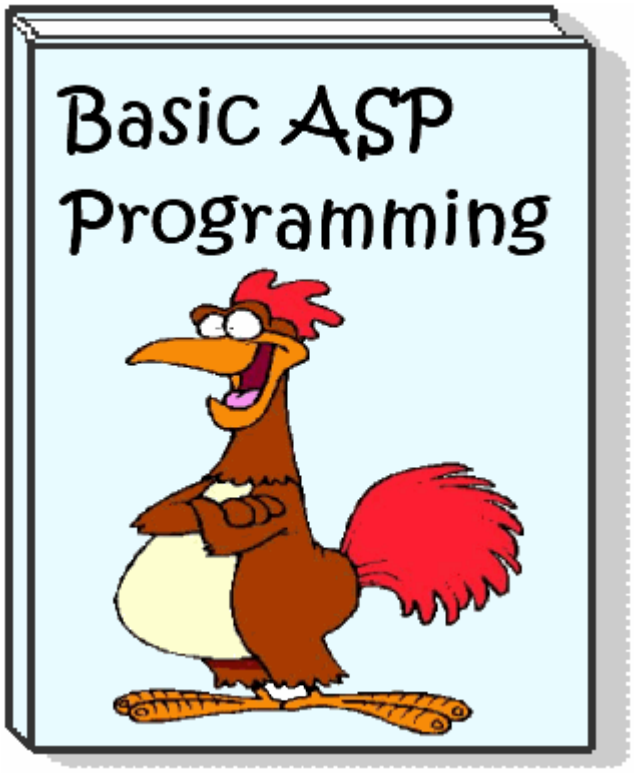


Contents

Warranty, Disclaimer and Copyright Policy

This material is provided on an "as-is" basis, and Bucaro TecHelp makes no warranty or representation, express or implied, with respect to its quality performance or fitness for a particular purpose. In no event shall Bucaro TecHelp be liable for direct, indirect, special, incidental, or consequential damages arising out of the use of this material.

No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this manual, Bucaro TecHelp assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein. This information is provided with the understanding that Bucaro TecHelp is not engaged in rendering medical, legal, accounting or other professional service. If legal advice or other expert assistance is required, the services of a competent professional person should be sought.

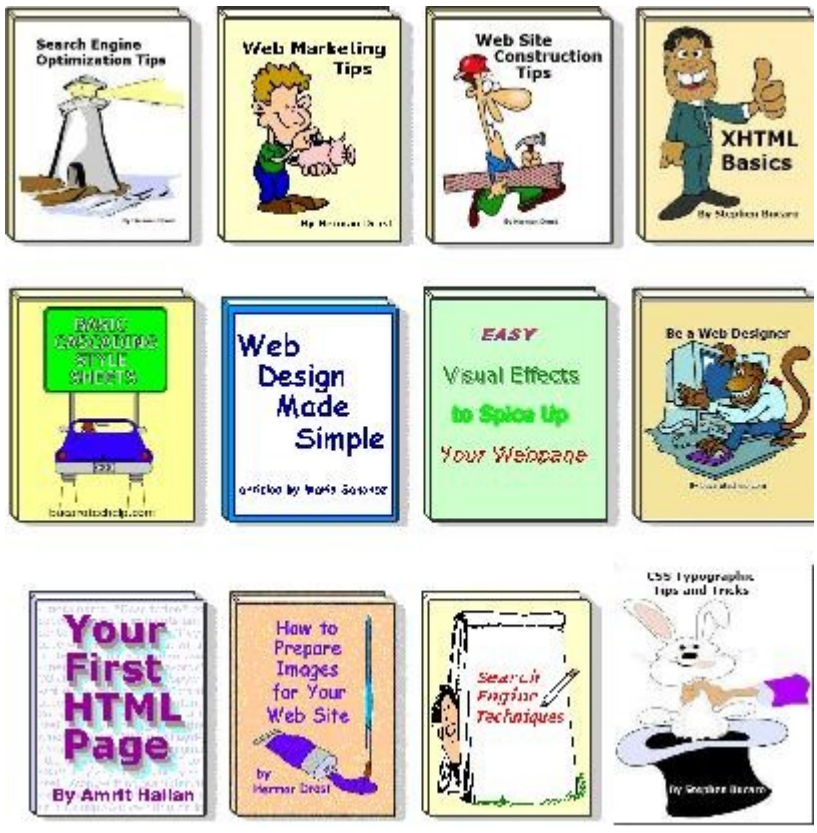
By using this material, the user assumes complete responsibility for any and all damages resulting from that use. Use of this program and materials requires agreement to the terms of this warranty and disclaimer. If you do not agree to the terms of this warranty, do not use this material.

Copyright(C)2008-2010 Bucaro TecHelp. Permission is granted for this program to forward, reprint, distribute, offer as free bonus or part of a product for sale as long as no changes are made to copies that are distributed.

To receive an email notification when new articles, ebooks, clipart, graphics, or other content has been added to Bucaro Techelp, Click Here to subscribe to Bucaro TecHelp News Feed Notification.

Contents

Get More Free eBooks and Other Cool Stuff



Visit Bucaro Techelp to download FREE ebooks including Bucaro TecHelp s popular PC Tech Toolkit. Read Bucaro TecHelp's famous Easy Java Script and Easy CSS

tutorials with cut-and-paste code. Learn Basic PC Anatomy and where to find FREE diagnostic Tools and technical assistance. Learn how to start your own online business, including many examples of people who started successful businesses.

To receive an email notification when new articles, ebooks, clip art, graphics, or other content has been added to Bucaro Techelp, Click Here to subscribe to Bucaro TecHelp News Feed Notification.

Contents

Introduction

In the early days of the web, creating a webpage was pretty straight forward. You formated the webpage using HTML (HyperText Markup Language). Every time you accessed the webpage, it looked exactly the same. Webpages were "static".

CGI (Common Gateway Interface) scripts were used to pass data from web forms to the web server. CGI scripts were usually written in Perl. Perl is a language that is about as easy to understand as differential calculus.

Later advances brought Java Script, DHTML (Dynamic HTML) and CSS (Cascading Style Sheets). This allowed webmasters to create some pretty amazing applications. With these new technologies web pages became "dynamic". But these are all client side technologies. The applications could use only data that was on the webpage or was entered the web browser by the user.

ASP (Active Server Pages) is a server side technology that allows you to share data between the web server and the user, and to create dynamic webpages without worrying about the capabilities of the user's browser. With ASP, technically you don't need any "webpages" on your server. You need only data and the ASP code to build webpages on the fly.

Today ASP competes with PHP (Personal Home Page), an open source scripting language that looks a lot like Perl. ASP can be written in Java Script, but is usually written in VBScript (Visual Basic Script). "Visual" refers to Microsoft's visual development environment where you create applications by dragging and dropping visual elements into place. Basic is a very easy to learn programming language.

In the early days of computers, programming languages where compiled. The programmer's source code was run through an application that converted it to machine instructions. Then the compiled executable program could be run on the computer. Today compiled programs are used when the code must be fast and efficient. But most modern programming lanugages are scripting languages.

With a scripting language, the programmer's source code is passed to an interpreter program that does the compiling on the fly. Today's computers are so powerful that scripting languages are fast enough for most purposes. VBScript is a powerful and easy to understand scripting language.

Contents

Classic ASP vs ASP.NET

By Stephen Bucaro

You may wondering "what's the difference between ASP and ASP.NET?". ASP.NET is the successor to ASP. When you take away all of the marketing hype, ASP.NET was developed because Microsoft's (COM) Component Object Model wasn't reliable over networks (especially the Internet), and Microsoft wanted to promote C#, its replacement for Java.

COM requires software objects to be listed in the computers registry. This works satisfactorily if the calling program and the object are on the same machine, but if a client computer wants to use a software object on a server, it has to look up that object in the servers registry. I remember many times keeping my fingers crossed hoping that would work.

Today network and Internet clients and servers use SOAP (Simple Object Access Protocol) which uses (XML) eXtensible Markup Language to communicate to each other about what software objects are available and how to access them. This whole process now know as "Web Services".

Another difference between Classic ASP and ASP.NET is that ASP is "interpreted", while ASP.NET is compiled. When you run an interpreted language, the source code is parsed and executed on the fly each time the program is executed. When a language is compiled, the source code must be converted to the computer's native binary language before it can be executed. This takes time, but after the compilation, the machine code program runs much faster.

In the early days when computers were much slower, you had to compile a program if you wanted respectable performance. But when computers became faster and more powerful, interpreted languages like VBScript and JavaScript provided acceptable performance. But despite today's computers being even faster and more powerful, code needs to be compiled to achieve acceptable performance.

Another difference between Classic ASP and ASP.NET is that ASP.NET is an "application framework". Basically this means it uses code libraries and a modular structure. For example there is a module called the (CLR) Common Language Runtime between the ASP.NET code and the machine that converts the ASP.NET code to the computer's native binary language before it can be executed.

So Which Should I Use, Classic ASP or ASP.NET?

If you want to develop a dynamic Web site where services are provide and consumed by your local web server, in other words you are not providing Web Services to other servers on the Internet and you are not accessing Web Services provided by other servers on the Internet, then I would recommend using Classic ASP. I say this because the complexity is much lower.

