COMMONWEALTH OF PENNSYLVANIA
Department of Human Services
INFORMATION TECHNOLOGY STANDARD

Name Of Standard: **Java Script Coding**	Number: **STD-EASS011**
Domain: **Application**	Category:
Date Issued: **06/29/2001**	Issued By Direction Of:
Date Reviewed: **2/17/2016**	Cliff Van Scyoc, Dir of Div of Tech Engineering

Table of Contents

JavaScript Coding Standards

Introduction

JavaScript is Netscape's cross-platform, object-oriented scripting language, which is compatible with Netscape Navigator and Microsoft Internet Explorer Web browsers. Core JavaScript contains a core set of objects, such as **`Array`**, **`Date`**, and **`Math`**, and a core set of language elements such as operators, control structures, and statements. Core JavaScript can be extended for a variety of purposes by supplementing it with additional objects. For example:

- ***Client-side JavaScript*** extends the core language by supplying objects to control a browser (Navigator or another Web browser) and its Document Object Model (DOM). For example, client-side extensions allow an application to place elements on a hypertext markup language (HTML) form and respond to user events such as mouse clicks, form input, and page navigation.
- ***Server-side JavaScript*** extends the core language by supplying objects relevant to running JavaScript on a server. For example, server-side extensions allow an application to communicate with a relational database, provide continuity of information from one invocation of the application to another, or perform file manipulations on a server.

JavaScript lets you create applications that run over the Internet. Client applications run in a browser, such as Netscape Navigator, IE and server applications run on a server, such as Netscape Enterprise Server, IIS. Using JavaScript, you can create dynamic HTML pages that process user input and maintain persistent data using special objects, files, and relational databases.

Purpose

The purpose of this document is to provide JavaScript coding standards.

Standard Change Log

Change Date	Version	CR #	Change Description	Author and Organization
06/29/01	1.0	N/A	Initial creation.	Deloitte Consulting
12/10/02	1.1	00GX	Edited for style.	Beverly Shultz Diverse Technologies Corporation / Deloitte Consulting
10/16/2013	1.1	N/A	Reviewed, no changes.	Brad Deetz
2/17/2016	1.2	NA	Updated to reflect agency name change	Brad Deetz

JavaScript Standards

1. Explicitly use LANGUAGE attribute in the <SCRIPT> tag.

   For example <SCRIPT LANGUAGE =”javascript”>

2. Embed script within HTML comment tags so that non Script-capable browsers ignore it. The following client script shows a script block in comments:

```
<SCRIPT LANGUAGE="javascript">
<!--
Code goes here
//-->
</SCRIPT>
```

3. Always use *var* keyword to declare all variables to prevent variable scope problems. Without *var* JavaScript will create a global variable.
4. Define common functions to handle common tasks in all pages.
5. Place reusable code in external JavaScript source files (like *.js or *.inc)

Validating Forms

With its capability to interact with form elements, JavaScript is well suited to validate information directly on the HTML page. Checking information on the client side also makes it much harder for users to send incompatible or damaging data to the server.

Three types of Edits, which must be done at Client side, are

- Check for Numeric entry
- Check for Alpha Numeric entry
- Check for Mandatory field entry

Use Page Level edits for validating the data if there is any error, display appropriate message and the position the cursor to that appropriate field.

Interface Controls

There is no way to universally implement many of the interface controls found in GUI operating systems on a Web browser. Menus, dialog boxes, associating sounds or animations with events, and other common graphical user interface (GUI) elements can be added to a Web application via JavaScript. Not all browsers however, support part or all of these elements ultimately resulting in the need to build two sites: one that supports the GUI controls and a second that uses standard hypertext markup language (HTML) elements.

Do not use GUI controls.

Status Bar Messages: With event handlers and the window status property, JavaScript enables the browser to display custom messages in the status bar that respond to user actions. One of the most popular implementation is a descriptive line for hyperlinks. One problem with the status property is that it becomes the default message until a browser-generated message overrides it, or status is set to a different value.

Do not use status bar messages.

Standards for Comments

Comments can be either single line or multi-line or trailing. Comments should be used to give overviews of code and provide additional information that is not readily available in the code itself. It should contain only information that is relevant to reading and understanding the program.

- Comments should not be enclosed in large boxes drawn with asterisks or other characters.
- Comments should never include special characters and unnecessary information.

Single Line Comments

The // comment delimiter will be used to comment out a complete line or only a partial line.

Example

```
if (condition){
 // Here goes the single line comment
Statements
}
```

Multi line Comments

Use Multi Line comments when commenting more than one line. This type of comments cannot be nested.

Example

```
  /* Here goes the multi line comment*/
```

Trailing Comments

Very short comments can appear on the same line as the code they describe, but should be shifted far enough to separate them from the statements. If more than one short comment appears in a chunk of code, they should all be indented to the same tab setting.

