**Version History**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Authors*** | ***Date*** | ***Reviewers*** | ***Review Date*** | ***Release Date*** |
| 1.0 | Application Development Team | 27-Aug-2018 | QMF | 31-Aug-2018 | 03-Sep-2018 |
| 2.0 | Application Development Team | 10-Dec-2019 | QMF | 13-Dec-2019 | 16-Dec-2019 |
| 3,0 | Application Development Team | 02-Nov-2020 | QMF | 06-Nov-2020 | 10-Nov-2020 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Change History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Ver. No.*** | ***Section*** | ***Date*** | ***Change Information*** | ***RFC No.*** |
| 1.0 | All | 03-Sep-2018 | New Release | - |
| 2.0 | All | 16-Dec-2019 | Annual Review | - |
| 3.0 | All | 02-Nov-2020 | Annual Review | - |
|  |  |  |  |  |
|  |  |  |  |  |

# STANDARDS FOR CODING IN ASP

1. **Objective**

The objective of this document is to define coding standards for programs developed in ‘ASP ’.

1. **Scope**

This standard applies to programs developed in ‘ASP ’.

1. **References to (checklists, forms, guidelines, lists, standards, Templates, other processes)**

| ***Item*** | ***Description*** | ***ID*** |
| --- | --- | --- |
| **Checklists** | *--* | *--* |

1. **Document Organization**

This document is organized as follows:

* ASP Application Directory Structure
* Filename Extension
* Object and Variable Initiation
* VBScript Conventions
* HTML Conventions for ASP Pages
* Connection Pooling

1. **ASP Application Directory Structure**

* The directory and file conventions suggested in this section optimize ASP application directory structure for consistency, clarity, and ease of use
* The directories and files described are listed here:
  + /Application\_Root (Default.htm & Global.asa )
  + /Classes
  + /Content
  + /Media
  + /Themes
  + /Data
  + /DLLs
  + /Helpers (not in the Web site directories identifiable to users)
  1. **Application Root Directory**
* For readability and clarity, the application root directory should be named in a way that conveys what the site is about. For example, a site about financial research might be named /FinancialResearch or /FR. Avoid application root names that might be misidentified as standard subdirectories of a site, such as /Media or /Content. Also, avoid names that sound like part numbers or codes, such as /FR98346A
* While the site is being developed, the application root directory should be stored under /InetPub/Wwwroot
* The root directory of every application should contain at least these files:
* Default.htm
* Global.asa
* Default.htm or Default.asp should be the default home page for the site, and the server default should be set accordingly. This combination enables users to find sites in your organization consistently, by typing the server address plus the application root directory name
* Global.asa contains scripts that affect the Application object and Session object. For example, Global.asa scripts make application- and session-scope variables available at startup. Global.asa should be present at the application root level for every ASP application
  1. **Classes Directory**
* The /Classes directory holds Java classes used by the application
  1. **Content Directory**
* The /Content directory holds all .htm files and all .asp files that may be run by a user of the site. The /Content directory has execute permissions. Keeping all scripted pages in this directory branch simplifies permissions management
  1. **Media Sub-directory**
* A sub-directory of the /Content directory, Media should contain sub-directories for sounds, images, animation files, .avi files, and similar items. An /Images sub-directory should contain bullets, buttons, icons, and others that are used throughout the application independently of changing themes
  1. **Themes Sub-directory**
* The /Themes directory is a sub-directory of the /Content directory. Establish a convention of using the /Themes directory to enable application-wide changes to the look of a site. The directory should contain style sheets, bullets, buttons, icons, rules, and similar items organized so that you can change the look of an application by changing any or all the theme-related items easily. Each item in the /Theme directory can be linked dynamically by setting an application variable to the image's virtual path. Themes are stored in Options.txt and assigned to the **Application** object in Global.asa
  1. **Data Directory**
* This directory should contain all database access information such as SQL scripts, file-based dataset names or other data needed by the application
  1. **DLLs Directory**
* This directory should contain COM server components and Visual Basic 5.0 run-time DLLs, such as Vbrun500.dll and Msvbvm50.dll
  1. **Helper Files**
* Helper files are including files or text files that make information available across the application. For security reasons the directory containing helper files should not be stored in the published Web space (the Web site directories identifiable to users)

1. **Filename Extension**

* 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
* Standards for Extensions for Page Files:
* .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
* Standards for Extensions for Included Files:
* .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
* .js--for common Java Source files
* 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)

1. **Object and Variable Initiation**

* The following discussion focuses on initializing and setting dimensions for objects and variables to achieve speed of execution and efficient use of server resources

