THE MINISTRY OF SCIENCE AND TECHNOLOGY

THE CORPORATION FOR FINANCING AND PROMOTING TECHNOLOGY

STANDARD

ASP Coding Convention

Code	09ce-HD/PM/HDCV/FSOFT
Version	1/0
Effective date	15/10/2003

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Purpose

Code conventions are important to programmers for a number of reasons:

- 80% of the lifetime cost of a piece of software goes to maintenance.
- Hardly any software is maintained for its whole life by the original author.
- Code conventions improve the readability of the software, allowing engineers to understand new code more quickly and thoroughly.

If you ship your source code as a product, you need to make sure it is as well packaged and clean as any other product you create.

The following suggested conventions apply to the development of ASP scripts written in Microsoft Visual Basic Scripting Edition (VBScript) and are designed to enhance consistency, performance, and reliability.

1.2. Application scope

Fsoft projects

1.3. Related documents

No.	Code	Name of documents
1	04e-QT/PM/HDCV/FSOFT	Process description: Coding

1.4. Definition

Terminology	Explanation

2. LAYOUT ORDER OF SCRIPTS IN ASP PAGES

The following list summarizes the layout of VBScript scripts in an .asp file, in order from top to bottom.

- Specify language.
- Use Option Explicit statement.
- List function library includes.
- Declare page-scoped variables.
- Assign values to page-scoped variables.
- Code HTML and in-line scripting.
- List functions called by in-line scripts.

The first two script statements should specify declaratives and Option Explicit to force explicit variable declaration.

Inline scripts with HTML markup should call objects of functions placed at the bottom of the page. The inline scripts should be short and readable, even if this requires splitting out procedures to be placed at the bottom of the page.

3. FILENAME EXTENSION STANDARDS

This section suggests conventions for standardizing on filename extensions, accounting for the types of files containing scripts, or HTML and scripts (VBScript or JScript) together.

3.1. *Extensions for Page Files*

Standards:

- .asp: for pages containing ASP scripts
- .htm:for straight HTML pages

You must use the .asp extension for pages that contain ASP scripts. To save time and resources when serving pages, use the .htm extension for files that don't require server-side script execution.

3.2. *Extensions for Included Files*

Standards:

- .inc: for large amounts of data, with scripting, to be included in a page
- .txt: for text-formatted data files, without scripting, to be included in a page

For consistency, use include files to make specific information available to more than one referring page (changes to include files are distributed to all the pages that include them).

4. PROGRAMMING STRUCTURE

4.1. *Comments in Scripts*

Write consistent comment blocks near the top of each page listing, such as the filename, the group developing the file (not the person--e-mail should go to a group alias), the date developed, the HTML standard, and clear descriptions of all changes made.

Scripts that are commented out should be deleted unless they are placeholders, in which case they should be labeled as such.

Use comments to explain obscure or complex code. Do not leave a phrase such as the following without a comment:

```
   If Err = -214983642 Then
```

Use comments to explain empty statements used as placeholders, such as:

```
If...Then...Else...Endif
```

4.2. *Constant Names*

To clearly distinguish constants from other elements, use all uppercase when naming them. An underscore can be used to separate terms when necessary. Example:

```
Const MIN_QUAL = 25
```

4.3. *Minimize Context Switching*

For readability, try to minimize context-switching between HTML and scripts. When possible, use a few large blocks of script on a page instead of several scattered fragments.

4.4. *Dictionary Object: Speed Access to Information*

The VBScript Dictionary object enables fast lookup of arbitrary key/data pairs. Because the Dictionary object gives you access to array items by key, it is faster for finding items that are not in contiguous memory since you are specifying a key, rather than knowing where in the array the item is located.

Use the Dictionary object when you have set a high priority on fast searches of nonlinear data.

4.5. Delimiting Lines of Script for Readability

Lines of script should be blocked between a pair of delimiters, rather than written with delimiters on each line.

Instead of this:

```
<% strEmail = Session("Email") %>

<% strFirstName = Request.Form ("FirstName") %>

<% strLastName = Request.Form ("LastName") %>
```

Do this:

```
<%

 strEmail = Session("Email")

 strFirstName = Request.Form ("FirstName")

 strLastName = Request.Form ("LastName")

%>
```

For a single stand-alone line of script, keep the delimiters on the same line as the script. Example:

```
<% strEmail = Session("Email") %>
```

If the line of script consists of an equal sign and a variable, make the equal sign part of the delimiter. Example:

```
<%= strSubscrLName %>
```

Similarly, if the line of script specifies an ASP declarative, include the @ sign as part of the opening delimiter. Example:

```
<%@ LANGUAGE = VBScript %>
```

4.6. Enable Session State Directive

Using the Enable Session State directive (set in Internet Services Manager) for your site enables the detailed tracking of user requests.