Here's an example of a trailing comment in Java Script code:

```
if (a == 2) {
    return TRUE.         /* special case */
}
```

Case Sensitivity and Naming Conventions

JavaScript is case sensitive: You have to use the correct combination of uppercase and lowercase for everything in the JavaScript namespace. This includes all keywords in any object models you use.

The following naming conventions will be enforced

- Must begin with alphanumeric character
- Letters must be lowercased or combination of upper- or lowercase e.g. FirstName
- Cannot contain an embedded period.
- Cannot exceed 40 characters.

Indentation Standards

Four spaces should be used as the unit of indentation. Always use tabs. Do not use spaces for indentation.

Line Length

Avoid lines longer than 70 characters, since they're not handled well by many terminals and tools.

Wrapping Lines

When an expression will not fit on a single line, break it according to these general principles:

- Break after a comma.
- Break before an operator.
- Prefer higher-level breaks to lower-level breaks.
- Align the new line with the beginning of the expression at the same level on the previous line.

If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Declaration Standards

Number per line

One declaration per line is recommended since it encourages commenting. In other words,

```
int level. // indentation level
int size. // size of table
```

is preferred over

```
int level, size.
```

Do not put different types on the same line.

Initialization

Local variables should be initialized where they are declared. The only reason not to initialize a variable where it is declared is if the initial value depends on some computation occurring first.

Placement

Declarations must be put only at the beginning of blocks. (A block is any code surrounded by curly braces "{" and "}".) Waiting to declare variables until their first use can be confusing and hamper code portability within the scope.

```
void myMethod() {
    int int1 = 0.      // beginning of method block
    if (condition) {
     int int2 = 0.  // beginning of "if" block
       ...
    }
}
```

The one exception to the rule is indexes of for loops, which in Java can be declared in the for statement:

```
for (int i = 0. i < maxLoops. i++) { ... }
```

Avoid local declarations that hide declarations at higher levels. For example, do not declare the same variable name in an inner block:

```
int count.
...
myMethod() {
    if (condition) {
        int count = 0.  // AVOID!
        ...
    }
    ...
     }
```

Function Declarations

A function definition should consist of a **`function`** statement and a block of JavaScript statements.

```
function FnName(param1,param2,…paramN){

  Statement or block of Statements
  return values if any
}
```

Standards for Statements

Simple Statements

Each line should contain at most one statement.

Compound Statements

Compound statements are statements that contain lists of statements enclosed in braces, as in this example: **`{statements}.`** The enclosed statements should be indented one more level than the compound statement. The opening brace should be at the end of the line that begins the compound statement. The closing brace should begin a line and be indented to the beginning of the compound statement.

Braces are used around all statements, even single statements, when they are part of a control structure, such as an **`if-else`** or **`for`** statement. This makes it easier to add statements without accidentally introducing bugs due to forgetting to add braces.

Return Statements

A `return` statement with a value should not use parentheses unless they make the return value more obvious in some way.

Example:

```
return expression.
return (size ? size : defaultSize).
```

if, if-else, if else-if else Statements

The **`if-else`** class of statements should have the following form:

```
if (condition) {
    statements.
}

if (condition) {
    statements.
} else {
    statements.
}

if (condition) {
    statements.
} else if (condition) {
    statements.
} else{
    statements.
}
```

Note: **`if`** statements always use braces {}. Avoid the following error-prone form:

```
if (condition) //AVOID! THIS OMITS THE BRACES {}!
    statement.
```

for Statements

A **`for`** statement should have the following form:

```
for (initialization. condition. update) {
    statements.
}
```

An empty **`for`** statement (one in which all the work is done in the initialization, condition, and update clauses) should have the following form:

```
for (initialization. condition. update).
```

When using the comma operator in the initialization or update clause of a **for** statement, usage of more than three variables needs to be avoided. If needed, separate statements should be used before the **for** loop (for the initialization clause) or at the end of the loop (for the update clause).

while Statements

A **`while`** statement should have the following form:

```
while (condition) {
     statements.
}
```

An empty **`while`** statement should have the following form:

```
while (condition).
```

do-while Statements

A **`do-while`** statement should have the following form:

```
do {
    statements.
} while (condition).
```

Label Statement

The syntax of the label statement looks like the following:

```
label :
   statement
```

The value of `label` may be any JavaScript identifier that is not a reserved word. The **`statement`** that you identify with a label may be any type.

switch Statements

A **switch** statement should have the following form:

```
switch (expression){
   case label :
      statement.
      break.
   case label :
      statement.
      break.
   ...
   default : statement.
}
```

The optional break statement associated with each case label ensures that the program breaks out of switch once the matched statement is executed and continues execution at the statement following switch. If break is omitted, the program continues execution at the next statement in the **switch** statement.

Every **switch** statement should include a default case. The **break** in the default case is redundant, but it prevents a fall-through error if later another **case** is added.

Standards for White Spaces

Blank Lines

Blank lines improve readability by setting off sections of code that are logically related.

