the **Enter** key to view the results, as shown in [Figure 26.3](#).

FIGURE 26.3: Add a “logged in” query string to the end of the URL.

- 1 Append a query string/value pair to the end of the URL (you may be using a different server).



- Change the query string to read:
NL `... ASPTest.asp?LI=False`

- You should be transferred to the URL specified for the **Redirect** method.

In this way, users are prevented from viewing the contents of **ASPTTest.asp** unless the correct query string and value are passed in the URL.

26.3.3 Using the Request object

In [Lesson 25](#) you learned about using query strings and forms to generate HTTP requests and send them to a web server. Using a second built-in ASP object—the **Request** object—you can extract the information contained in HTTP requests and use it in server-side processing. For example, in the previous section, you used the **Request** object to extract the value of the **LI** query string sent by the client. The value was then used by the server to make a simple decision whether to redirect the user.

26.3.3.1 Request object collections

The **Request** object takes all the information in the HTTP request and organizes it into tidy collections. For our purposes, the two most important collections are

- **QueryString** — contains the query/value pairs appended to the URL, and
- **Form** — contains the field/value pairs sent when the form's **Submit** button is pressed.



Like all collections, the contents of the `QueryString` and `Form` collections can be accessed using the `Item` method combined with the name of the key that uniquely identifies the item.

For example, assume a form has the textbox shown below:

```
NL <HTML><BODY>
NL <FORM METHOD="POST"
  ACTION="demo.asp">
NL <INPUT TYPE="text"
  NAME="txtUserName">
NL <INPUT TYPE=submit NAME="cmdSubmit"
  VALUE="Submit">
NL </FORM>
NL </BODY></HTML>
```

When the **Submit** button is pressed, the `Request` object containing the `QueryString` and `Form` collection is sent to the page specified in the `ACTION` attribute (in this case, a file called `demo.asp`).

Within `demo.asp`, it is possible to extract and use the values contained in the `Request` object. For example, in the code below, the user name entered into the form is saved to a local variable (`strUserName`) and then converted to uppercase:

```
NL <HTML>
NL <HEAD><TITLE>Demo.asp</TITLE><HEAD>
NL <BODY>
```

```
NL strUserName =
  Request.Form.Item("txtUserName")
NL <P><%= UCase(strUserName) %></P>
NL </BODY></HTML>
```



Of course, there is a shortcut: Since `Item` is the default method for all collections in ASP, you can omit it and use the following syntax instead:

```
NL strUserName =
  Request.Form("txtUserName")
```

26.3.3.2 Processing forms

In this section, you will create an ASP page that contains a mock authorization routine for the login page you created in [Section 24.5](#). The routine is “mock” because true authorization requires database access, which we have not yet covered.

➔ Create a new ASP file called `Authorize.asp`.



Since the `Authorize.asp` page will contain ASP code only (no HTML) *do not* add the `<HTML>`, `<HEAD>`, and `<BODY>` tags.

➔ Enter the following VBSCRIPT code to implement the mock login decision process.



The resulting ASP file is shown in Figure 26.4.

```
NL <%
NL strUserName=Request.Form.
    Item("txtUserName")
NL strPassword=Request.Form.
    Item("txtPassword")
NL If strUserName=strPassword Then
NL     Response.Redirect "Menu.html"
NL Else
NL     Response.Redirect "Login.html"
NL End if
NL %>
```

In the code above, the values from the form are assigned to local variables (**strUserName** and **strPassword**). The only reason the local variables are used is to simplify the **If... Then** statement; they could just as easily have been omitted and **Request.Form.Item(...)** used instead.



In VBSCRIPT, variables do not have to be declared before use. Contrast this with the use of the **Dim** statement in VBA (see Section 18.3.4.1).

The mock login procedure is as follows:

- If the user's user name and password are identical, then the user is transferred to the application's main menu.

FIGURE 26.4: Create a mock authorization routine using VBSCRIPT.

1 Save the two field values as local variables.

```
authorize.asp - Notepad
File Edit Format Help
<%
strUserName=Request.Form.
Item("txtUserName")
strPassword=Request.Form.
Item("txtPassword")
If strUserName=strPassword Then
Response.Redirect "Menu.html"
Else
Response.Redirect "Login.html"
End if
%>
```

2 Use the local variables in a simple authorization routine.



VBSCRIPT does not have separate data types (string, integer, etc.) and does not require variable declaration. As such, you must be careful not to misspell the names of variables in your code.

