





Front End Engineer

Developer Tools and Evaluation

TOPICS

Writing valuable documentation

Debugging

Introduction to TDD in JavaScript.



Basic page manipulation

Form validation

JavaScript Use In 2003

*Custom cross-browser code to work
around differences in DOM*



HTML5 Media APIs

Canvas API

Managing Offline Application Cache

Responsive Foreground Images

Web Workers History API

Replacing Flash

jQuery Single Page Web Apps

Modernizr

Grunt

Parallax and Other Effects

matchMedia API

MV* Frameworks

Device Orientation,
Direction, and
Motion Events

JavaScript Use In

2013

Touch Events

Data connections to cross-domain
third-party web services

Node.js

Zepto

Local Storage APIs

RequireJS

GeoLocation

WebRTC

Working around browser vendor
prefixes
Form validation

Drag & Drop API

Social Media Integration

CSS Animation & Transition Events

Polyfills

postMessage API





Maintainability

Why do we care?



Most of your time is spent maintaining code



Now, you can see:

- 01 Introduction to debugging
- 02 Tracing through a section of code
- 03 Understanding error messages

<http://youtu.be/otubD6inrO4>

<http://youtu.be/OCT39W704Es>

<http://youtu.be/6JhlJQQ9zKk>



JavaScript code is getting large and complex.

How Can We Be Certain We Have High-Quality JavaScript Code?

We can't afford to have errors occur in such complex systems.

A single error in the front-end can stop a user interface from responding entirely.

That means code we can have confidence in - error free, bug free, efficient and performant, with no memory leaks



Maintainability

Who cares?



We all want to be rock stars

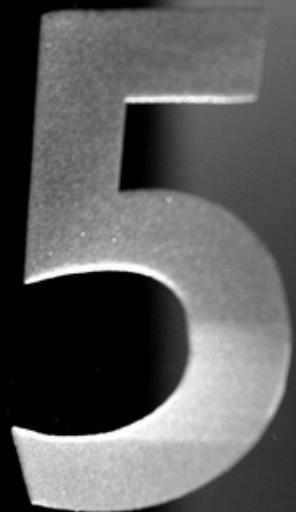
"Don't mess with my process, man! It's about the music!"



Maintainability

What is maintainable code?