COM works perfectly fine between a Web server and Web Browsers, except when a COM DLL needs to be updated, then the Web server needs to be restarted in order to register the COM objects in the DLL. Most ASP COM objects are very

mature today, so updates should be rare.

The difference between "interpreted" and "compiled" is mostly hype, because the Internet uses HTML and XML, both which are required to be "parsed", in other words "interpreted".

With Classic ASP you can design a very powerful Web site without having to figure out how to use a complicated IDE (Integrated Development Environment). You can just type easy to understand VBScript or JavaScript into Web pages using notepad.

However, if you plan to provide or consume Web services, you will probably want to go with ASP.NET. To develop your Web site you can use Microsoft Visual Web Developer, the Express version which you can download from Microsoft's Web site for free.

Contents

Creating Webpages With ASP

By Stephen Bucaro

ASP webpages have the file extension *.asp*. You can change the file extension of an html webpage from *.htm* to *.asp*, and run it on an ASP server. ASP will scan through the text on the page looking for the characters <% and %>. ASP executes any instructions it finds between the <% tag and the %> tag. These tags are referred to as the *ASP delimiter* tags.

To create a simple ASP page, type the following text into Windows Notepad and save the file with the extension *.asp*.

```
<html>
<body>
<% Response.Write "This is my first ASP page" %>
</body>
</html>
```

The above code uses the *Write* method of ASPs built-in *Response* object to write to the client's browser.

In the example below, VBScript's *Date* function is used to write todays date and time to the client's browser.

```
<html>
<body>
<%
Dim today
today = Date()
Response.Write "Today's Date: " & today
%>
</body>
</html>
```

The first line of ASP code creates a variable (a named storage location) to store the date/time string returned by the *Date* function. The *Response* object is then used to write the character string "Today's Date: " and the contents of the variable. Note how an ampersand (&) is to concatenate the strings.

VBScript provides many useful date and time functions. In the example below, the *DatePart* function is used to extract the hour part of the date/time string returned by the *Date* function.

```
<html>
<body>
<%
Dim hour
hour = DatePart("h",Date)
Response.Write "The Hour is: " & hour
%>
</body>
</html>
```

In order to use ASP to create dynamic webpages, you need to use VBScript's decision making capabilities. VBScript makes decisions by using *conditional statements*. The most common conditional statement is *If Then Else*.

```
If "condition" Then
   "Do this"
Else
   "Do that"
End If
```

The example below uses an *If Then Else* statement used to decide which message to write to the client's browser, "Good Morning" or "Good Evening".

```
<html>
<body>
<%
Dim hour
hour = DatePart("h",Date)

If hour < 12 Then
   Response.Write "Good Morning"
Else
   Response.Write "Good Evening"
End If
%>
</body>
</html>
```

The code above uses the < (less than) comparison operator in the conditional statement. The conditional statement says "if the hour is less than 12 then write good morning, else write good evening". Technically the second part of the statement is not required. You could have an *If Then* statement without the *Else* as shown below.

```
<%
If hour < 12 Then
   Response.Write "Good Morning"
```

```
End If
%>
```

You could also rearrange the conditional statement to use the > (greater than) comparison operator as shown below.

```
<%
If hour > 12 Then
   Response.Write "Good Evening"
Else
   Response.Write "Good Morning"
End If
%>
```

You could also extend the conditional statement by adding another *Else If* as shown below.

```
<%
If hour < 12 Then
   Response.Write "Good Morning"
Else If hour >= 12 And hour <= 16 Then
   Response.Write "Good Afternoon"
Else
   Response.Write "Good Evening"
End If
%>
```

The code above uses the >= (greater than or equal to) and <= (less than or equal to) comparison operators in the *Else If* statement. That's required because otherwise when the hour is exactly 12 or exactly 16, no conditional statement would be true. The added *Else If* statement says "if the hour is greater than or equal to 12 and less than or equal to 16 then write "Good Afternoon".

If you need too many added *Else If* statements in your control structure, you might want to use a *Select Case* control structure instead, as shown below.

```
<%
Select Case hour
Case 7
   Response.Write "Time to wake up!"
Case 9
   Response.Write "Time to start work"
Case 12
   Response.Write "Lunch Time"
Case 17
   Response.Write "Time to stop work"
Case 22
   Response.Write "Time to go to sleep"
End Select
%>
```

When you use Response.Write to write to the user's browser, the text you write can be HTML, CSS or even Java Script. For example the statement below writes HTML *font* tags along with the text.

Response.Write "Good Evening"

Note the use of double quotation (") and single quotation (') marks in the write string above. Quotation marks have significance in both ASP and HTML. The ASP *Response.Write* command is going to start writing at the first single or double quotation mark it finds, and stop writing at the next similar quotation mark it finds.

Using ASP to write code to a webpage (html or Java Script) can get quite complicated. With a combination of double quotation marks, single quotation marks, escape characters (\") and character codes (&quote;) you can usually get the job done.

Some programmers prefer to avoid using the *Response.Write* command by using ASP delimiter tags as shown below.

```
<html>
<body>
<%
Dim hour
hour = DatePart("h",Date)

If hour < 12 Then
%>
Good Morning
<%
Else If hour >= 12 And hour <= 16 Then
%>
Good Afternoon
<%
Else
%>
Good Evening
<%
End If
%>
</body>
</html>
```

This is perfectly legal and works fine. I wouldn't use this method in this particular case because all the ASP delimiter tags make the code look sloppy and confusing. But if you have large areas of text and/or html between the ASP code blocks, this method lets you avoid typing many *Response.Write* statements.

In this article, you learned that ASP is a server technology that allows you to create dynamic webpages. ASP is usually written in VBScript. VBScript is a powerful but easy to learn scripting language. Your code must be placed within ASP delimiter tags <% %> on the webpage to be executed by ASP. The webpage must be saved with the file extension *.asp*.

Contents

Retrieving Form Data With ASP

By Stephen Bucaro

We are all familiar with those forms on the web where you type some information into a text box, set a checkbox, or select an item in a list, and then click on the *Submit* button to send the information off to the server. An HTML form allows a webpage to display input fields and controls where users can enter data and submit it to the server. The HTML code for an example form is shown below.

```
<form>
Name: <input type="text" name="name"><br>
Email: <input type="text" name="email"><br>
Message: <textarea name="message"></textarea><br>
<input type="submit" name="submit" value="Submit">
</form>
```

The code above would create a typical; contact form. After the user enters the appropriate data in the form's fields and clicks on the *Submit* button, the browser will construct a *querystring* containing the names and contents of the forms two text boxes and the textarea. The querystring will be appended to the webpage's URL.

For example, if the user entered the name "Bob Sled", the email address "bsled@dmn.com" and the mesage "good morning", the URL with the querystring appended would be as shown below.

```
http://domain.com/page.asp?
name=Bob+Sled&email=bsled@dmn.com&message=good+morning
```

(on one line)

This line would appear in the browsers address bar. Note that the querystring is separated from the URL with a question mark, and the querystring consists of name=value pairs separated by ampersands. Spaces are replaced with plus signs.

It's interesting to know that you can test your asp form processing code by typing the URL?querystring directly into the browsers address bar, or by formatting the URL?querystring as a link and clicking on it. The form is not required.

This example used the <form> tag with no attributes. In that case the form is submitted to the same webpage that contains the form. You could submit the form to a different webpage by adding the "action" attribute to the <form> tag, as shown below.

```
<form action="http://domain.com/process_from.asp">
```

This example submitted the form with the default *get* method. Form data submitted by the *get* method can be read by a server-side ASP script using the *Request.QueryString* method as shown below.

```
Dim strName, strEmail, strMessage
strName = Request.QueryString("name")
strEmail = Request.QueryString("email")
strMessage = Request.QueryString(""message")
```

There are several things to consider when submitting a form with the *get* method. Some older browsers and web servers limit how long a querystring can be. the

querystring is visible in the browser's address bar. If the form contains hidden data or the user enters a password, these will be visible to anyone standing near the screen.

Using the *post* method to submit a form causes the browser to send the form data to the server in a way that is not visible to the user. To use the *post* method to submit the form, add the *method* attribute to the the <form> tag as shown below.

```
<form method="post">
```

Note: It's good practice to always put both the *action* and *method* attributes in the <form> tag. if want to submit the form with the *get* method, use method="get".

Form data submitted by the *post* method can be read by a server-side ASP script using the *Request.Form* method as shown below.

```
Dim strName, strEmail, strMessage
strName = Request.Form("name")
strEmail = Request.Form("email")
strMessage = Request.Form(""message")
```

Other Form Controls

This example used the textbox control and the textarea control to collect data from the user. You can also use the checkbox, radio button, and select list controls. The code shown below puts a checkbox on the form.

```
<input type="checkbox" name="color" value="red">Red
```

The checkbox can be read by an server-side ASP script as shown below.

```
Dim strColor
strColor = Request.Form("color")
```

You need a separate *Request.Form* statement for each checkbox in the form. Radio buttons, however, are usually used in groups. In a group of radio buttons only one radio button can be set. Setting a different radio button causes the previously set radio button to be unchecked. The form shown below contains a group of radio buttons.

```
<form action="process_form.asp" method="post">
<input type="radio" name="color" value="red" checked>Red<br>
<input type="radio" name="color" value="blue">Blue<br>
<input type="radio" name="color" value="green">Green<br>
<input type="submit" name="submit" value="Submit">
</form>
```

These radio buttons act as a group because they all have the same name. The *checked* parameter in a radio button tag causes it to be checked by default.