- If the user name and password are not identical, the user is transferred back to the login page.

Of course, we will change this later, but it is sufficient for our current purposes.



If you did not create a `Menu.html` page in [Section 24.5](#), you will get an HTTP 404 (page not found) when the authorization succeeds.

- Edit the `Login.html` file you created in [Section 25.5](#). Replace the existing value of the **ACTION** attribute with the relative URL of the new ASP page:

```
NL <HTML>
NL ...
NL <FORM ... ACTION="Authorize.asp">
NL ...
NL </HTML>
```

- Test your login-authorization pages by typing in different combinations of user names and passwords.

26.3.4 Dealing with statelessness

There are two fundamental shortcomings with the current authorization scheme (apart from its simplicity, of course):

1. If the authorization fails, the user is simply returned to the login page with no explanation
2. The authorization mechanism can be bypassed by simply typing the URL of the menu page directly into the browser.

Remember that HTTP is a stateless protocol. What that means is that the web server simply serves up pages without any memory of which user has viewed which pages. In the authorization routine that you have created, the user is redirected back to the login page on authorization failure; however, the web server has no memory of how the user got to the login page. As a consequence, it is up to the application designer to build memory into the application. A simple way to accomplish this is to use query strings.

- Change the file name extension of `Login.html` to `Login.asp`.



Since you will add some VBSCRIPT code to the login page, it must have an `.asp` extension. Otherwise, the code will be ignored by the web server.

- Open `Login.asp` in your editor.
- Add the following conditional heading before the form. You may choose to add heading tags or other formatting.

```
NL <HTML>
NL ...
NL <BODY>
NL <H3><% If Request.QueryString.
    Item("LI") = "Fail" Then
```



```

NL   Response.Write("Login incorrect:
NL   please try again")
NL   Else
NL   Response.Write("Please enter your
NL   user name and password")
NL   End If %></H3>
NL   <FORM ... ACTION="Authorize.asp">
NL   ...
NL   </FORM></BODY></HTML>

```



Another way to accomplish this without using the `Response.Write` method within the code is to keep the HTML code outside of the scripting tags. For example, the code below yields the same result:

```

NL   <HTML>
NL   ...
NL   <BODY>
NL   <% If
NL   Request.QueryString.Item("LI") =
NL   "Fail" Then %>
NL   <H3>Login incorrect: please try
NL   again</H3>
NL   <% Else %>
NL   <H3>Please enter your user name
NL   and password</H3>
NL   <% End if %>
NL   <FORM ... ACTION="Authorize.asp">
NL   ...
NL   </FORM></BODY></HTML>

```

With the addition of this code (either of the variations above will work), the login page will include an error message whenever the request contains the query/value pair `LI=Fail`. If request contains a different value for `LI`, or if `LI` is undefined, then the page will contain a simple instructional message.

➔ Edit `Authorize.asp` and add a query string (recall [Section 25.3.1](#)) to the redirect statement for failed authorization:

```

NL   <%
NL   strUserName=Request.Form.
NL   Item("txtUserName")
NL   strPassword=Request.Form.
NL   Item("txtPassword")
NL   If strUserName=strPassword Then
NL   Response.Redirect "Menu.html"
NL   Else
NL   Response.Redirect
NL   "Login.asp?LI=Fail"
NL   End if
NL   %>

```



Remember to change `Login.html` to `Login.asp` in the redirect statement.

➔ Test your application to ensure that the conditional heading is working correctly.

By carrying the value of the query string `LI` from one page to the next, a primitive form of memory has been added to the application.



26.3.5 Robust authorization

A robust authorization mechanism is one that cannot easily be defeated or bypassed. As it now stands, our authorization mechanism can be side-stepped by typing `Menu.html` into the browser's address window.

It would be straightforward to add a conditional redirect to the start of the menu file using VBSCRIPT:

```
NL <% If Response.QueryString("LI") <>
NL     True Then
NL     Response.Redirect "Login.asp"
NL End If %>
NL <HTML>
NL ...
NL </HTML>
```

However, this provides very little additional protection since a knowledgeable user could simply type `Menu.asp?LI=True` into the browser's address window.