Maintainable code works for five years without major changes

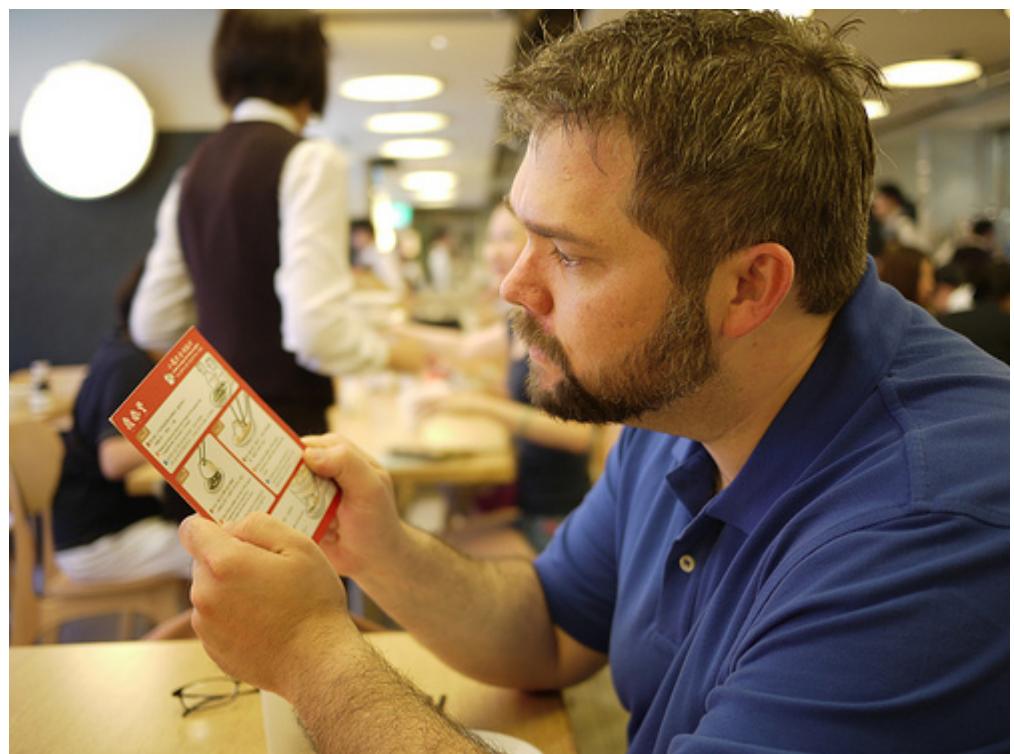


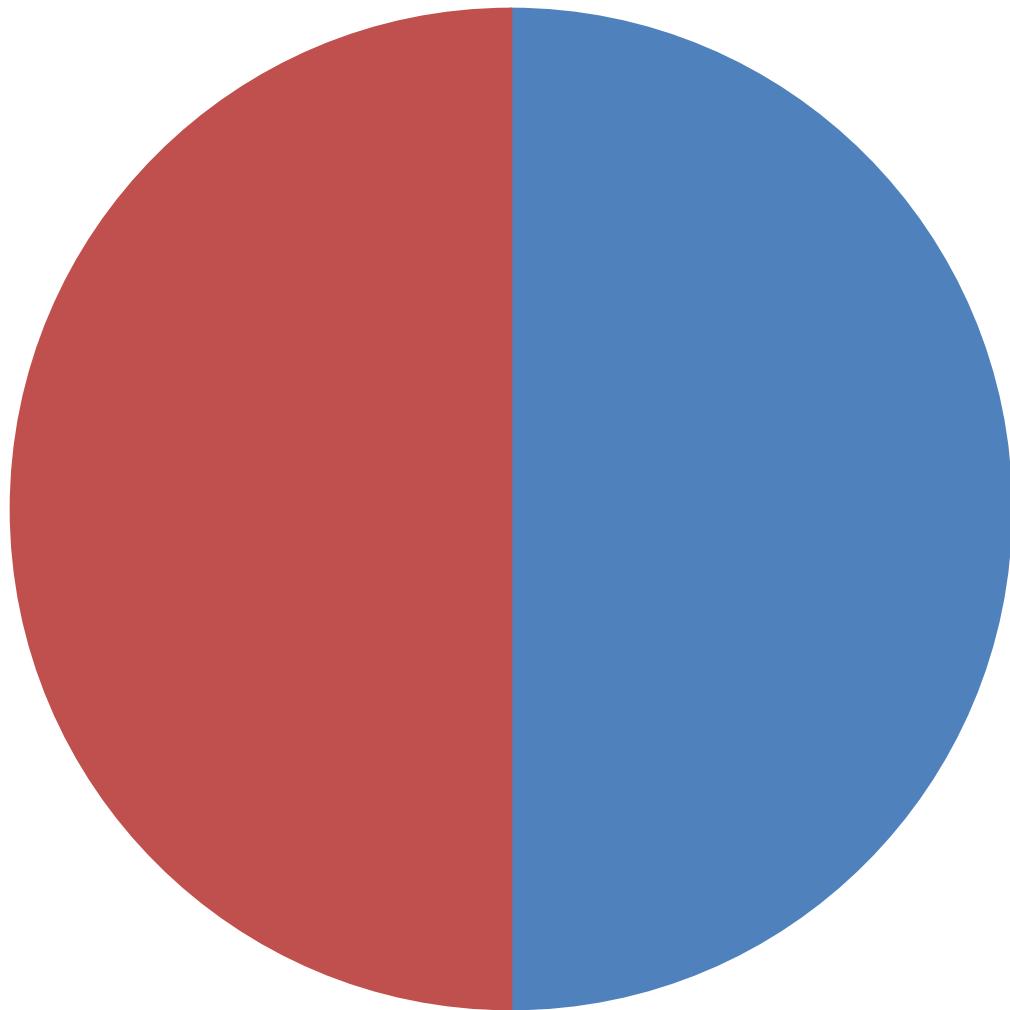
- Intuitive
- Understandable
- Adaptable
- Extendable
- Debuggable
- Testable



Be kind to your future self.