Which radio button the user selected can be determined by a server-side ASP script statement as shown below.

```
Dim strColor
strColor = Request.Form("color")
```

Note that the statement used the name of the group to retrieve the value of the radio button that was set by the *checked* parameter or by the user. You can have more than one group of radio buttons in a form by giving all the buttons the name of their specific group.

The *select* list works similar to a set of radio buttons. A *select* list is also called a "drop down list". The selected item in the list is highlighted. In a select list only one item can be selected. Selecting a different item in the list causes the previously item to be unselected. An example of a select list is shown below.

```
<select name="color">
<option value="red" selected>Red</option>
<option value="blue">Blue</option>
<option value="green">Green</option>
</select>
```

The *selected* parameter in an option tag causes it to be selected by default.

Which item the user selected can be determined by a server-side ASP script statement as shown below.

```
Dim strColor
strColor = Request.Form("color")
```

Note that the statement uses the value of the name attribute in the <select> tag to retrieve the value of the option that was selected by The *selected* parameter or by the user.

You can have more than one select list in a form by giving each one a different name.

Adding the *multiple* parameter to a <select> tag allows the user to select more than one item in the list. With a multi-select list you can use the *Count* property of the *Request.Form* method to determine how many items the user selected. An example of a multi-select list is shown below.

```
<select name="color" size=6 multiple>
<option value="red">Red</option>
<option value="blue">Blue</option>
<option value="green">Green</option>
<option value="black">Black</option>
<option value="gray">Gray</option>
<option value="white">White</option>
</select>
```

Note the size attribute in the <select> tag above will cause all 6 items in the list to be visible, creating a regular list rather than a drop-down list.

You could retrieve all the selected items from the list with a server-side ASP script *For* loop as shown below.

```
For i = 1 To Request.Form("color").Count
   Response.Write("Selected color: "
      & Request.Form("color")(i) & "<br>"
Next
```

Form Validation

Generally you want to use client-side script to check the forms data before you allow it to be submitted to the server. One way to do that is to replace the froms *Submit* button with a regular button and use that button's "onClick" event to execute a client-side validation script, as shown below.

```
<input type="button" value="send"
   onClick="Validate(this.form)">
```

The client-side validation script would check the form controls contents and, if acceptable, would submit the form, for example with the following statement.

```
form.submit();
```

If the the form controls contents was not acceptable, the form would not be submitted, but instead the user would be presented with a dialog box informaing them of the problem. The validation function could be written in client-side Java Script. Java Script form validation is the topic of another article.

However, one basic validation that you might want to perform at the server-side is to verify that the *post* does contain **some** data. You can use *Request.ServerVariables("CONTENT_LENGHT")* to retrieve the length of the post data and use it as shown below.

```
If Request.ServerVariables("CONTENT_LENGHT") <> 0 Then
.
.
.
End If
```

If you submit the form using the *get* method, You can use *Request.ServerVariables("QUERY_STRING")* to test if the QueryString is empty as shown below.

```
If Request.ServerVariables("QUERY_STRING") <> "" Then
.
.
.
End If
```

Complete Example

Now lets create a more complete example of an ASP webpage that retrieves data from a form. Normally you would do something with the data, like write it to a file or enter it into a database. But writing files and accessing databases with ASP are subjects for future articles. In this example we will simple reflect the data back to the users browser.

The ASP webpage fits into the basic structure shown below.

```
<%
If Request.ServerVariables("CONTENT_LENGHT") <> 0 Then
' show results
Else
```

```
' show form
End If
%>
```

If the Server Variable *CONTENT_LENGHT* equals 0, that means the form has not yet been submitted, so put the html code to display the form in the *Else* section as shown below.

```
<%
If Request.ServerVariables("CONTENT_LENGHT") <> 0 Then
' show results
Else
' show form
%>
 <form action="process_from.asp" method="post">
 Name: <input type="text" name="name"><br>
 Age: <input type="text" name="age"><br>
 Favorite Color:
 <select name="color">
 <option value="red">Red</option>
 <option value="blue">Blue</option>
 <option value="green">Green</option>
 <option value="black">Black</option>
 <option value="gray">Gray</option>
 <option value="white">White</option>
 </select><br>
 <input type="submit" name="submit" value="Submit">
 </form>
<% End If %>
```

Note the arrangement of the ASP delimiters, which puts the form html outside the ASP code so that we don't need to code *Response.Write* statements to generate the form.

If the Server Variable "CONTENT_LENGHT" does not equal 0, that means the form has been submitted, we create three variables to hold the form's data, we use three *Request.Form* statements to retrieve the froms data, and then we use three *Response.Write* statements to generate a webpage to display the results in the users browser.

```
<%
If Request.ServerVariables("CONTENT_LENGHT") <> 0 Then
' show results
Dim strName, nAge, strColor
strName = Request.Form("name")
nAge = CInt(Trim(Request.Form("age")))
strColor = Request.Form("color")
Response.Write "Name: " & strName & "<br>"
Response.Write "Age: " & CStr(nAge) & "<br>"
Response.Write "Color: " & strColor & "<br>"
Else
' show form
%>
 <form action="process_from.asp" method="post">
 Name: <input type="text" name="name"><br>
```

```
 Age: <input type="text" name="age"><br>
 Favorite Color:
 <select name="color">
 <option value="red">Red</option>
 <option value="blue">Blue</option>
 <option value="green">Green</option>
 <option value="black">Black</option>
 <option value="gray">Gray</option>
 <option value="white">White</option>
 </select><br>
 <input type="submit" name="submit" value="Submit">
 </form>
<% End If %>
```

Note the little trick that did with the "age" value. Decode a statement like this from the inside out. The results of *Request.Form("age")* is passed to the *Trim()* function, which removes any leading or trailing blanks. The results of the *Trim()* function is passed to the *CInt()* function, which converts the string value to an integer.

I did this to demonstrate how you would convert the data returned by a form into a number that you could use in further mathematical calculations. You might go one step further and test nAge to make sure its a number, as shown below.

```
If isNumeric(nAge) Then
' use in calculation
Else
' handle error
End If
```

Then, just before I write nAge back to the users browser, I convert it back to a string. (I'm almost sure the *Response.Write* method would have done that automatically).

Note: there are many other variable type conversion functions, for example *CSng()* converts a value to a single-precision floating point number, *CCur()* converts a value to a currency.

In this article, you learned how to create some basic html forms and how to use ASP to retrieving data from forms. You also learned that you should use client-side script to check the forms data before you allow it to be submitted to the server. You received a hint as to how to convert the from's data to numerical values if you need to use it in calculations. And lastly, you understand that retrieving the forms data just to reflect it back to the user with *Response.Write* statements is only of limit value. Next we will learn how to write the data to a file or enter it into a database.

Contents

The ASP Application Object

By Stephen Bucaro

When you install Internet Information Server (IIS), the installer creates a default Web site in a directory named *wwwroot*. You could start building your website in the wwwroot directory, but if you are a Web host provider, you will want to configure multiple websites to rent to different customers. You create websites using the Internet Service Manager (ISM).

Each website will run in its own separate process, so if one website crashes, the other websites will continue to run. Each website is an application associated with an application object. You can create subdirectories within the website (or virtual directories at any location) and use ISM to make them into applications. Then if an application crashes, the website will continue to run.

A website can store data in named memory locations called "variables". Each variable has a "scope" or level from which it is visible and accessible. For example, when a variable is declared within a procedure or function, it is only visible to and can be accessed by only code within that procedure or function. When processing exits the procedure or function, a variable declared within is lost.

Variables can be declared on a webpage outside of any procedure or function. That variable would have a "page" scope and would be visible to and can be accessed by only code within that webpage. Variables can be declared at the application level. Below is an example of how to declare an application level variable.

```
Application("variable_name") = "some text"
```

A variable declared at the application level is visible to and can be accessed by all pages within that application. An "Application" object is associated with each application and provides an event that is triggered when the application starts and when the application ends. Active Server Pages (ASP) provides access to these events through a special file named "global.asa". The contents of the global.asa file is shown below.

```
<script language="VBScript" runat="server">
Sub Application_OnStart

End Sub

Sub Application_OnEnd

End Sub
</script>
```

The *Application_OnStart* and *Application_OnEnd* event handlers can be used for code that needs to run when the first page of the application is served and when the Website is shut down.

Scripts running at the application level can be accessed by all user sessions. The Application object provides two methods we can use to protect shared variables from being written to by more than one user at a time.