One blank line should always be used in the following circumstances:

- Between functions
- Between the local variables in a function and its first statement
- Before a block or single-line comment
- Between logical sections inside a function to improve readability

Blank Spaces

Blank spaces should be used in the following circumstances:

- A blank space should appear after commas in argument lists.
- All binary operators except “.” should be separated from their operands by spaces. Blank spaces should never separate unary operators such as unary minus, increment ("++"), and decrement ("--") from their operands.

Example:

```
a += c + d.
 a = (a + b) / (c * d).

 while (d++ = s++) {
     n++.
  }
```

The expressions in a **`for`** statement should be separated by blank spaces.

Example:

```
for (expr1. expr2. expr3)
```

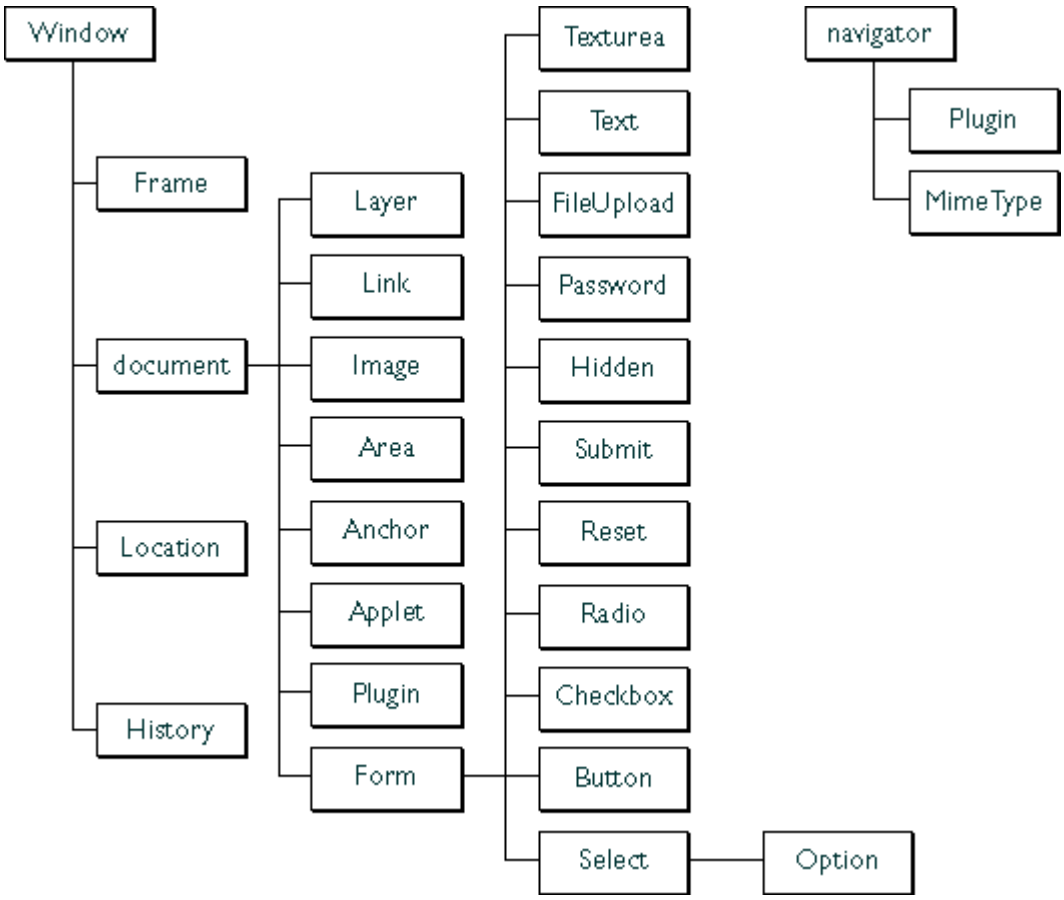
Reserved Words

This appendix lists the reserved words in JavaScript.

The reserved words in this list cannot be used as JavaScript variables, functions, methods, or object names. Some of these words are keywords used in JavaScript. Others are reserved for future use.

Reserved Words			
abstract	else	instanceof	switch
boolean	enum	int	synchronized
break	export	interface	this
byte	extends	long	throw
case	false	native	throws
catch	final	new	transient
char	finally	null	true
class	float	package	try
const	for	private	typeof
continue	function	protected	var
debugger	goto	public	void
default	if	return	volatile
delete	implements	short	while
do	import	static	with
double	in	super	

Client-Side JavaScript Object Hierarchy



As depicted in this diagram, all client-side objects are derived from either the Window or the Navigator object. By using the Window object the programmer is allowed to access the various frames, documents, layers, and forms on a page, as well as many other objects and properties. The programmer should maintain this hierarchy.

The Navigator object pertains to elements that are part of the browser itself. This specifically refers to the plug-ins installed and the Multipurpose Internet Mail Extensions (MIME) types with which the browser is associated. Using the Navigator object allows checking for the browser version, determining the plug-ins installed, and what programs are associated with the various MIME types registered on the system.