Chris Eppstein,
Creator of Compass





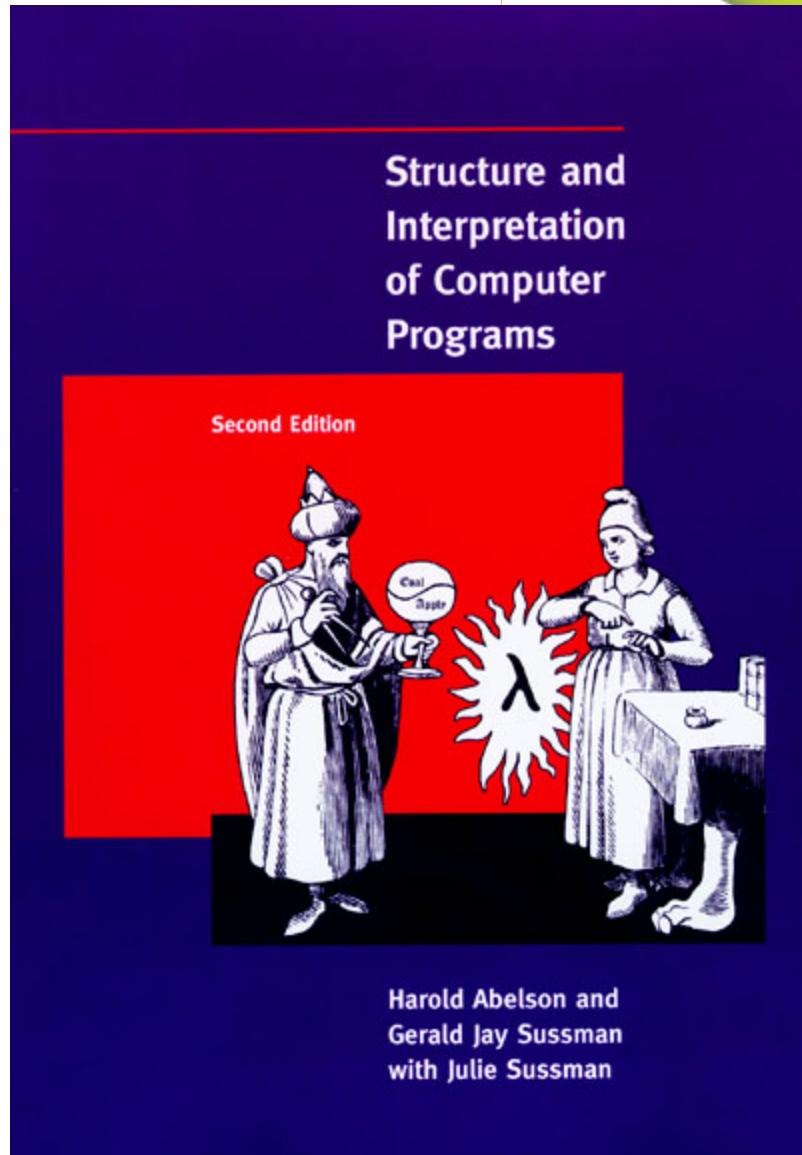


Code Style Guide

Communicating with each other through code

Programs are meant to
be read by humans and
only incidentally for
computers to execute.

H. Abelson and G. Sussman,
*The Structure and
Interpretation of Computer
Programs*





Code Conventions for the JavaScript Programming Language

This is a set of coding conventions and rules for use in JavaScript programming. It is inspired by the [Sun](#) document [Code Conventions for the Java Programming Language](#). It is heavily modified of course because [JavaScript is not Java](#).

