JavaScript coding standards

Indenting

Use an indent of 2 spaces, with no tabs. No trailing whitespace.

String Concatenation

Always use a space between the `+` and the concatenated parts to improve readability.

```
var string = 'Foo' + bar;
string = bar + 'foo';
string = bar() + 'foo';
string = 'foo' + 'bar';
```

When using the concatenating assignment operator ('+='), use a space on each side as with the assignment operator:

```
var string += 'Foo';
string += bar;
string += baz();
```

CamelCasing

Unlike the variables and functions defined in Drupal's PHP, multi-word variables and functions in JavaScript should be lowerCamelCased. The first letter of each variable or function should be lowercase, while the first letter of subsequent words should be capitalized. There should be no underscores between the words.

Semi-colons

JavaScript allows any expression to be used as a statement and uses semi-colons to mark the end of a statement. However, it attempts to make this optional with "semi-colon insertion", which can mask some errors and will also cause JS aggregation to fail. All statements should be followed by `;` except for the following:

```
for, function, if, switch, try, while
```

The exceptions to this are functions declared like

```
  Drupal.behaviors.tableSelect = function (context) {
    // Statements...
  };
```

and

```
  do {
    // Statements...
  } while (condition);
```

These should all be followed by a semi-colon.

In addition the `return` value expression must start on the same line as the return keyword in order to avoid semi-colon insertion.

If the "Optimize JavaScript files" performance option in Drupal 6 is enabled, and there are missing semi-colons, then the JS aggregation will fail. It is therefore very important that semi-colons are used.

Control Structures

These include `if, for, while, switch`, etc. Here is an example `if` statement, since it is the most complicated of them:

```
if (condition1 || condition2) {
  action1();
}
else if (condition3 && condition4) {
  action2();
}
else {
  defaultAction();
}
```

Control statements should have one space between the control keyword and opening parenthesis, to distinguish them from function calls.

You are strongly encouraged to always use curly braces even in situations where they are technically optional. Having them increases readability and decreases the likelihood of logic errors being introduced when new lines are added.

switch

For `switch` statements:

```
switch (condition) {
  case 1:
    action1();
    break;

  case 2:
    action2();
    break;

  default:
```

```
    defaultAction();
}
```

try

The `try` class of statements should have the following form:

```
  try {
    // Statements...
  }
  catch (error) {
    // Error handling...
  }
  finally {
    // Statements...
  }
```

for in

The `for in` statement allows for looping through the names of all of the properties of an object. Unfortunately, all of the members which were inherited through the prototype chain will also be included in the loop. This has the disadvantage of serving up method functions when the interest is in data members. To prevent this, the body of every `for in` statement should be wrapped in an `if` statement that does filtering. It can select for a particular type or range of values, or it can exclude functions, or it can exclude properties from the prototype. For example:

```
for (var variable in object) if (filter) {
  // Statements...
}
```

You can use the `hasOwnProperty` method to distinguish the true members of the object:

```
for (var variable in object) if (object.hasOwnProperty(variable)) {
  // Statements...
}
```

Functions

Functions and Methods

Functions and methods should be named using lowerCamelCase.

```
Drupal.behaviors.tableDrag = function (context) {
  for (var base in Drupal.settings.tableDrag) {
    if (!$('#' + base + '.tabledrag-processed', context).size()) {
      $('#' + base).filter(':not(.tabledrag-processed)').each(addBehavior);
      $('#' + base).addClass('tabledrag-processed');
    }
  }
};
```

Function Calls

Functions should be called with no spaces between the function name, the opening parenthesis, and the first parameter; spaces between commas and each parameter, and no space between the last parameter, the closing parenthesis, and the semicolon. Here's an example:

```
foobar = foo(bar, baz, quux);
```

As displayed above, there should be one space on either side of an equals sign used to assign the return value of a function to a variable. In the case of a block of related assignments, more space may be inserted to promote readability:

```
short        = foo(bar);
longVariable = foo(baz);
```

If a function literal is anonymous, there should be one space between the word function and the left parenthesis. If the space is omitted, then it can appear that the function's name is actually "function".

```
div.onclick = function (e) {
  return false;
};
```

Function Declarations

```
function funStuff(field) {
  alert("This JS file does fun message popups.");
  return field;
}
```

Arguments with default values go at the end of the argument list. Always attempt to return a meaningful value from a function if one is appropriate.

Please note the special notion of anonymous functions explained above.

Variables and Arrays

All variables should be declared with `var` before they are used and should only be declared once. Doing this makes the program easier to read and makes it easier to detect undeclared variables that may become implied globals.

Variables should not be defined in the global scope; try to define them in a local function scope at all costs. All variables should be declared at the beginning of a function.

Constants and Global Variables

lowerCamelCasing should be used for pre-defined constants. Unlike the PHP standards, you should use lowercase `true`, `false` and `null` as the uppercase versions are not valid in JS.

