

JSLint

Improving the quality of JavaScript code

<http://jshint.com>



The [JavaScript](#) Code Quality Tool

Edition 2010-07-14

[Read the instructions.](#) [Set the options.](#) [Enjoy *The Good Parts*.](#)

JSHint

clear

Paste your program into the text box above and click a [JSHint](#) button.

[Warning: JSHint will hurt your feelings.](#)

Overview

- * Identifies real and potential issues in JavaScript code, JSON, HTML and CSS
- * Enforces coding conventions
- * Can enforce many recommendations in the book "JavaScript: The Good Parts"
- * Written in JavaScript by Douglas Crockford

Ways To Use

* In a web browser

- * browse <http://jshint.com/>
- * specify options in checkboxes and text fields at bottom of page (see slide 8)
- * paste code in textarea at top of page (see slide 2)
- * press "JSHint" button

* From command-line

- * using Rhino, a JavaScript interpreter written in Java
 - * download `jshint.js` from <http://www.jshint.com/rhino/>
 - * `java -jar $RHINO_DIR/js.jar jshint.js file-path-to-be-checked`
 - * write a script to run the above command
- * using Windows Script Host under Microsoft Windows
 - * download `jshint.js` from <http://www.jshint.com/wsh/>
 - * `cscript jshint.js < file-path-to-be-checked`

Example Input

```
function foo() {  
    a = 1; // undeclared variable  
    bar(); // call to function that hasn't been defined yet  
    var b = 2;  
    var c = 3; // more than one var statement in function  
    c = b // no terminating semicolon  
    if (b == c) // == instead of ===  
        print('equal'); // no braces around statement after if  
  
    print('Hello'); // wrong indentation  
}  
  
function bar() {  
    return true;  
}
```

Example Output

Lint at line 5 character 5: 'a' is not defined.

```
a = 1; // undeclared variable
```

Lint at line 6 character 5: 'bar' is not defined.

```
bar(); // call to function that hasn't been defined yet
```

Lint at line 8 character 9: Too many var statements.

```
var c = 3; // more than one var statement in function
```

Lint at line 9 character 10: Missing semicolon.

```
c = b // no terminating semicolon
```

Lint at line 10 character 11: Expected '===' and instead saw '=='.

```
if (b == c) // == instead of ===
```

Lint at line 11 character 9: Expected '{' and instead saw 'print'.

```
print('equal'); // no braces around statement after if
```

Lint at line 11 character 9: Expected 'print' to have an indentation at 5 instead at 9.

```
print('equal'); // no braces around statement after if
```

Lint at line 13 character 3: Expected 'print' to have an indentation at 5 instead at 3.

```
print('Hello'); // wrong indentation
```

Lint at line 16 character 13: 'bar' was used before it was defined.

```
function bar() {
```

Cleaned Up Code

```
function bar() {  
    return true;  
}  
  
function foo() {  
    var a, b, c;  
    a = 1;  
    bar();  
    b = 2;  
    c = 3;  
    c = b;  
    if (b === c) {  
        print('equal');  
    }  
  
    print('Hello');  
}
```

JSLint output for this code is
jslint: No problems found in fixed.js

JSLint Options

Options

- Stop on first error
- Strict white space
- Assume a browser
- Assume `console`, `alert`, ...
- Assume a [Yahoo Widget](#)
- Assume Windows
- Assume [Rhino](#)
- Safe Subset
- [ADsafe](#)
- Tolerate `debugger` statements
- Tolerate `eval`
- Tolerate sloppy line breaking
- Tolerate [unfiltered](#) for `in`
- Tolerate inefficient subscribing
- Tolerate CSS workarounds
- Tolerate `HTML` case
- Tolerate `HTML` event handlers
- Tolerate `HTML` fragments
- Tolerate ES5 syntax
- Allow one `var` statement per function
- Disallow undefined variables
- Disallow dangling `_` in identifiers
- Disallow `==` and `!=`
- Disallow `++` and `--`
- Disallow bitwise operators
- Disallow insecure `.` and `[^...]` in `/RegExp/`
- Require Initial Caps for constructors
- Require parens around immediate invocations
- Require `"use strict"`;