Lock restricts other code from writting
Unlock releases the variable for writting

For example, if you wanted to keep a count of the number of pages served by an application, in the global.asa file you could declare an application level variable

hits as shown below.

```
<script language=VBScript runat=server>
Sub Application_OnStart
   Application.Lock
   Application("hits") = 0
   Apllication.Unlock
End Sub
</script>
```

When the *Application_OnStart* event is fired at the start of the application, the application is locked while the variable *hits* is initialized to zero. Every page of the application must contain code that increments the variable when that page is served, as shown below.

```
<%
Application.Lock
Application("hits") = Application("hits") + 1
Application.Unlock
%>
```

The variable *hits* has application scope, so it is visible and accessible by all pages in the application. The code locks the application while *hits* is incremented by one with every webpage served. You don't need to lock the application to read application scope variables because reading does not modify the value of the variable. The code shown below could be used on one or more pages to display the current value of the "hits" variable.

```
Number of hits: <%= Application("hits") %>
```

If you wanted to keep a count of the number of times a specific page was served, the page could contain code that uses an application level varaible as shown below.

```
<%
Dim strUnique
strPage = Request.Servervariables("SCRIPT_NAME")

Application.Lock

If isEmpty(Application(strPage)) Then
   Application(strPage) = 0
End If

Application(strPage) = Application(strPage) + 1

Application.Unlock
%>
```

The script uses the Request.Servervariables method with the *SCRIPT_NAME* constant to retrieve the name of the script, which is actually the file path of the webpage. It then locks the application while using the isEmpty function to determine if a variable with that name has been created. The first time the page is served, the variable has not been created, so the code creates the variable and initializes it to zero. Then the variable is incremented by one.

On the same webpage, somewhere below that code, the code shown below could be used to display the current value of the variable.

Page Hits: <%= Application(strPage) %>

In this article, you learned that variables can be declared at the application level that will be visible to all webpages in the application. This means that multiple pages may try to write to an application level simultaineousy. To prevent this, the Application object provides the Lock and Unlock methods. The Application object also provides the *Application_OnStart* and *Application_OnEnd* event handlers which can be used for code that needs to run when the first page of the application is served, and when the Website is shut down.

Contents

The ASP Session Object

By Stephen Bucaro

The period during which a visitor is at your website is called a "session". When a visitor retrieves their first page from your website, ASP creates a *session* object. A session object is associated with each user (each web browser) while they are interacting with your website.

Variables can be declared at the session level. Below is an example of how to declare a session level variable.

Session("variable_name") = "Some text"

A variable declared at the session level is visible to, and can be accessed by any webpage visited during a session. You can use the session object to store information related a specific visitor. You can store information like the user's login status, website personalization parameters, or shopping cart contents in a session level variable.

If the visitor does not interact with your website again within a certain period, the session is considered closed and the session object is destroyed. That time period is the *session timeout* set in IIS using the Internet Service Manager (ISM).

The session object provides an event that is triggered when the session starts and when the session ends. ASP provides access to these events through a special file named "global.asa". The contents of the global.asa file is shown below.

```
<script language=VBScript runat=server>
Sub Session_OnStart

End Sub

Sub Session_OnEnd

End Sub
</script>
```

The session object relies on browser cookies. ASP creates a session cookie the first time a user accesses your website. If the user disables cookies in their browser, you cannot track their session with the session object.

• You can still track the user through query strings or through form elements.

Only the *SessionID* is stored in the cookie, and it's encrypted. Any other session level variables that you define are stored on the server.

You can enable and disable the session object on a page by page basis. The statement below disables the creation of the SessionID cookie for the page containig the statement.

< @ enablesessionstate = false %>

You can also disable the session object in ISM. If *Session State* is disabled in ISM, you can't re-enable it for a particular page.

You might disable session tracking if you don't use it. This will cause your webpages to load faster and increase trust with your visitors. However, be sure to consider the impact on your server log of disabling session tracking.

Below is an example of a simple password protection scheme using the session object.

```
<% If Request.Form("password") <> "rosebud" Then %>
   <form action="login.asp" method="post">
   Password:<input type="password" name="password">
   <input type="submit" value="Submit">
   </form>
<% Else
   Session("loggedin") = "yes"
   Response.Redirect "homepage.asp"
End If %>
```

The webpage *login.asp* shown above presents the user with a login form. If the user enters an incorrect password the login form is presented again. If the user enters the correct password (rosebud), the session variable "loggedin" is assigned the value "yes", and the user is redirected to homepage.asp.

```
<%
If Session("loggedin") <> "yes" Then
   Response.Redirect "login.asp"
%>
```

The code shown above, placed in the head section each webpage, checks the session variable "loggedin" to verify that the user is logged in. If the session variable "loggedin" does not contain the value "yes", the user is redirected to login.asp.

This is a simple but effective password protection scheme. A more sophisticated password scheme would require a unique password for each user. It might look for the password submitted by the user in a list or in a database, and if found, then assign the value "yes" to the session variable "loggedin".

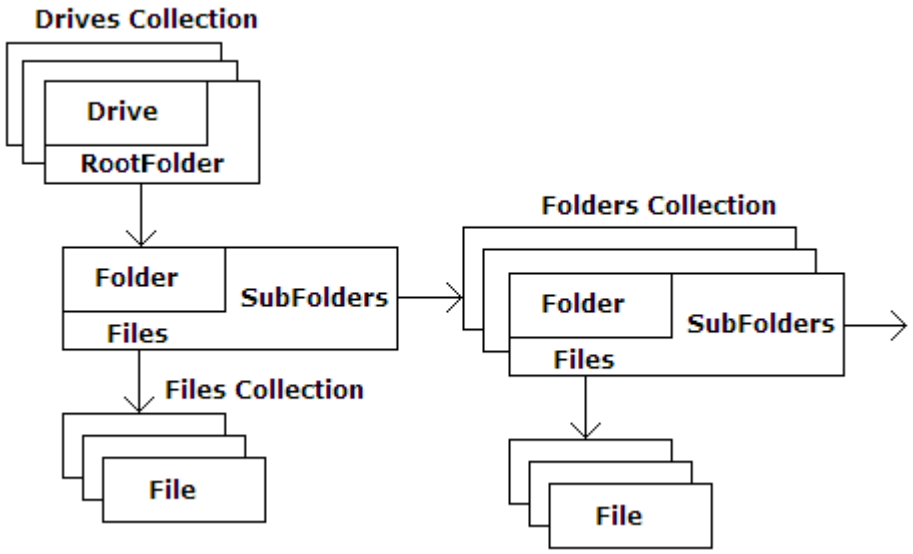
Contents

Working With Drives in ASP

By Stephen Bucaro

When ASP is installed, the FileSystemObject is automatically installed along with it. The FileSystemObject allows you to work with drives, folders, and files on the server.

The FileSystemObject has a collection property called *Drives*. A collection is a list or array of objects of the same type. Each drive in the drives collection has a *RootFolder* object. Each Folder (including the RootFolder) has a *subfolders* collection and a *Files* collection.



In certain instances, you may not be sure that a specific drive exists. In that case, you can use the *DriveExists* method of the FileSystemObject before attempting to access the drive. Example code for this is shown below.

```
<%
Dim objFS

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
If objFS.DriveExists("c:") = true then
   Response.Write("Drive Exists")
Else
   Response.Write("Drive Doesn't Exist")
End If
Set objFS = Nothing
%>
```

The drive that you are attempting to access might be a removable storage unit. In that case, you should check the Drive objects *IsReady* property to make sure the removable media is installed before attempting to access the drive. Example code for this is shown below.

```
<%
Dim objFS, Drv

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Drv = objFS.GetDrive("c:")

If Drv.IsReady = True Then
   Response.Write("Drive Ready")
Else
   Response.Write("Drive Not Ready")
End If
Set Drv =  Nothing
Set objFS = Nothing
%>
```

The FileSystemObject *Drives* collection can be used to get information about the drives in the system. You can use a *For Each object in collection* structure for this purpose. Example code for this is shown below.

```
<%
Dim objFS, Drvs, Drv

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Drvs = objFS.Drives

For Each Drv In Drvs
   Response.Write Drv.DriveLetter & "
"
   If Drv.IsReady = True Then
      Response.Write "Volume Name: " & Drv.VolumeName & "<br>"
      Response.Write "Drive Root: " & Drv.RootFolder & "<br>"
      Response.Write "Drive Type: " & Drv.DriveType & "<br>"
      Response.Write "File System: " & Drv.FileSystem & "<br>"
      Response.Write "Disk Size : " & Drv.TotalSize & "<br>"
      Response.Write "Free Space: " & Drv.FreeSpace & "<br>"
   Else
      Response.Write Drv.DriveLetter & "Drive Not Ready"
   End If
Next
%>
```

A drives *RootFolder* object has a *Subfolders* collection that can be used to get information about the folders on the drive. Example code for this is shown below.

```
<%
Dim objFS, Drv, rootFldr, subFldr

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Drv = objFS.GetDrive("c:")

If Drv.IsReady = True Then
   Set rootFldr = Drv.RootFolder
   For Each subFldr In rootFldr
```

```
      Response.Write "Folder Name: " & subFldr.Name & "<br>"
      Response.Write "Folder Size: " & subFlder.Size & "<br>"
      Response.Write "SubFolders: " & _
        subFlder.Folders.Count & "<br>"
      Response.Write "Files : " & _
        subFlder.Files.Count & "<br>"
   Next
Else
   Response.Write Drv.DriveLetter & "Drive Not Ready"
End If
%>
```

Each *Folder* object, including the RootFolder and Subfolders, has a *Files* collection that can be used to get information about the files in a folder. Example code for this is shown below.

```
<%
Dim objFS, Drv, rootFldr, Fls, Fl

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
Set Drv =  = objFS.GetDrive("c:")
If Drv.IsReady = True Then
   Set rootFldr = Drv.RootFolder
   Set Fls = rootFldr.Files
   For Each Fl in Fls
      Response.Write "Folder Name: " & Fl.Name & "<br>"
      Response.Write "File Size: " & Fl.Size & "<br>"
      Response.Write "Created: " & _
        Fl.DateCreated & "<br>"
      Response.Write "Last Accessed: " & _
        Fl.DateLastAccessed & "<br>"
      Response.Write "Last Modified: " & _
        FL.DateLastModified & "<br>"
   Next
Else
   Response.Write Drv.DriveLetter & "Drive Not Ready"
End If
%>
```

By creatively using each Folder object's *Folders.Count* and *Files.Count* properties, we could display all the folders and files on a drive in a tree structure.