Save the resources used by IIS to process scripts for pages not using session state information by setting the Enable Session State directive to FALSE for those pages:

```
<%@ ENABLESESSIONSTATE = False %>
```

4.7. Language Default

Override the scripting language being used as the server's default language (VBScript) by using the @ language directive. For example, to ensure that the default language is JScript, put the following code at the top of each ASP page:

```
<%@ LANGUAGE = JScript %>
```

4.8. Blank Lines in Files

In general, use blank lines sparingly. You should remove all blank lines at the beginnings and ends of files, but may want to include one blank line between statements, to increase readability.

4.9. Object and Procedure Call Names

To clearly distinguish object names and procedure calls from elements such as variables, begin each object name or procedure call with a verb. Use initial capitalization for each term within an object name or procedure call. The following list suggests some naming conventions that could be used to name objects for some typical activities.

Name	**Function**	**Example**
Add New	add new records	**Customer.AddNew**
Update	edit records	**Customer.Update**
Remove	delete records	**Customer.Remove**
Get	return row from database	**Customer.GetNewest**
List	return multiple rows	**Customer.GetNew**

To limit criteria for methods that return information, use From and For with a method or function name.

Name	**Function**	**Example**
GetFor	returns criteria-based row	**Customer.GetForName**

Name	Function	Example
ListFor	returns criteria-based multiple rows	**Customer.ListForPeriod**

4.10. Object Naming

As with variables and statements, used mixed case in naming objects, and in spelling out names of intrinsic objects. Capitalize the first letter of each term.

Use descriptive names for objects, even though this requires more characters per name.

The following example conforms to this convention (cnn prefixes an ADO connection variable, see "Variable Names" below):

```
Set cnnSQL = Server.CreateObject("ADODB.Connection")Set

cnnSQLServer = Server.CreateObject("ADODB.Connection")
```

These do not conform:

```
Set cnnS = Server.CreateObject("ADODB.Connection")

Set cnnsql = Server.CreateObject("ADODB.Connection")
```

4.11. Paths, using MapPath

Consider using MapPath instead of literal paths in ASP applications. The ASP Server.MapPath method allows you to physically relocate an ASP application without recoding scripts. This saves program modification and maintenance effort.

Performance is affected slightly, because every time you use MapPath in a script, IIS must retrieve the current server path. Consider placing the result of the method call in an application variable to avoid retrieving the server path.

4.12. Select Case Statement for Readability

For readability and efficiency, use the Select Case statement in place of If...Else to repeatedly check for the same variable for different values. For example:

```
Select Case intYrsService

  Case 0 to 5

    strMessage = "You have ten days paid leave this year."
  Case 6 to 15

    strMessage = "You have fifteen days paid leave this year."

  Case 15 to 30

    strMessage = "You have twenty days paid leave this year."

  Case 30 to 100

    strMessage = "Will you never leave?"

End Select
```

4.13. Spaces in Scripts

To enhance script readability, use spaces consistently:

- Before and after operators, such as plus and equal.
- After commas, as when passing parameters.
- Exception: do not use spaces between arguments in ADO connection strings.

4.14. Statement Styles

Each scripting language has its own conventions for capitalization, indentation, and other style-related characteristics. Since VBScript is case-insensitive, capitalization conventions can be created to improve readability, as the following suggestions illustrate.

If...Then...Else...End If statements:

- Capitalize the first letter of If, Then, Else, and End If.
- Indent the clauses two spaces, or one tab.
- Put spaces at each end of an equal sign.
- Avoid using unnecessary parentheses.

Correct example:

```
<%

If Request("FName") = "" Then

  Response.Clear
  Response.Redirect "test.html"

Else

  Response.Write Request("FName")

End If

%>
```

Similarly, capitalize the first letters of function and subroutine statements, and indent their definitions 2 spaces. Example:

```
Sub SessionOnStart

 Set Session("MyId") = Request.ServerVariables(...)

End Sub
```

As with variable names, use mixed case, capitalizing the first letter in each term in a name. Avoid underscores. Example:

```
FirstName
```

4.15. String Concatenation

For the sake of consistency and to achieve more self-documenting scripts, use the string concatenation (&), instead of a plus (+) sign.

