JQuery Core Style Guidelines

CONTENTS

JSLINT

jQuery core passes JSHint (not JSLint) with a few stipulations. To run JSHint against jQuery you can download jQuery from Git and run 'grunt'.

We ignore the following warnings from JSHint:

"Expected an identifier and instead saw 'undefined' (a reserved word)." - In jQuery we re-declare 'undefined' as we use it internally. Doing so prevents users from accidentally munging the name (assigning to undefined and messing up our tests) and also makes the undefined checks slightly faster.

"Use '===' to compare with 'null'." and "Use '!==' to compare with 'null'." - Almost universally in jQuery core we use === or !== with the exception of when we're comparing against null. It's actually quite useful to do == null or != null as it will pass (or fail) if the value is either null or undefined. If we want to explicitly check for a null or undefined value we use === null or === undefined (discussed in depth below).

"Expected an assignment or function call and instead saw an expression." - There are two areas in which we access a property with seemingly no side effect, those are:

`parent.selectedIndex;` (or similar) this is due to the fact that WebKit mis-reports the default selected property of an option, accessing the parent's selectedIndex property fixes it.

`Array.prototype.slice.call( document.documentElement.childNodes, 0 )[0].node...` - This check simply attempts to determine if, after a slice on a NodeList, the resulting collection still holds DOM nodes. This is a feature detection and is wrapped in a try/catch (if the property doesn't exist it'll throw an exception.

"Expected a 'break' statement before 'case'." - There is only one legitimate reason to use switch statements over an object of properties or a series of if/else statements: passthroughs. We use it inside of Sizzle to optimize our `:only-child` selector checks, reducing code and improving performance.

Additionally we enable the following two JSLint options by default:

"evil" - This allows code evaluation. We only use it in one place inside of jQuery: Backwards compatibility support for JSON parsing (we preferably use the modern JSON.parse method if it's available).

"forin" - This removes the requirement of doing a hasOwnProperty check inside of for/in loops. jQuery doesn't support working in an environment that has manipulated the `Object.prototype` as it's considered harmful.

SPACING

Use tabs to indent your code.

Be sure to use liberal spacing in your code.

```
// No:
if(blah==="foo"){
  foo("bar","baz",{zoo:1});
}

// Yes:
if ( blah === "foo" ) {
  foo( "bar", "baz", {
    zoo: 1
  });
}
```

Opening braces are always prefixed by a space.

Don't use tabs inside a line.

```
var a = true,
  // yayy
  c = false,
  // grrr
  b   = false;
```

Lines with nothing on them should have no whitespace.

There should be no whitespace at the end of a line.

COMMENTS

Long comments should use `/* ... */`.

Single line comments should always be on their own line and be above the line they reference. Additionally there should be an extra endline above it. For example:

```
var some = "stuff";

// We're going to loop here
for ( var i = 0; i < 10; i++ ) {}
```

Inline comments are allowed as an exception when used to annotate special arguments in formal parameter lists:

```
function foo( types, selector, data, fn, /*INTERNAL*/ one ) {
   // do stuff.
 }
```

EQUALITY

Strict equality checks (===) should be used in favor of == wherever possible.

BLOCKS

if/else/for/while/try always have braces and always go on multiple lines.

Braces should always be used on blocks.

```
// NO:
if ( true )
   blah();

// YES:
```

```
if ( true ) {
   blah();
}
```

Don't put statements on the same line as a conditional.

```
// NO:
if ( true ) return;
if ( true ) blah();

// YES:
if ( true ) {
   return;
}

if ( true ) {
   blah();
}
```

else/else if/catch go on the same line as the brace.

```
if ( blah ) {
   baz();
} else {
   baz2();
}
```

Exception: else if is allowed, a else { } is not required

```
} else if ( test ) {
  blah();
}

// Not needed:
} else {
  if ( test ) {
    blah();
  }
}
```

Don't use ternary operators instead of if/else statements.

Don't use `object && object.method()` instead of if/else statements (except in conditionals).

FUNCTION CALLS

Always include extra spaces around the arguments.

```
foo( true );
```

Exception: `foo(true)` (no spaces) is allowed if you're already inside of another function call.

```
foo( bar(true) );
```

Exception: Functions, object literals, array literals and string literals go snug to front and back of the parentheses - but ONLY when it's the only argument.

```
foo(function() { });
foo({   });
foo([   ]);
foo("single string argument");
```

Exception: Multi-line function/object/array literals go snug at end (and at start ONLY if they are the only argument).

```
WRONG:
foo( true, { blah: "baz" });

RIGHT:
foo( true, {
  blah: "baz"
});
```

Empty functions, object literals and array literals don't need filler spaces.

```
foo();
{}
[]
```

Always have spaces after commas and colons.

ARRAYS AND OBJECTS

Empty objects and arrays don't need filler spaces.

```
[]
{}
```

Always have spaces after commas and colons.

ASSIGNMENT

Assignments should always have a semicolon after them.

Semicolons should always be followed by an endline.

Assignments in a declaration should always be on their own line. Declarations that don't have an assignment should be listed together at the start of the declaration. For example:

```
var a, b, c,
  test = true,
  test2 = false;
```

TYPE CHECKS

String: `typeof object === "string"`

Number: `typeof object === "number"`

Boolean: `typeof object === "boolean"`

Object: `typeof object === "object"`

Plain Object: `jQuery.isPlainObject(object)`

Function: `jQuery.isFunction(object)`

Array: `jQuery.isArray(object)`

Element: `object.nodeType`

null: `object === null`

null or undefined: `object == null`

undefined:

Global Variables: `typeof variable === "undefined"`

Local Variables: `variable === undefined`

Properties: `object.prop === undefined`

REGEXP

All RegExp operations should be done using `.test()` and `.exec()`. `"string".match()` is no longer used.

STRINGS

Strings should always use double-quotes instead of single-quotes.

NODES

`.nodeName` should always be used in favor of `.tagName`.

`.nodeType` should be used to determine the classification of a node (not `.nodeName`).

Don't attach expandos to nodes - only attach data using .data().