The Good Parts

Strict white space indentation

Maximum line length

Clear All Options

Maximum number of errors

Predefined (, separated)

```
/*jslint browser: true, maxerr: 50, indent: 4 */
```

changing options updates the comment in the box at the bottom which can be copied into a source file

Configuration in Source Files

- * Specified with comments at top of JavaScript source files
- * Descriptions of all options are on the next three slides
- * Basic example

```
/*jslint browser: true, indent: 2 */  
/*global $: true, someNamespace: true */
```

- * "Good Parts" example

```
/*jslint white: true, browser: true, onevar: true, undef: true,  
nomen: true, eqeqeq: true, plusplus: true, bitwise: true,  
regexp: true, newcap: true, immed: true, strict: true */
```

Options ...

Description	option	Meaning
ADsafe	adsafe	true if ADsafe rules should be enforced. See http://www.ADsafes.org/ .
Disallow bitwise operators	bitwise	true if bitwise operators should not be allowed. (more)
Assume a browser	browser	true if the standard browser globals should be predefined. (more)
Tolerate HTML case	cap	true if upper case HTML should be allowed.
Tolerate CSS workarounds	css	true if CSS workarounds should be tolerated. (more)
Tolerate debugger statements	debug	true if debugger statements should be allowed. Set this option to <code>false</code> before going into production.
Assume <code>console</code> , <code>alert</code> , ...	devel	true if browser globals that are useful in development should be predefined. (more)
Disallow <code>==</code> and <code>!=</code>	equeqeq	true if <code>===</code> should be required. (more)
Tolerate ES5 syntax	es5	true if ES5 syntax should be allowed.
Tolerate <code>eval</code>	evil	true if <code>eval</code> should be allowed. (more)
Tolerate unfiltered <code>for in</code>	forin	true if unfiltered <code>for in</code> statements should be allowed. (more)
Tolerate HTML fragments	fragment	true if HTML fragments should be allowed. (more)

... Options ...

Require parens around immediate invocations	<code>immed</code>	<code>true</code> if immediate function invocations must be wrapped in parens <pre>var result = (function (params) { ... } ());</pre>
Strict white space indentation	<code>indent</code>	The number of spaces used for indentation (default is 4)
Tolerate sloppy line breaking	<code>laxbreak</code>	<code>true</code> if statement breaks should not be checked. (more)
Maximum number of errors	<code>maxerr</code>	The maximum number of warnings reported (default is 50)
Maximum line length	<code>maxlen</code>	The maximum number of characters in a line
Disallow dangling _ in identifiers	<code>nomen</code>	<code>true</code> if names should be checked for initial or trailing underbars sometimes used to indicate private data, but isn't really private
Require Initial Caps for constructors	<code>newcap</code>	<code>true</code> if Initial Caps must be used with constructor functions. (more)
Tolerate HTML event handlers	<code>on</code>	<code>true</code> if HTML event handlers should be allowed. (more)
Allow one <code>var</code> statement per function	<code>onevar</code>	<code>true</code> if only one <code>var</code> statement per function should be allowed. (more)
Stop on first error	<code>passfail</code>	<code>true</code> if the scan should stop on first error.