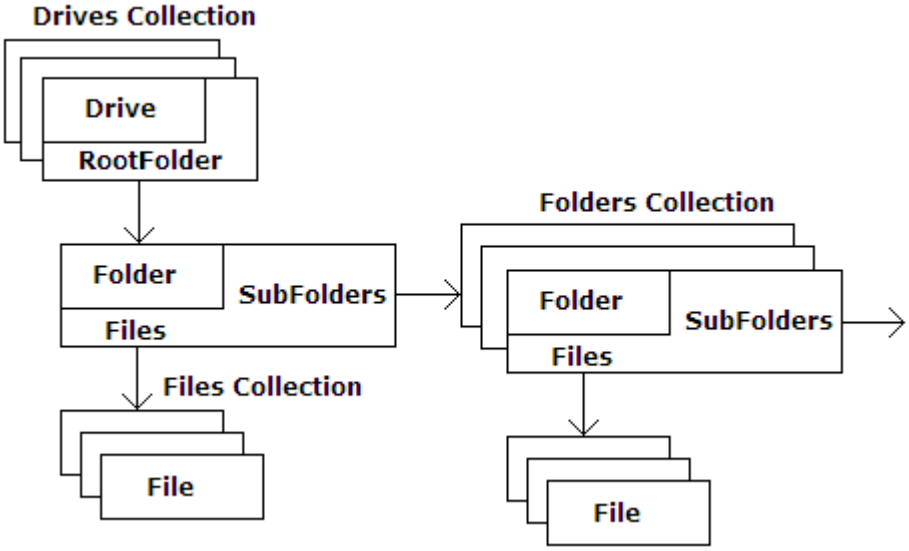
Contents

Working With Folders in ASP

By Stephen Bucaro

When ASP is installed, the FileSystemObject is automatically installed with it. The FileSystemObject allows you to work with drives, folders, and files on the server.

The FileSystemObject has a collection property called *Drives*. A collection is a list or array of objects of the same type. Each drive in the drives collection has a *RootFolder* object. Each Folder (including the RootFolder) has a *subfolders* collection and a *Files* collection.



In certain instances, you may not be sure that a specific drive exists. In that case, you should use the "DriveExists" method of the FileSystemObject before attempting to access the drive. Example code for this is shown below.

```
<%
Dim objFS

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
If objFS.DriveExists("c:") = true then
   Response.Write("Drive Exists")
Else
   Response.Write("Drive Doesn't Exist")
End If
Set objFS = Nothing
%>
```

The drive that you are attempting to access might be a removable storage unit. In that case, you should check the Drive objects *IsReady* property to make sure the removable media is installed before attempting to access the drive. Example code for this is shown below.

```
<%
Dim objFS, Drv

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Drv = objFS.GetDrive("c:")

If Drv.IsReady = True Then
   Response.Write("Drive Ready")
Else
   Response.Write("Drive Not Ready")
End If
```

```
Set Drv =  Nothing
Set objFS = Nothing
%>
```

The FileSystemObject *CreateFolder* method allows you to create a new folder. Before creating a new folder, you should verify that the destination exists. Example code for this is shown below.

```
<%
Dim objFS, Drv, Fldr

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Drv = objFS.GetDrive("c:")
If Drv.IsReady = True Then
   If objFSFolderExists("c:\asp") Then
      set Fldr = objFS.CreateFolder("c:\asp\newfolder")
      Response.Write "New folder created"
   Else
      Response.Write _
         "Destination for new folder does not exist"
   End If
Else
   Response.Write Drv.DriveLetter & "Drive Not Ready"
End If

Set Fldr = Nothing
Set Drv =  Nothing
Set objFS = Nothing
%>
```

The FileSystemObject *CopyFolder* method allows you to copy a folder from one location to another. The format for this method is shown below.

```
object.CopyFolder(source, destination[, overwrite]);
```

Note the optional *overwrite* argument. This is a boolean value that indicates if existing folders are to be overwritten. If false, existing folders are not overwritten. If overwrite is false and the source already exists in the destination, an error occurs. False will also generate an error if attempting to copy a read-only file into a read-only directory. The CopyFolder method stops on the first error, and no roll back of any changes made before the error is made. The default value of overwrite is true.

Example code to copy a folder is shown below.

```
<%
Dim objFS

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
objFS.CopyFolder("c:\foldername\subfldname\*", _
   "c:\newfolder\", False)

Set objFS = Nothing
%>
```

The source path can include wildcard characters in the last path component. If the source path contains wildcard characters or the destination path ends with a \, it is assumed that destination is an existing folder in which to copy matching folders and subfolders. Otherwise, destination is assumed to be the name of a folder to create. Wildcard characters are not allowed in destination path.

The FileSystemObject *MoveFolder* method allows you to move a folder from one location to another. The format for this method is shown below.

```
FileSystemObject.MoveFolder(source, destination)
```

Similar to CopyFolder, the source path can include wildcard characters in the last component. The destination cannot contain wildcard characters. If the source already exists in the destination, an error occurs.

Example code to move a folder is shown below.

```
<%
Dim objFS

set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
objFS.MoveFolder("c:\foldername\subfldname\*", _
   "c:\newfolder\")

set objFS = nothing
%>
```

Each *Folder* object, including the RootFolder and Subfolders, has a *Subfolders* property with a *Folders* collection that can be used to get information about the Subfolders in a folder. Example code for this is shown below.

```
<%
Dim objFS, Fldr, subFldrs, folder

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Fldr = objFS.GetFolder("c:\foldername")
Set subFldrs = Fldr.SubFolders

For Each folder in subFolders
   Response.Write "FolderName: " & folder.Name & "
"
   Response.Write "Attributes: " & folder.Attributes & "
"
   Response.Write "Size: " & folder.Size & "
"
Next

Set subFldrs = Nothing
Set Fldr = Nothing
Set objFS = Nothing
%>
```

Each *Folder* object, including the RootFolder and Subfolders, has a *Files* property with a "File" collection that can be used to get information about the files in a

folder. Example code for this is shown below.

```
<%
Dim objFS, drv, Fldr, Fls, Fl

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
Set Fldr = objFS.GetFolder("c:\foldername")
Set Fls = Fldr.Files

For Each Fl in Fls
   Response.Write "Folder Name: " & Fl.Name & "<br>"
   Response.Write "File Size: " & Fl.Size & "<br>"
   Response.Write "Created: " & Fl.DateCreated & "<br>"
   Response.Write "Last Accessed: " & _
      Fl.DateLastAccessed & "<br>"
   Response.Write "Last Modified: " & _
      Fl.DateLastModified & "<br>"
Next

Set Fls = Nothing
Set Fldr = Nothing
Set objFS = Nothing
%>
```

The FileSystemObject *DeleteFolder* method allows you to delete an existing folder. Before attempting to delete a folder, you should verify that the folder exists. Example code for this is shown below.

```
<%
Dim objFS

Set objFS = _
   Server.CreateObject("Scripting.FileSystemObject")
If objFS.FolderExists("c:\asp\foldername") Then
   objFS.DeleteFolder("c:\asp\foldername")
End If

Set objFS = Nothing
%>
```

By creatively using each Folder object's *Folders.Count* and *Files.Count* properties, you could display all the folders and files on a drive in a tree structure.

Contents

Working With Files in ASP

By Stephen Bucaro

When ASP is installed, the FileSystemObject and the TextStream object are automatically installed along with it. The ASP FileSystemObject allows you to work with drives, folders, and files on the server. The TextStream object allows you to

create, read, and write files.

In certain instances, you may not be sure that a specific file exists. In that case, you should use the FileSystemObject's *FileExists* method before attempting to access the file. Example code for this is shown below.

```
<%
Dim objFS

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
If objFS.FileExists("c:\asp\newfolder\newfile.txt") Then
   Response.Write "File exists"
Else
   Response.Write "File does not exist"
End If

Set objFS = Nothing
%>
```

When you enter a URL into the address bar in your Web browser, it consists of a domain name followed by a file path. The URL file path has no relation to the actual file path on a physical storage device because a website may be located or moved from one physical storage location to another without changing the URL of any file on that Web site. The path you enter into a Web browser is a "virtual path".

The ASP Server object represents the Web server itself and has a method named MapPath that accepts a virtual path and returns the actual physical path. Below is how a call to the MapPath method looks.

```
Server.MapPath(path)
```

For example, you could use the code shown below to determine the actual physical location of the web server.

```
<%
Dim strPath

strPath = Server.MapPath("/")
response.write(strPath)
%>
```

The resulting response from the above code might be "c:\inetpub\wwwroot". Note that you don't need to create the Server object because if the Web server is running, the server object already exists.