4.16. String Function

Use the String() function to create a string consisting of repeated characters. Example: a string of non-breaking spaces (&nbsp;). The String() function is more readable and elegant than, for example:

```
For I = 0 to 11 . . .
```

4.17. Case Values

Keep case consistent in both variable assignment and logical tests by using UCase() or LCase(). This is especially important when assigning and logically testing HTML intrinsic form controls, such as check boxes and radio buttons.

4.18. Trimming Values

Be consistent in trimming values. For example, consider always trimming values before putting them in state. This will eliminate errors in processing caused by inconsistencies in trimming schemes.

4.19. Variable Declaration

Declaring variables explicitly helps expose errors, such as misspelled variable names. To make code more reliable and readable use the Option Explicit statement in VBScript.

When you want to use strong variable typing, the logic should be programmed into a component built with a language that supports it, such as Visual Basic 5.0 or Visual J++. Loosely typed variables, such as Variants in VBScript, can affect performance, especially when mathematical computations are involved.

4.20. Variable Names for Consistency

For consistency in naming variables, use initial capital letters in variable names (do not capitalize prefixes).

To make the intended use of a variable clear to others reading your script, use three-character prefixes to indicate data type. Even though explicit typing is not supported in VBScript, the use of prefixes is recommended to denote the intended data type.

Table B.1 Suggested Prefixes for Indicating the Data Type of a Variable

Data Type	Prefix
Boolean	bln
Byte	byt
Collection object	col
Currency	cur
Date-time	dtm
Double	dbl
Error	err
Integer	int
Long	lng

Data Type	**Prefix**
Object	obj
Single	sng
String	str
User-defined type	udt
Variant	vnt
ADO command	cmd *Jul 17, 1998 editor's note: This was previously published as cmn in the Resouce Kit; we now recommend cmd.*
ADO connection	cnn
ADO field	fld
ADO parameter	prm
ADO recordset	rst

To keep variable name lengths reasonable, use standardized abbreviations. For clarity, keep abbreviations consistent throughout an application, or group of related applications.

Instead of:

```
strSocialSecurityNumber
```

Use:

```
StrSSN
```

4.21. Variable Scope for Performance

Use local scope (within subroutines and functions) for variables unless you have a compelling reason to use global scope. Local variables are resolved into table lookups when compiled, enabling fast execution at run time.

Declare global variables using the Dim statement before using them. This saves valuable first-use time by eliminating a search of the entire variable list.

5. HTML CONVENTIONS FOR ASP PAGES(OPTIONAL)

This section is optional, you can customize it depend on customer requirements of your project.

ASP applications serve dynamic information in standard HTML format. This means that you can customize information for a wide range of users.

5.1. Support Text-Only Browsing

Many users browse the Web in text-only mode in order to speed up the display of information to their browsers. To ensure that you present as much content as possible to these users, take the following steps to support text-only display:

- Include text-only navigation objects, such as menus.
- Include the alternative (ALT) parameter with each image tag to provide information about images.
- <IMG SRC="gravity.jpg" ALT="Picture of apple falling">
- When providing client-side imagemaps, add a text-only list of the mapped areas, with links to the information that would have been loaded if the user had clicked on a map link.

5.2. Checking HTML Files

You should check and debug your HTML code formally, using either a dedicated HTML checker or an HTML editor that includes a checker.

- Choose an HTML checker that helps enforce your HTML version standard.
- Debug each new HTML file as it is developed. Then debug your files again after each time they are modified.
- To debug an HTML page that contains ASP scripts:
- Run the page that contains ASP scripts.
- View the source.
- Save the output, which is pure HTML.
- Run the saved output through an HTML checker.

This process is especially important for ASP pages that include forms. In these cases, HTML errors may corrupt the Request object sent from the browser to the server and cause a run-time error.

5.3. *Using the 216-Color Palette*

Color palette mismatches are ever-present concerns when you are creating images for a multi-platform Web. Even images in formats that require compact palettes, such as GIF (256 colors max), often display disparate colors when displayed on different platforms, such as Mac OS, Windows 95, and UNIX.

To ensure that your images display the same colors in different browsers on diverse platforms, use the 216-color palette—also called the safety palette, or the 6x6x6 palette. This allows nearly as many colors as a GIF image can display, and displays consistent colors across systems.

5.4. *Designing for Small Screens*

Small-screen formats are still the standard for many users. Although larger formats are making progress, even 800 by 600 is too large to fit on millions of displays. For example, there are about ten million Macintoshes currently in use, many of which display 640 by 480 pixels.

To accommodate a broad spectrum of users, including those with small screens, design for 640 by 480 pixels.

For usability with small screens, keep the average line of text to approximately 12 words.