... Options

Disallow ++ and --	plusplus	true if ++ and -- should not be allowed. (more)
Predefined (, separated)	predef	An array of strings, the names of predefined global variables. <code>predef</code> is used with the <code>option</code> object, but not with the <code>/*jshint */</code> comment. Use the <code>var</code> statement to declare global variables in a script file.
Disallow insecure . and [^...]. in /RegExp/	regexp	true if . and [^...] should not be allowed in RegExp literals. These forms should not be used when validating in secure applications.
Assume Rhino	rhino	true if the Rhino environment globals should be predefined. (more)
Safe Subset	safe	true if the safe subset rules are enforced. These rules are used by ADsafe . It enforces the safe subset rules but not the widget structure rules.
Require "use strict";	strict	true if the ES5 "use strict"; pragma is required. Do not use this option carelessly.
Tolerate inefficient subscripting	sub	true if subscript notation may be used for expressions better expressed in dot notation.
Disallow undefined variables	undef	true if variables must be declared before used. (more)
Strict white space	white	true if strict whitespace rules apply.
Assume a Yahoo Widget	widget	true if the Yahoo Widgets globals should be predefined. (more)
Assume Windows	windows	true if the Windows globals should be predefined. (more)

see
"ECMAScript 5"
section

Biggest Benefits ...

- * Reduces debugging time by finding issues before changes to a web app. are deployed

Fixing issues reported by JSLint can improve browser compatibility of the JavaScript code since some browser JavaScript engines are more forgiving than others.

- * Finds misuse of functions and variables

- * all functions and variables (global or local) must be declared in a `var` statement before they are used
- * each function (named or anonymous) can only contain one `var` statement
 - * encourages declaring all local variables at the top of each function
- * global variables defined elsewhere
 - * must be listed in the `/*global: ... */` comment at top of file
 - * the "browser" option predefines many global variables that are made available by web browsers
 - * for example, "document" and "alert"
- * constructors versus normal functions
 - * functions that start with an uppercase letter cannot be directly invoked (assumed to be constructors)
 - * functions that start with a lowercase letter cannot be used with `new` (assumed to not be constructors)

... Biggest Benefits ...

- * Avoids issues with automatic determination of statement ends
 - * requires statement termination with semicolons
 - * except `for`, `function`, `if`, `switch`, `try` and `while`
 - * multi-line statements can only be broken after certain characters
 - * see "line breaking" in documentation

- * Avoids confusion over where statement blocks end
 - * requires code that appears where a block can appear to be in curly braces
 - * `if-else`, `for`, `while`, `do` and `try-catch-finally` statements

- * Avoids issues with automatic type conversions
 - * requires use of `===` and `!==` instead of `==` and `!=`

... Biggest Benefits

- * Avoids unintentional assignments in conditions of `if`, `for`, `while` and `do` statements
 - * cannot assign to a variable within a condition
- * Avoids issues with fall-through cases in `switch` statements
 - * code in all case clauses must end with a `break` statement
- * Avoids "tricky" code
 - * use of `with` - sets context for contained function calls
 - * use of `eval` - evaluates strings of JavaScript code
 - * requires use of `+= 1` and `-= 1` instead of `++` and `--`
- * Requires consistent code indentation

```
obj.method1();  
obj.method2(arg1, arg2);  
  
with (obj) {  
  method1();  
  method2(arg1, arg2);  
}
```

Part of Build Process

- * Make use of JSLint part of the build process for your web application

- * don't have to remember to run it
- * build can fail if JSLint finds any issues, requiring fixes before continuing
- * avoid unnecessary checking of files that have already passed and haven't changed
 - * by keeping results in `.jslint` files that are empty files only retained to have timestamp for the last check

- * Recipe

- * for each `.js`, `.html` and `.css` file
 - * if there is no previous `.jslint` file for this source file or the source file is newer than the `.jslint` file
 - * run JSLint on the file redirecting output to a file with the same name and an extension of `.jslint` (file will be empty if no issues are found)
 - * if the output file doesn't only have one line that contains "No problems found"
 - * send its contents to stdout
 - * delete the output file
 - * exit

may want to write output files to a different directory and/or make them hidden files by starting their names with a period

could implement with a bash script that is invoked from Ant using the `exec` task (ask me for the code if interested)

Give In!

- * Like all coding standards, you may not agree with everything JSLint wants
- * However, the benefits it provides in finding issues before deployment and enforcing code consistency far outweigh the discomfort you may feel over some of its demands