Generally, you would use the MapPath method to determine the actual physical path of a file from its virtual path, as shown below.

```
<%
Dim strPath

strPath = Server.MapPath("/foldername/filename.asp")
response.write(strPath)
%>
```

If the path argument doesn't start with a slash (/ or \), MapPath assumes that it is a path relative to the .asp file being processed. For example, if an .asp file located at the physical path c:\inetpub\wwwroot\myfolder\myfile1.asp calls the MapPath method with the code shown below.

```
<%
Dim strPath

strPath = Server.MapPath("../foldername/filename.asp")
response.write(strPath)
%>
```

The resulting response would be "c:\inetpub\wwwroot\foldername\filename.asp".

Use the FileSystemObject's OpenTextFile method to work with files. The format for the OpenTextFile method is shown below.

FileSystemObject.OpenTextFile(filename, mode, create, format)

mode

1 = ForReading - Open file for reading.
2 = ForWriting - Open file for writing. Overwrite any data currently in the file.
8 = ForAppending - Open file for writing. Append new data to end of the file.

create (optional)

True = If file does not exist, create it.

False = If file does not exist, do not create it.

The default is False.

format (optional)

0 = Open the file as ASCII.
-1 = Open the file as Unicode.
-2 = Open the file using the system default.

The default is ASCII.

The OpenTextFile method creates a TextStream object with the properties you have configured. After you have created a TextStream object, you can use its methods as shown below.

Read(characters) = Read the specified number of characters from the file.
ReadLine = Read a line from the file excluding the line break.
ReadAll = Read the entire file.
Write(string) = Write a string to the file.
WriteLine(string) = Write a string to the file and append a line break.
WriteBlankLines(lines) = Write the specified number of blank lines (line breaks).
Skip(characters) = Move the file pointer past the specified number of characters in the file.
SkipLine = Move the file pointer past the next line break.

close = Close the file.

The TextStream object has the following properties which you can use to control the file reading or writing process.

AtEndOfLine = Returns True when file pointer is at a line break.
AtEndOfStream = Returns True when file pointer is at the end of the file.
Column = The current character position of the file pointer.
Line = The current line number of the file pointer.

The most common method used to read an existing file is shown below.

```
<%
Dim objFS, strPath, txtFile, line

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
strPath = Server.MapPath("../foldername/filename.txt")
Set txtFile = objFS.OpenTextFile(strPath,1,0)

Do
   line =  txtFile.ReadLine
   Response.Write line
Loop Until txtFile.AtEndOfStream

txtFile.Close
Set objFS = Nothing
%>
```

Note that the MapPath method to retrieve the physical path to the file. The actual physical path of the file could have been passed to the OpenTextFile method, but then if the physical location of the website changes, the code breaks. Using MapPath allows the website to be moved to another physical storage location without breaking the code.

The TextStream object allows you to create, read, and write files. The code shown below demonstrates how create and write to a text file.

```
<%
Dim objFS, txtFile

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
Set txtFile =
   objFS.CreateTextFile("c:\asp\newfolder\newfile.txt")
txtFile.Write("This is the text for my new file.")
txtFile.Close
Set txtFile = Nothing
Set objFS = Nothing
%>
```

The format for the CreateTextFile method is shown below.

```
TextStreamObject = FileSystemObject.CreateTextFile(filename[, overwrite[,
unicode]])
```

overwrite - optional Boolean value. If overwrite is set to true, and the file already exists, the file will be overwritten. If overwrite is set to false and the file already exists, you will receive an error. The default value for the overwrite parameter is true. To avoid an error, you should check if the file already exists before attempting to create it.

force - optional Boolean value. If unicode is set to True, the file is created as a Unicode file. If unicode is set to false, the file is created as an ASCII file. The default value for the force parameter is false.

Note that, after creating the text file, you don't have to open it for writing, it's already open.

If the file already exists, use the FileSystemObject's OpenTextFile method to open the file for writing, as shown below.

```
<%
Dim objFS, txtFile

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
Set txtFile =
   objFS.OpenTextFile("c:\asp\newfolder\newfile.txt",8,0)
txtFile.Write("Append this text to the file")
txtFile.Close
Set txtFile = Nothing
Set objFS = Nothing
%>
```

The second parameter of the OpenTextFile method, after the file path, is the I/O Mode.

8 Open for appending
1 Open for reading

The third parameter is the Format.

0 Open as ASCII file
-1 Open as Unicode file
-2 Open using the system default

The TextStream object has three methods for writing.

Write - writes a string of characters to the file
WriteLine - writes a string of characters to the file, placing a newline character after the last character.
WriteBlankLines - writes newline characters to the file. Pass the number of newline characters to be written as an argument, ie. TextStream.WriteBlankLines(3)

To create a copy of a file in a different folder, use the FileSystemObject's CopyFile method as shown below.

```
<%
Dim objFS
```

```
Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
objFS.CopyFile("c:\sourcefolder\filename.txt", _
   "c:\destfolder\", true)
Set objFS = Nothing
%>
```

The format for the CopyFile method is shown below.

FileSystemObject.CopyFile(source, destination[, overwrite])

overwrite - optional Boolean value. If overwrite is set to true, and the file already exists, the file will be overwritten. If overwrite is set to false and the file already exists, you will receive an error. The default value for the overwrite parameter is true. To avoid an error, you should check if the file already exists before attempting to create it. If the destination folder is read-only, the CopyFile method will fail.

To move a file to a different folder use the FileSystemObject's MoveFile method as shown below.

```
<%
Dim objFS

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
objFS.MoveFile("c:\web\*.gif","c:\images\")
set objFS = Nothing
%>
```

The format for the MoveFile method is shown below.

FileSystemObject.MoveFileI(source, destination)

The source file path can contain wildcard characters in the last component. If the destination is an existing file an error occurs.

To delete a file use the FileSystemObject's DeleteFile method as shown below.

```
<%
Dim objFS

Set objFS =
   Server.CreateObject("Scripting.FileSystemObject")
objFS.DeleteFile("c:\sourcefolder\filename.txt",true)
Set objFS = Nothing
%>
```

The format for this method is shown below.

DeleteFile(filepath[, force])

force - optional boolean that determines whether read-only files will be deleted, a true value means read-only files will be deleted, a false value means they will not be deleted. The default is false. You can Use the wildcard character (*) to delete

multiple files.

Contents

Sending Email With ASP

By Stephen Bucaro

Sending email with ASP (Active Server Pages) is not difficult. You can use ASP email to create a contact form, support request form, or even perform mass emailings. CDO (Collaboration Data Objects) is the Microsoft technology that simplifies the creation of email applications.

SMTP (Simple Mail Transfer Protocol) is a service installed with IIS that allows you to send email from an ASP webpage. This article assumes SMTP is properly installed and configured on your server.

• Microsoft's previous technology to send email was CDONTS (Collaboration Data Objects for New Technology Server). With Windows 2000 server, CDONTS was replaced with CDO. CDO which provides the ability to send messages through a remote SMTP (Simple Mail Transfer Protocol) server.

Windows 2003 server doesn't install CDONTS by default, therefore if you used CDONTS in your ASP applications, you should update the code to use CDO. You can use CDO (also referred to as CDOSYS) in ASP and ASP.NET applications.

Sending a Plain Text Email

The script below demonstrates how to send a simple pre-defined plain text email message.

```
<%
Set objMail. = CreateObject("CDO.Message")

objMail.Subject = "Sending email with CDO"
objMail.From = "mymail@yourdomain.com"
myMail.To = "someone@domain.com"
objMail.TextBody = "This is a test message."
objMail.Send

set objMail. = Nothing
Response.Write "Message Sent"
%>
```

To try the example, paste the script into a file on your ASP Web server named, for example *cdotest.asp*. Edit the *From* and *To* email addresses, then execute the script.

An ASP Email Contact Form

One of the most common uses of ASP email is to create a Contact Form for your Web site through which visitors can provide comments or ask questions. Below is an example of an ASP Web page for a Contact form.

```
<html>
<head>
<title>Email Contact Form</title>
</head>
<body>

<%
Dim strName, strEmail, strMessage

strName = Trim(Request.Form("sender"))
strEmail = Trim(Request.Form("email"))
strMessage = Trim(Request.Form("comment"))

If(strMessage <> "") Then
   Set objMail = CreateObject("CDO.Message")

   objMail.From = strEmail
   objMail.To = "mymail@yourdomain.com"
   objMail.Subject = "Contact Form from " & strName
   objMail.TextBody = strMessage
   objMail.Send

   Set objMail = Nothing

   Response.Write "Thank You"
Else
%>

Email Form<br />
<form method="post">
Name: <input type="text" name="sender" size=40></input><br />
Email: <input type="text" name="email" size=40></input><br />
Comment:<br />
<textarea name="comment" cols=40 rows=5></textarea><br />
<input type="submit" value="Submit"></input>
</form>

<% End If %>
</body>
</html>
```

Paste this script into a file on your Web server named, for example *contact.asp*. Edit the To email address to the address where you want to receive the contact form messages. You can also edit the text "Thank You" which is the message the user receives after they click the [Submit] button.