There are at least two ways around this problem:

1. Use a **hash function** to generate a value for `LI` that is difficult for users to guess. For example, instead of setting `LI=True`, the hash function would generate a longer value such as `LI=FJ910392XZZ381847292873086` that could also be independently generated and verified on subsequent pages.

2. Use ASP's built-in `session` object to read and write session-level variables on the server.

In the following sections, we will explore the use of hash values and query strings. In [Lesson 27](#), we will use ASP's session-level variables to implement a slightly different authentication scheme.

26.3.5.1 Creating a hash function

A hash function is simply a function that takes some known values and transforms them into a jumbled "hash" of digits and/or characters.

- ➡ Make the following changes to `Authorize.asp`, as shown in [Figure 26.5](#):

```
NL <%
NL     strUserName=Request.Form.
NL         Item("txtUserName")
NL     strPassword=Request.Form.
NL         Item("txtPassword")
NL     If strUserName=strPassword Then
NL         strHash = CStr(CLng(Date)* 3317)
NL         Response.Redirect "Menu.html?LI="
NL             & strHash
NL     Else
NL         Response.Redirect
NL             "Login.asp?LI=Fail"
NL     End if
NL %>
```



FIGURE 26.5: Generate a hash value when authorization is successful.

1 This hash value simply transforms the current date into a long numerical value.

```

Authorize.asp - Notepad
File Edit Search Help
<%
strUserName = Request.Form.Item("txtUserName")
strPassword = Request.Form.Item("txtPassword")

If strUserName = strPassword Then
    strHash=CStr(CLng(Date)*3317)
    Response.Redirect "Menu.asp?LI=" & strHash
Else
    Response.Redirect "Login.asp?LI=Fail"
End If
%>
  
```

2 Pass the hash value to the target page using a query string.

This code calculates an extremely simple hash value and appends it as a query string to the redirection URL. The elements of this particular hash function are:

- **Date()** is a built-in function that returns the current date. The underlying representation for dates and times is numeric.

- **CLng()** is a built in function that converts a number into a long integer. In this case, it converts the number returned by the **Date()** function.
- 3317 is just an arbitrary multiplier
- **CStr()** is a built-in function that converts any data into a string of characters.

Like all hash functions, this one takes one or more publicly known values (the current date) and transforms it in a (hopefully) non-obvious way into a longer series of digits. Since the same hash function can be calculated on other pages and compared to the value being passed around in the query string, it is possible to identify users who have logged in successfully.

26.3.5.2 Using a hash value to confirm authorization

In this section, you will recalculate the hash value in the menu page and compare it to the value passed in the query string. If they are the same, then it is likely that the user was properly authorized by the code in **Authorize.asp**.

- ➔ Rename **Menu.html** to **Menu.asp** and open the file for editing.
- ➔ Add the following code to the top of the file, as shown in [Figure 26.6](#):


```

NL <%
NL strHash=CStr(CLng(Date)*3317)
  
```



```

NL If Request.QueryString.Item("LI")
  <> strHash Then
NL     Response.Redirect "Login.asp"
NL End If
NL %>
NL <HTML>
NL ...
NL </HTML>

```

- ➔ Open **Authorize.asp** for editing and replace the reference to **Menu.html** with **Menu.asp**.

If the hash value passed in the query string is not identical to the hash value calculated on the menu page, the user is redirected to the login page.



If you make a typographical error and the hash function you use in **Authorize.asp** is different from the one in **Menu.asp**, you will never see the menu. Instead, each time you will be immediately redirected to the login page. Such redirection bugs can be tricky to diagnose because you are immediately transferred off the page with the error during testing.

Naturally, an authorization mechanism based on visible hash values is only useful if the hash function is secret and is very difficult to guess. In this example, the hash values change every day. However, the values change in such a

predictable and linear manner that it would be very easy for a hacker to induce the hash function and gain unauthorized access to the application.



There is an entire science devoted to the creation of difficult-to-guess hash functions. In general, the function should generate long values that have a highly non-linear relationship with the inputs.



You might be tempted to pass a simple authorization string (e.g., **LI=True**) via a **hidden field** in a form (revisit [Section 25.3.4](#) for more information on different field types in HTML). Although hidden fields are "less visible" than query strings, they are in plain view in the form's HTML source. As such, they provide no more security than query strings.

26.4 Discussion

26.4.1 Server-side scripting and CGI

The earliest web servers used a mechanism called the **Common Gateway Interface** (CGI) to allow the web server to interact with other server-side programs. The difference between CGI programming and server side scripting is that in server-side scripting:



FIGURE 26.6: Create a conditional redirect feature that compares a hash value to a value passed in a query string.