5.5. *Displaying Standard Error Messages*

For consistency and to make error messages informative, use standard Response.Status values, such as "401 Unauthorized - Logon failed" and other IIS standard responses in your pages. You can customize error messages in cases where additional information is needed.

5.6. *Using Object Statements with Embed Statements*

To use interactive objects delivered to multiple browsers you must code for browsers that do not support the HTML <OBJECT> tag.

To script the use of interactive objects, ActiveX Controls, or Java applets in HTML pages designed for a wide range of browsers:

- Use the <OBJECT> tag to place the object on the page.

- Add the <EMBED> tag to support browsers that do not support the <OBJECT> tag.
- Add a display object using the <NOEMBED> tag for browsers that cannot "play" the object.

The following example places a Shockwave control onto a page, and provides for the contingencies in the preceding sections:

```
<OBJECT ID="ShockedPMDauntless">

  CLASSID="clsid:59CCB4A0-727D-11CF-AC36-00AA00A47DD2"

  CODEBASE="http://www.fabrikam.com/marketing/movers/..."

  WIDTH=180 HEIGHT=120 >

<PARAM NAME="swURL" VALUE="dauntless.dcr">

<EMBED SRC="dauntless.dcr" ALT="Movie of Fabrikam Dauntless model
in action" WIDTH=180 HEIGHT=120>

</EMBED>

<NOEMBED>

<IMG SRC="dauntless.gif" ALT="Picture of FabrikamDauntless model in
action" WIDTH=180 HEIGHT=120>

</NOEMBED>

</OBJECT>
```

5.7. Script for Browser

When possible, write scripts to be executed by the browser, rather than the server. By using the client platform to perform tasks such as data validation, edit-checking, and selecting simple conditional output, you can reserve server resources for those tasks requiring ASP.

6. SOME TIPS

6.1. Tip 1: Always use Option Explicit and variable naming conventions.

This tip is one of the simplest, but also among the most important. If you have programmed in Visual Basic, you probably know about Option Explicit. Option Explicit forces you to declare all variables before using them. If you don't declare Option Explicit, you can create nasty bugs by simply misspelling a variable name.

In the world of ASP there are two types of Option Explicit statements: client-side and server-side. The first Option Explicit, near the top, is for the server-side code, while the Option Explicit below the Client-Side Functions comment is for client-side code. It is important to use both types of Option Explicit in your ASP code.

There is only one explicit data type in VBScript: a variant. Since a variant can represent many different data types, you should keep track of what data type you put into the variant. Otherwise, it is all too easy to forget what type of data is really in the variable. Microsoft suggests you use a form of Hungarian notation for VBScript. For example, intQuantity represents a quantity that is an integer data type

6.2. Tip 2: Use On Error Resume Next carefully.

On Error Resume Next is the only mechanism available to handle runtime errors. As you may know, the On Error Resume Next statement causes the VBScript interpreter to jump to the next line if there is a runtime error. This allows you to handle the error more gracefully.

However, if you place On Error Resume Next so that it has global scope or is in your Main subroutine, you can accidentally disable all runtime checking and introduce difficult-to-find bugs. This is because it affects all the code following the statement, including all function calls. Suppose you had the following code in your Main subroutine:

```
Option Explicit

 On Error Resume Next

 Dim strExample

strExampl = "Test String"
```

Since the assignment statement has misnamed it, the variable strExample will never be assigned the value Test String and you won't get a runtime error!

To avoid this problem, it is best to encapsulate On Error Resume Next in a function that really needs error checking. This limits the scope of the On Error Resume Next statement to that function and the functions called within it. The IsValidTime function in Figure 2 demonstrates this technique.

6.3. *Tip 3: Use Response.Write for debugging.*

You can use the Response.Write method to send any HTML back to the client. I use it primarily for sending debug information back to the client screen so that I can detect bugs. The DebugOutput subroutine below shows a simple function that you can use for debugging. The Main subroutine calls DebugOutput to send "Unexpected option selected" to the client.

```
Main

 Sub Main

     DebugOutput "Unexpected option selected"

 End Sub

 Sub DebugOutput(strDebugMsg)

     Response.Write(strDebugMsg & "<BR>")

     Response.Flush

End Sub
```

Notice that DebugOutput is calling Response.Flush, which is necessary if buffering is turned on (Response.Buffer = True). Otherwise, if you wrote code that caused an infinite loop, you would not see the output from DebugOutput on the client side. You could also modify DebugOutput to use a global debug flag to indicate whether debugging is turned on or off. You could then turn off the DebugOutput when you want to put your code into production. Since ASP is interpreted, you have to be careful not to degrade system performance with too many calls to DebugOutput when you move code to production. This could happen even though the DebugOutput routine does nothing because of the time spent pushing and popping the stack.