Sending HTML Email

Today most email messages are formatted and use a font chosen by the writer, some may even contain images and/or other graphic elements. This is accomplished by using html in the email message. Use the *CDO.Message* object's *HTMLBody* property to define an html message, as shown below.

```
<%
Set objMail = CreateObject("CDO.Message")
```

```
objMail.Subject = "Sending html Email with ASP"
objMail.From = "from@domain.com"
objMail.To = "to@domain.com"
objMail.HTMLBody = "<p>This is an html email</p><img
src='http://domain.com/imagename.gif'>"
objMail.Send

set objMail = nothing

Response.Write "Message Sent"
%>
```

In the example above, note how the html assigned to the *HTMLBody* property contains code to load an image.

Sending an Attachment

An attachment is a document that you send along with an email, but not as part of the message's text. Use the *CDO.Message* object's *AddAttachment* property to specify the document to attach to the message, as shown below.

```
<%
Set objMail = CreateObject("CDO.Message")

objMail.Subject = "Sending ASP Email with Attachment"
objMail.From = "from@domain.com"
objMail.To = "to@domain.com"
objMail.TextBody = "See Attachment."
objMail.AddAttachment "http://domain.com/filename.ext"
objMail.Send

set objMail = nothing

Response.Write "Message Sent"
%>
```

Sending ASP Email with CC and/or Bcc

To "CC" someone means to "Carbon Copy" them. Of course in an email message we don't use carbon paper, we simply send a copy of the same message to each email address in the CC field.

When you use CC, everyone on the CC list can see everyone else's email address. Some people may not appreciate you publishing their email address. It's often better to use the BCC (Blind Carbon Copy) field, where the other recipients email address are not visible.

Use the *CDO.Message* object's Bcc property to list the email addresses to which copies of the message will be sent, as shown below.

```
<%
Dim aRecipients = Array("name1@domain.com","name2@domain.com")

Set objMail = CreateObject("CDO.Message")

objMail.Subject = "Sending ASP email to a list"
objMail.From = "from@domain.com"
```

```
objMail.To = "to@domain.com"
objMail.Bcc = Join(aRecipients,":")
objMail.TextBody = "This is a message."
objMail.Send

set objMail = nothing

Response.Write "Message Sent"
%>
```

In a CC or BCC list the recipients email addresses are separated by semicolons. In the example above, the recipients email addresses are listed in an array, and the *Join* function is used to create a semicolon separated list from the array which is then assigned to the *CDO.Message* object's *Bcc* property.

Redirecting to a Different Webpage

In the previous examples, after the email was sent we displayed "Message Sent" on the webpage. Instead, you can redirect the user to another webpage as shown below.

```
<%
Set objMail. = CreateObject("CDO.Message")

objMail.Subject = "Sending email with CDO"
objMail.From = "mymail@yourdomain.com"
myMail.To = "someone@domain.com"
objMail.TextBody = "This is a test message."
objMail.Send

set objMail. = Nothing

Response.Redirect("thankyou.asp")
%>
```

This technique can be used to present the user with, a list of newsletters to which they can subscribe, a list of new articles posted on your Web site, your site map, or many other purposes.

This article reviews basic methods of sending email With ASP. To these examples you might want to add code to validate the users form entries, and to handle a possible server error. The examples demonstrate that sending email with ASP is not difficult.

Contents

Working With Cookies in ASP

By Stephen Bucaro

Cookies are small text files that are placed in a visitor's Web browser's cookie cache by a Web server when someone visits a Web site. By using a cookie the Web server can store and retrieve information about the visitor between pages. The cookie may store information about what pages the visitor viewed or what

links the visitor clicked on while they visited the Web site.

In Windows, cookies are stored in:

```
C:\users\username\AppData\Local\Microsoft\Windows\Temporary Internet Files
```

You can view the cookie files in Internet Explorer 7 by selecting *Internet Options* in the *Tools* menu and on the *General* tab, in the *Browsing History* section, click on the [Settings] button. In the *Temporary Internet Files* dialog box that appears, click on the the [View Files] button. Explorer will open, showing you the Internet Explorer temporary files, including the cookies.

Some cookies are removed from the visitors computer when they leave the Web site, these are called *session* cookies. Other cookies stay in the visitors browser cookie cache until an *expiration date* specified by the Web site that deposited the cookie.

Cookie Reliability

Cookies are generally used to personalize Web sites in order to make the visitors experience more productive and enjoyable. However, some people don't want any information about them saved. For this reason some people have cookies disabled in their browser. A user can delete their Temporary Internet Files and cookies any time they feel like it. The cookie cache can hold only a limited number of cookies. When the cache is full and a new cookie arrives, the oldest cookies are deleted to make room for the new cookie.

Storing a Cookie

Cookies are sent to the visitor's browser using the ASP *Response* object. The code for stroring a cookie is shown below.

```
Response.Cookies("cookiename") = "cookievalue"
```

Note that the cookie specified above contains no expiration date, this means it's a session cookie. The code for storing a cookie with an expiration date is shown below.

```
Response.Cookies("cookiename") = "cookievalue"
Response.Cookies("cookiename").Expires = "July 10,2012"
```

Note that the cookie specified above sets an explicit expiration date. Generally a cookie's expiration date is set by adding a predetermined amount of time to the current date. The code for this is shown below.

```
Response.Cookies("cookiename") = "cookievalue"
Response.Cookies("cookiename").Expires = DateAdd("m",1,Date())
```

The VBScript *DateAdd* function returns a date to which a specified amount of time has been added. The "m" in the code above means the time units to be added will be months ("d" would mean days, "h" would mean hours, "n" would mean minutes). The 1 in the code above means 1 time unit (in this 1 month) should be added. The VBScript *Date* function returns the current date.

Note that if you were specifying an interval of less than 1 day, you would replace

the *Date* function with the *Now*) function, which returns the current date and time.

Retrieving a Cookie

Cookies are retrieved from the visitors browser using the ASP *Request* object. To retrieve a cookie, use the ASP "Request" object providing the cookie's name. The code for retrieving a cookie is shown below.

```
Request.Cookies("cookiename")
```

Create an ASP web page that stores a cookie named "myCookie" with an expiration interval of 10 minutes. Then create an ASP web page with the code shown below to retrieve that cookie and display its name.

```
<%
Dim cookiename
cookiename = Request.Cookies("myCookie")
response.write("My cookies name is: " & cookiename)
%>
```

Storing Multiple Values in a Cookie

The *Response* object's *Cookies* collection allows you to store multiple values in a single cookie. The code for storing multiple values in a cookie is shown below.

```
Response.Cookies("cookiename")("valuename1") = "value1"
Response.Cookies("cookiename")("valuename2") = "value2"
Response.Cookies("cookiename")("valuename3") = "value3"
```

There is alimit to how much information you store in a cookie. Most browsers support cookies of up to 4096 bytes.

Retrieving Multiple Values From a Cookie

To read multiple values from a cookie, use the ASP *Request* object providing the cookie's name and the name of the value to retrieve. The code for retrieving multiple values from a cookie is shown below.

```
Request.Cookies("cookiename")("valuename1")
Request.Cookies("cookiename")("valuename2")
Request.Cookies("cookiename")("valuename3")
```

How can you determine if a cookie has more than one value? If a cookie contains multiple values, each value needs an identifying name, called a "key". The Cookies collection has a property named *HasKeys* that is set to *False* if the cookie has no keys and is set to *True* if the cookie does have keys.

```
Dim keyname
If Request.Cookies("cookiename").HasKeys Then
   For Each key in Request.Cookies(keyname)
      Response.Write(keyname & "=" & Request.Cookies("cookiename")
(keyname))
   Next
Else
   Response.Write Request.Cookies("cookiename")
```

```
End If
```

Setting Other Cookie Properties

By default a cookie stores the domain name, path, and security information of the Web site that stored the cookie. A cookie can be read only by the domain and path (and its subdirectories) stored in the cookie. However, you can explicitly set the domain name stored in the cookie to a different domain name. By default the path property is set to the root of the domain, allowing the cookie to be read by any subfolder in that domain. However for security purposes, you may want to explicitly set the path to a specific sub folder.

```
Response.Cookies("cookiename").Domain = "domain-name.com"
Response.Cookies("cookiename").Path = "/subfolder-name/"
Response.Cookies("cookiename").Secure = False
```

By default the *Secure* property is set to *False*. If the *Secure* property is set to *True*, the cookie can only be stored or retrieved if the browser is using secure sockets or https:// to connect.

Modifying a Cookie

You cannot modify a cookie stored on the visitor's computer. To modify a cookie you have to create a new cookie with the same name (with modified values) and then send the new cookie to the visitor's browser to overwrite the old cookie.

Deleting a Cookie

You cannot delete a cookie stored on the visitor's computer. To delete a cookie you have to create a new cookie with the same name (but with the cookie's expiration set to a date in the past) and then send the new cookie to the visitor's browser to overwrite the old cookie. The browser will check the cookie's expiration and will delete the cookie.

Modifying or Deleting Subkeys

To delete an individual subkey, you can manipulate the cookie's *Values* collection, which holds the subkeys. First recreate the cookie by getting it from the Cookies object. Then call the *Remove* method of the *Values* collection, passing the name of the subkey to delete. Then add the cookie to the Cookies collection so it will be sent in its modified form back to the browser.