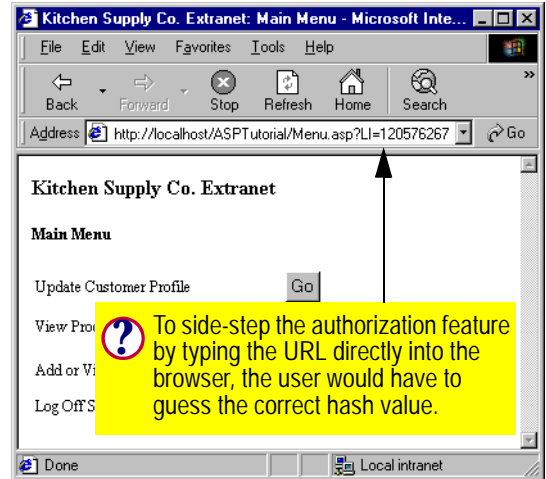
1 Recalculate the hash value using the same secret hash function.

```

Menu.asp - Notepad
File Edit Search Help
<%
strHash=CStr((CLng(Date)*3317))
IF Request.QueryString.Item("LI") <> strHash Then
Response.Redirect "Login.asp"
End If
%>
<HTML>
<HEAD>
<TITLE>Kitchen
</HEAD>
<BODY>
<H2>Kitchen Sup
<H3>Main Menu,
</BODY>
</HTML>

```

2 Compare the calculated hash value to the one passed in the query string. If they are different, redirect to the login page.



- the client requests a document containing special tags and code interspersed within the page's HTML structure; and
- the code is executed on the server before sending the document to the client.

In CGI programming, the URL in the client's HTTP request is not a document at all, but the name of an executable program in a special directory on the server (e.g., <http://<server>>

[name>/cgi-bin/run_me](http://<server>/cgi-bin/run_me)). When the server receives the request from the client, the program is executed. Since the CGI program is not embedded within an HTML document, it is the CGI program's responsibility to "write" a response that can be sent back to the client. That is, the CGI program must do its task (e.g., perform a database lookup) *and* generate all the HTML text to be sent back to the browser.



Although server-side scripting is less wasteful of server resources (a major consideration when the potential exists for thousands of users to access your site at the same time) and is easier to create and deploy than CGI, CGI is still widely used. One reason for its longevity is that it is a vendor-independent standard that is bundled with all web servers. ACTIVE SERVER PAGES is freely bundled, but only with the MICROSOFT'S INTERNET INFORMATION SERVER¹. COLDFUSION is available for a wider range of web servers, but is sold as a separate product. CGI is free.



A number of cross-platform open-source server-side scripting languages have emerged, including PHP (see www.php.net) and JAVA SERVER PAGES (JSP). Although the format of the tags and the syntax of the language varies depending on which scripting infrastructure you use, the basic principles of server-side scripting are identical in ASP, COLDFUSION, PHP, and JSP.

Another reason for CGI's continued use is the sheer bulk of legacy code written for CGI applications. Although a CGI program can be written in just about any language (C++,

FORTRAN, VISUAL BASIC), a freely available interpreted language called PERL caught on early as an easy way to parse text and dynamically create HTML (remember, the output of a CGI script is sent back to the client, not an intermediate page, as in ASP).

26.5 Application to the project

➔ Create a new ASP file called **Menu_process.asp** to process the information passed to the server by the menu page.

HINT: You can use the **IF... ELSEIF...** construct to check whether specific name/value pairs were passed in the HTTP request. For example:

```
NL <%
NL If Request.Form("cmdCustomer")="Go"
NL Then
NL     strRedirect="Customer.asp"
NL ElseIf
NL     Request.Form("cmdProduct")="Go"
NL Then
NL     strRedirect="ProductList.asp"
NL ElseIf
NL     Request.Form("cmdOrder")="Go" Then
NL     strRedirect="Order.asp"
NL ...
NL End If
```

¹ CHILLISOFT sells a product that provides an ASP functionality for non-IIS web servers.



```
NL Response.Redirect strRedirect
```

```
NL %>
```



VBSCRIPT also supports a **select... Case... End select** construct that is functionally identical (but slightly more elegant) than the **ElseIf** construct.