Variables added through `drupal_add_js()` should also be lowerCamelCased, so that they can be consistent with other variables once they are used in JavaScript.

```
<?php
drupal_add_js(array('myModule' => array('basePath' => base_path())),
'setting');
?>
```

This variable would then be referenced:

```
Drupal.settings.myModule.basePath;
```

Arrays

Arrays should be formatted with a space separating each element and assignment operator, if applicable:

```
someArray = ['hello', 'world'];
```

Note that if the line spans longer than 80 characters (often the case with form and menu declarations), each element should be broken into its own line, and indented one level:

Note there is no comma at the end of the last array element. This is different from the PHP coding standards.. Having a comma on the last array element in JS will cause an exception to occur.

Comments

Non-documentation comments should use capitalized sentences with punctuation. All caps are used in comments only when referencing constants, e.g., TRUE. Comments should be on a separate line immediately before the code line or block they reference. For example:

```
// Unselect all other checkboxes.
```

If each line of a list needs a separate comment, the comments may be given on the same line and may be formatted to a uniform indent for readability.

C style comments (`/* */`) and standard C++ comments (`//`) are both fine.

Header Comment Blocks

All source code files in the core Drupal distribution should contain the following comment block as the header:

```
// $Id$
```

This tag will be expanded by the CVS to contain useful information

```
// $Id: CODING_STANDARDS.html,v 1.14 2008/02/19 03:36:41 ax Exp $
```

JS code placement

JavaScript code should not be embedded in the HTML where possible, as it adds significantly to page weight with no opportunity for mitigation by caching and compression.

"with" statement

The `with` statement was intended to provide a shorthand for accessing members in deeply nested objects. For example, it is possible to use the following shorthand (but not recommended) to access `foo.bar.foobar.abc`, etc:

```
with (foo.bar.foobar) {
  var abc = true;
  var xyz = true;
}
```

However it's impossible to know by looking at the above code which `abc` and `xyz` will get modified. Does `foo.bar.foobar` get modified? Or is it the global variables `abc` and `xyz`?

Instead you should use the explicit longer version:

```
foo.bar.foobar.abc = true;
foo.bar.foobar.xyz = true;
```

or if you really want to use a shorthand, use the following alternative method:

```
  var o = foo.bar.foobar;
  o.abc = true;
  o.xyz = true;
```

Operators

True or false comparisons

The `==` and `!=` operators do type coercion before comparing. This is bad because it causes:
`' \t\r\n' == 0`
to be true. This can mask type errors. When comparing to any of the following values, use the `===` or `!==` operators, which do not do type coercion:
`0 '' undefined null false true`

Comma Operator

The comma operator causes the expressions on either side of it to be executed in left-to-right order, and returns the value of the expression on the right, and should be avoided. Example usage is:

```
var x = (y = 3, z = 9);
```

This sets x to 9. This can be confusing for users not familiar with the syntax and makes the code more difficult to read and understand. So avoid the use of the comma operator except for in the control part of `for` statements. This does not apply to the comma separator (used in object literals, array literals, etc.)

Avoiding unreachable code

To prevent unreachable code, a `return`, `break`, `continue`, or `throw` statement should be followed by a `}` or `case` or `default`.

Constructors

Constructors are functions that are designed to be used with the `new` prefix. The `new` prefix creates a new object based on the function's prototype, and binds that object to the function's implied this parameter. JavaScript doesn't issue compile-time warning or run-time warnings if a required `new` is omitted. If you neglect to use the `new` prefix, no new object will be made and this will be bound to the global object (bad). Constructor functions should be given names with an initial uppercase and a function with an initial uppercase name should not be called unless it has the `new` prefix.

Use literal expressions

Use literal expressions instead of the `new` operator:

- Instead of `new Array()` use `[]`
- Instead of `new Object()` use `{}`
- Don't use the wrapper forms `new Number`, `new String`, `new Boolean`.

In most cases, the wrapper forms should be the same as the literal expressions. However, this isn't always the case, take the following as an example:

```
var literalNum = 0;
var objectNum = new Number(0);
if (literalNum) { } // false because 0 is a false value, will not be
executed.
if (objectNum) { }  // true because objectNum exists as an object, will be
executed.
if (objectNum.valueOf()) { } // false because the value of objectNum is 0.
```

Typeof

When using a `typeof` check, don't use the parenthesis for the typeof. The following is the correct coding standard:

```
if (typeof myVariable == 'string') {
  // ...
}
```

Modifying the DOM

When adding new HTML elements to the DOM, don't use `document.createElement()`. For cross-browser compatibility reasons and also in an effort to reduce file size, you should use the jQuery equivalent.

Don't:

```
this.popup = document.createElement('div');
this.popup.id = 'autocomplete';
```

Do:

```
this.popup = $('<div id="autocomplete"></div>')[0];
```