Checking If the Visitor's Computer Accepts Cookies

To determine if the visitor's computer has cookies disabled you would need to send two Web pages, a first page to set a cookie, and a second page to read that cookie. However by using a little client-side Java Script you can do it with a single Web page. The code for performing a cookie test with a single Web page is shown below.

```
<%
Response.Cookies("testcookie") = "testcookie"
%>
<head>
```

```
<script type="text/javascript">
function cookieTest()
{
  var strCookie = document.cookie;
  var found = false;
  var loc1, loc2;
  var i = 0;

  while(i <= strCookie.length)
  {
    loc1 = i;
    loc2 = loc1 + 6;
    if(strCookie.substring(loc1,loc2) == "testcookie")
    {
        found = true;
        break;
    }
    i++;
 }

 if(!found)
 {
   document.location = "login.asp";
 }
}
</script>
<noscript>
  <meta http-equiv="Refresh" content="login.asp">
</noscript>

</head>
<body onload="cookieTest()">
```

The first line in the code is where the ASP *Response* object sets the cookie, in this case the cookie's name is "testcookie". The *onload* event in the <body> tag is executed after the Web page completes loading. The *onload* event calls a Java Script function, named "cookietest", which is defined in the <head> section of the Web page.

The "cookietest" function looks for a cookie named "testcookie", and if it doesn't find it, it redirects the browser to another Web page. I've also included a <noscript> code block which uses a meta tag refresh to redirect the browser to another Web page if the user has Java Script disabled.

Contents

Handling ASP Errors

By Stephen Bucaro

ASP will let you know an error occurs, usually by displaying the message "The page cannot be displayed" along with the code "HTTP 500 - Internal Server Error -

ASP Error - Internet Information Sever" and a terse message giving some usually not too helpful information about what caused the error. This is NOT the type of message you want to send to your Web site's visitor.

ASP errors are basically of two types. The first type is "syntax" errors or "logic" errors caused by the programmer. The second type "run-time" errors generated by the system. Examples of system errors are a file or database that ASP can't perform an operation on because of low system resources of a loss of connectivity.

There are two ways to respond to an error. The first way is to just let them happen and expose your Web site's visitors to the default error message, or by "Handling" the error. Handling the error involves determining where an error might occur in your code, for example in code that accesses a file or database, and adding code to do something about the error.

Things you might do about an error are display a friendly error message to the user, perform any clean up tasks required by the error, log the error, or notify the webmaster of the error.

There are two ways to handle an error. The first way is to use VBScript's "On Error Resume Next" statement. Below is an example of using the "On Error" statement.

```
<%
Set cn = CreateObject("ADODB.Connection")

On Error Resume Next
cn.Provider = "sas.localprovider.1"
cn.Properties("Data Source") = "c:\testdata"
cn.Open
cn.Close
%>
```

This code creates a database connection, configures the connection, and then opens and closes the connection. If an error occurs when attempting to configure and use the database connection, the "On Error Resume Next" will cause execution of the code to continue on the next line after the line that caused the error.

The "On Error Resume Next" stops the default error message from appearing and allows the webpage to continue execution, but it doesn't do anything about the error.

The second way to handle an error is to use the *Err* object. The properties and methods of the Err object are listed below.

Properties

.Number
.Description
.Source

Methods

.Clear
.Raise

The properties of the Err object are automatically set when an error occurs.

When no error occurs, the Err object's *Number* property is 0. When an error occurs, the Err object's *Number* property is set to the error's number, and the error *Description* and *Source* properties are also set. To test for an error you can check the Err object's *Number* property, as shown below.

```
<%
On Error Resume Next
   ' possible error causing statement
If Err.Number <> 0 Then
Response.Write "Error number:" & Err.Number & _
   "Error Description:" & Err.Description
End If
%>
```

Note that you still need the "On Error Resume Next" statement to turn on error handling. This code merely displays information about the error provided by the Err object. The code shown below redirects the user to another webpage which might politely explain that an error has occurred.

```
<%
On Error Resume Next
   ' possible error causing statement
If Err.Number <> 0 Then
   response.redirect "errornote.asp"
Else
  ' continue processing
End If
%>
```

The Err Object's *Clear* method can be used to reset the Err object allowing the code to continue executing. In the example shown below, division by zero would cause error number 11. The code checks for that error number and if found, clears the error and assigns the value 1 to the denominator.

```
<%
On Error Resume Next
   amount = numerator / denominator
If Err.Number = 11 Then
   Err.Clear
   demoninator = 1
   amount = numerator / denominator
End If
%>
```

If you want to use this technique to handle an error, it's important to use the Err objects *Clear* method to reset the error, otherwise, the information from the error will remain in the Err object and if you check the error object again you'll get false information.

The Err Object's *Raise* method can be used to artificially create an error if you want to test how your code would respond to such an error, as shown below.

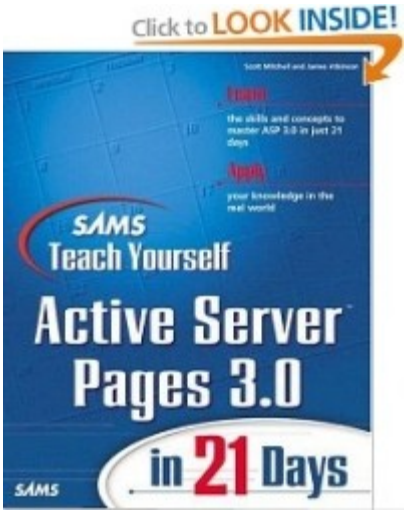
```
<%
```

```
On Error Resume Next
   amount = numerator / denominator
   Err.Raise 11
If Err.Number = 11 Then
   Err.Clear
   demoninator = 1
   amount = numerator / denominator
End If
%>
```

Even if your code is perfect, "run-time" errors can still be generated by the system. If you don’t properly handle the error, the message "The page cannot be displayed" will be sent to your Web site's visitor. To avoid that, use the "On Error Resume Next" statement and the Err Object to handle errors.

Contents

Sams Teach Yourself Active Server Pages 3.0 in 21 Days



The friendly, tutorial style of Sams Teach Yourself Active Server Pages 3.0 in 21 Days empowers you to create your own Active Server Pages quickly and easily. Using client-proven methods, and his award-winning advice to Web developers from around the world, ASP master Scott Mitchell provides you with an understanding of ASP and IIS fundamentals, and guides you through the use of VBScript and ASP's built-in objects, enabling you to create your own dynamic, database-driven Web solutions. You'll benefit from Scott's expert knowledge of topics including creating dynamic content, interacting with the user, reading and writing files on the Web server, creating personalized content with cookies, reading a database using ASP, and debugging your ASP scripts.

Contents

- Working with Objects
- Using the Response Object
- Communicating with the User
- Collecting the Form Information
- Working with the Request Object
- Maintaining Persistant Information on the Web
- Working with Common ASP Components
- Reading and Writing Files on the Web Server
- Debugging Your ASP SCripts and Handling Errors
- Using Databases
- Reading from a Database Using ASP
- Inserting, Updating, and Deleting Database Records
- Examining the Recordset Object
- Using SQL Statements to Query Data
- Using Advanced Database Techniques
- Practicing Intelligent Application Design
- VBScript Reference
- Built-in ASP Objects Reference

Johan Steen of Gothenburg, Sweden says,"I picked up the book a few days ago, and from never have used ASP before, I finished the book in 3 days (I was very eager to get going and was short of time to finish what I wanted to do), and now I easily put together very advanced dynamic userbased websites with connections to databases. Needless to say that I finished my project on time. The book was excellent written and all the information was easy to understand and to learn immediately. I can't say it enough, one of the best programming books I've read since I started programming way back in the 80's on the Amiga. A huge thanks to the authors, it was a literally a lifesaver!"

Click Here for more information.

Contents

Useful Resources

Many of the good ASP resources on the Web have totally converted over to ASP.NET, but there are still a few useful resources for clasic ASP.

bucarotechelp.com Find information about ASP and other programming languages, also a Java Script quick reference and a Cascading Style Sheets (CSS) quick reference, plus tons of cut-and-paste code for your Web site, also tons of other useful information about careers, how to make money on the Web, and just about anything else you can think of.

MSDN Library MicroSoft Developers Network Library. This link goes to the Active Server Pages of the MSDN, where you can find Technical Articles, Code Samples, Tutorials, Active Server Pages Reference, and just about everything else related to developing with Microsoft's products.

W3C's Schools World Wide Consortium's Schools Web site where you can learn ASP, HTML, XML, CSS, and just about any other Web techenology that you can think of. It's all written very clearly in plain simple language, and it's all free.

aspfree.com Lots of useful information, articles about ASP, ASP tutorials, ASP tips and tricks. Make sure your PC's speaker is turned all the way down and your popup blocker enabled to ward off the abusive advertising by this site.

Contents

Visit Bucaro Techelp to download FREE ebooks including Bucaro TecHelp s popular PC Tech Toolkit. Read Bucaro TecHelp's famous Easy Java Script and Easy CSS tutorials with cut-and-paste code. Learn Basic PC Anatomy and where to find FREE diagnostic Tools and technical assistance. Learn how to start your own online business, including many examples of people who started successful businesses.

To receive an email notification when new articles, ebooks, clipart, graphics, or other content has been added to Bucaro Techelp, Click Here to subscribe to Bucaro TecHelp News Feed Notification.

Contents