* 1. **Application-Wide Scope for Convenience**
* Information stored in variables with application-wide scope is in memory and is cached for access by any page in the application. Establish a convention to give application scope to variables used often within an application if the values do not change frequently
* Whatever the potential benefits, use caution in deciding whether or not to give application scope to an object. It can potentially affect performance and decrease reliability (your application could hang). For best results, use application scope for objects with threading marked BOTH, aggregating the **FreeThreadedMarshaler**
  1. **Declaring Objects with the <OBJECT> Tag**
* For objects that may or may not be used in an application, it's often most efficient to declare the objects without instantiating them until the need to use them arises (until they are referenced). To declare an object without actually creating it, use the <OBJECT> tag on the page instead of **Server.CreateObject()**
* Another advantage to bear in mind is that the <OBJECT> tag uses CLASSIDs, which are unique and tend to eliminate name collisions. The PROGIDs used by **Server.Create.Object()** do not force unique names, creating the possibility of name collisions
* Also, the <OBJECT> tag is supported in Global.asa and can be used to define scope using SCOPE or SESSION as values

***Note:*****Visual Basic Applications as DLLs**

* When you convert Visual Basic applications for use in ASP, they should be run as DLLs (components), rather than being converted to VBScript. Visual Basic DLLs will generally run more efficiently than scripts written in any scripting language. Encapsulate the Visual Basic code in DLLs and use **Server.CreateObject()**
* Visual Basic 5.0 should be used to create an ActiveX DLL that has its project properties set to Run in **Unattended Mode, Apartment Model Threaded**, and **Multi-instance**

1. **VBScript Conventions**

* 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. **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
  1. **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

* 1. **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
  1. **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
  1. **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 %>

* 1. **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 %>

* 1. **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 %>

* 1. **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
  1. **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
  1. **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*** |
| AddNew | 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 |
| ListFor | returns criteria-based multiple rows | Customer.ListForPeriod |

* 1. **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")

* 1. **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
  1. **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

* 1. **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
  1. **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
  1. **String Concatenation**
* For the sake of consistency and to achieve more self-documenting scripts, use the string concatenator (&), instead of a plus (+) sign
  1. ***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 . . .

* 1. **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
  + 1. ***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
  1. **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
  + 1. ***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 |
| Object | obj |
| Single | sng |
| String | str |
| User-defined type | udt |
| Variant | vnt |
| ADO command | 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

* + 1. ***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
  1. **Control Names for Consistency**
* For consistency in naming controls, use initial capital letters in control names (do not capitalize prefixes)
* To make the intended use of a control clear to others reading your script, use three-character prefixes to indicate control type. Even though explicit typing is not supported in VBScript, the use of prefixes is recommended to denote the intended control type
* Table B.2 Suggested Prefixes for Indicating the Data Type of a Control

|  |  |
| --- | --- |
| ***Input / Control Type*** | ***Prefix*** |
| Text | txt |
| Password | pwd |
| Radio | rad |
| Check Box | chk |
| Combo Box (Select) | cbo |
| ListBox | lst |
| Command Button | cmd |
| Submit Button | sub |
| Reset Button | rst |
| Form | frm |

1. **HTML Conventions for ASP Pages**

* ASP applications serve dynamic information in standard HTML format. This means that you can customize information for a wide range of users
  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
  1. **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
  1. **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 pallete, or the 6x6x6 palette. This allows nearly as many colors as a GIF image can display, and displays consistent colors across systems
  1. **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
  1. **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
  1. **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 Fabrikam

Dauntless model in

action" WIDTH=180 HEIGHT=120>

</NOEMBED>

</OBJECT>

1. **Connection Pooling**

* One of the potential bottlenecks in ASP application performance is connection management. If not managed properly, the opening and closing of connections can happen too frequently and can cause reduced server performance. Windows NT Server 4.0 features built-in support for Connection Pooling, maintaining connections optimally to achieve faster application performance and graceful timeout management with less coding effort
* To use ODBC connection pooling:
* Configure the driver for the database to which you are establishing a connection
* Check the Windows NT registry to verify that Connection Pooling is on, and let it handle the connection logic
* If Connection Pooling is off, use Regedit32 to turn it on in the Windows NT registry.
* Open individual connections--in your ADO code--just before you need data access on each individual page
* Close your connections when the data access activities are complete
* Enabling connection pooling involves setting the **CPTimeout** property
* Enabling connection pooling involves setting the **CPTimeout** property
* For descriptions and examples on how to configure the **CPTimeout** property in the registry, see *Internet Information Server Resource Kit,* Chapter 6, "Data Access and Transactions."