Although this article was written with Internet Information Server (IIS) 3.0 in mind, it is worth noting that IIS 4.0 (currently codenamed K2) provides new script debugging features that will make your life a lot easier. IIS 4.0 should be available on the Web by the time you read this article.

6.4. Tip 4: Explicitly convert form fields to avoid problems.

Form fields are always passed as strings to the server side and will remain strings unless you explicitly convert them. In other words, since VBScript only supports the variant data type, you have to be careful that you explicitly convert your fields if you want something other than a string.

Exp:

When the checkbox is turned off, no conversion is used and, as a result, "2" + "2" = "22". If the checkbox is turned on, then 2 + 2 = 4.

If blnUseConversion is False, then intSum equals "22" because the VBScript engine treats the operands as strings that are concatenated. If blnUseConversion is True, then the form fields are explicitly converted and intSum equals 4. Although I admit this example is contrived, I have seen several difficult-to-find bugs that were the result of not explicitly converting fields. In general, it is better to always explicitly convert your form fields.

6.5. Tip 5: Use session variables to assist in page redirection.

You can use session variables for data that must persist across multiple ASP pages. For instance, you can have a user enter a user ID and password and then use session variables to maintain the information across other ASP pages. You can do similar things with hidden fields or cookies, but session variables stay primarily on the server side. This makes them more efficient because it reduces network traffic and client-side processing.

There are a couple of potential pitfalls related to using session variables. First, session variables can get deleted automatically when a session times out. You can control this timeout with the Session.Timeout property. Second, you can use the Session.Abandon method at any time to manually delete session variables. This is necessary if you want to manually recover the memory allocated on the server for the session variables.
Session variables do use HTTP cookies for their implementation, so they do have some of the physical downsides of explicit cookie use: a tiny amount of net traffic, and a few users complaining about cookies. They also use some memory on the server. On the whole, though, session variables are a lot easier to use than cookies.

Well, that's enough theory—let's look at a useful application of session variables. I often use session variables to pass data to a destination page during page redirection. Suppose you have two pages, a user-input page and a database-update page. The database-update page displays a success message if the database updated successfully. Now assume that validation requires access to the database. If validation is unsuccessful, you can pass all the data back to the previous page by copying all the fields to session variables and redirecting to the user-input page. The following code shows an example:

```
Sub Main

     'Data validation here

     ...

     Failed data validation

     CopyFormFieldsToSession

     Response.Redirect "UserInput.asp"

 End Sub

 Sub CopyFormFieldsToSession

     Dim strFV

     For Each strFV In Request.Form

         Session(strFV) = Request.Form(strFV)

     Next

End Sub
```

As an alternative to session variables, you can also pass back variables by appending a query string to Response.Redirect. For example, the following code passes to UserInput.asp a variable called Name that is set to Joe:

```
Response.Redirect "UserInput.asp?Name=Joe"
```

This works fine when you want to pass a few fields, but the query string must be less than 1,024 characters, so you have to be careful. If you have a lot of data, session variables are a better way to go. You'll see another example of

6.6. Tip6: Client side or server side?

There are two places that you can include user-input validation: the client side or the server side. It is important to clearly understand what goes where. I use these rules to put the validation in the right place:

Rule 1: Put as much validation as you can on the client side.

Rule 2: If the user-input validation requires database access, then that validation should be done on the server side.

The justification for Rule 1 is that response time is much better for the user if the validation is done on the client side. Suppose you have an ASP page that requires the user to input a Social Security number and get back information about the person that is assigned that number. On the client side, you should validate that the user-entered number uses the format of ###-##-####. There is no need to submit the page if the validation fails. On the server side, validate that the Social Security number is in the database. If it is not, display an error through redirection or some other mechanism.

6.7. Tip 7: Write a standard error-handling page for ADO and other types of errors.

There are two types of errors: user errors and system errors. User errors are typical validation errors that the user inputs mistakenly. For example, the user may enter a quantity of 100 when the maximum permitted quantity is 99.

System errors can be caused by software or hardware errors. For example, a SQL select statement might contain invalid syntax. These types of errors must be handled in a reasonable way—they should not cause a runtime abort. The functions are both high and low level, so you can pick and choose the level of abstraction that is right for you. You don't need to do any error checking for most of the function calls because they don't return if there is a serious error.

Approver	Reviewer	Creator
Nguyen Lam Phuong	Pham Minh Tuan	Duong Thanh Nhan