The long-term value of software to an organization is in direct proportion to the quality of the codebase. Over its lifetime, a program will be handled by many pairs of hands and eyes. If a program is able to clearly communicate its structure and characteristics, it is less likely that it will break when modified in the never-too-distant future.

Code conventions can help in reducing the brittleness of programs.

All of our JavaScript code is sent directly to the public. It should always be of publication quality.

Neatness counts.

JavaScript Files

JavaScript programs should be stored in and delivered as `.js` files.

JavaScript code should not be embedded in HTML files unless the code is specific to a single session. Code in HTML adds significantly to pageweight with no opportunity for mitigation by caching and compression.

`<script src=filename.js>` tags should be placed as late in the body as possible. This reduces the effects of delays imposed by script loading on other page components. There is no need to use the `language` or `type` attributes. It is the server, not the script tag, that determines the MIME type.

Indentation

The unit of indentation is four spaces. Use of tabs should be avoided because (as of this writing in the 21st Century) there still is not a standard for the placement of tabstops. The use of spaces can produce a larger filesize, but the size is not significant over local networks, and the difference is eliminated by [minification](#).

Line Length

Avoid lines longer than 80 characters. When a statement will not fit on a single line, it may be necessary to break it. Place the break after an operator, ideally after a comma. A break after an operator decreases the likelihood that a copy-paste error will be masked by

<http://javascript.crockford.com/code.html>



Google JavaScript Style Guide

Revision 2.28

Aaron Whyte
Bob Jervis
Dan Pupius
Eric Arvidsson
Fritz Schneider
Robby Walker

Each style point has a summary for which additional information is available by toggling the accompanying arrow button that looks this way: . You may toggle all summaries with the big arrow button:

Toggle all summaries

Table of Contents

JavaScript Language Rules	var Constants Semicolons Nested functions Function Declarations Within Blocks Exceptions Custom exceptions Standards features Wrapper objects for primitive types Multi-level prototype hierarchies Method definitions Closures eval() with() this for-in loop Associative Arrays Multiline string literals Array and Object literals Modifying prototypes of builtin objects Internet Explorer's Conditional Comments
JavaScript Style Rules	Naming Custom toString() methods Deferred initialization Explicit scope Code formatting Parentheses Strings Visibility (private and protected fields) JavaScript Types Comments Inner Classes and Enums Compiling Tips and Tricks

Important Note

Displaying Hidden Details in this Guide

This style guide contains many details that are initially hidden from view. They are marked by the triangle icon, which you see here on your left. Click it now. You should see "Hooray" appear below.

Background

JavaScript is the main client-side scripting language used by many of Google's open-source projects. This style guide is a list of *dos* and *don'ts* for JavaScript programs.

JavaScript Language Rules

<http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>



Google JavaScript Style Guide

Revision 2.28

Aaron Whyte
Bob Jervis
Dan Pupius
Eric Arvidsson
Fritz Schneider
Robby Walker

Each style point has a summary for which additional information is available by toggling the accompanying arrow button that looks this way: . You may toggle all summaries with the big arrow button:

Toggle all summaries

Table of Contents

JavaScript Language Rules	var Constants Semicolons Nested functions Function Declarations Within Blocks Exceptions Custom exceptions Standards features Wrapper objects for primitive types Multi-level prototype hierarchies Method definitions Closures eval() with() this for-in loop Associative Arrays Multiline string literals Array and Object literals Modifying prototypes of builtin objects Internet Explorer's Conditional Comments
JavaScript Style Rules	Naming Custom toString() methods Deferred initialization Explicit scope Code formatting Parentheses Strings Visibility (private and protected fields) JavaScript Types Comments Inner Classes and Enums Compiling Tips and Tricks

Important Note

Displaying Hidden Details in this Guide

This style guide contains many details that are initially hidden from view. They are marked by the triangle icon, which you see here on your left. Click it now. You should see "Hooray" appear below.

Background

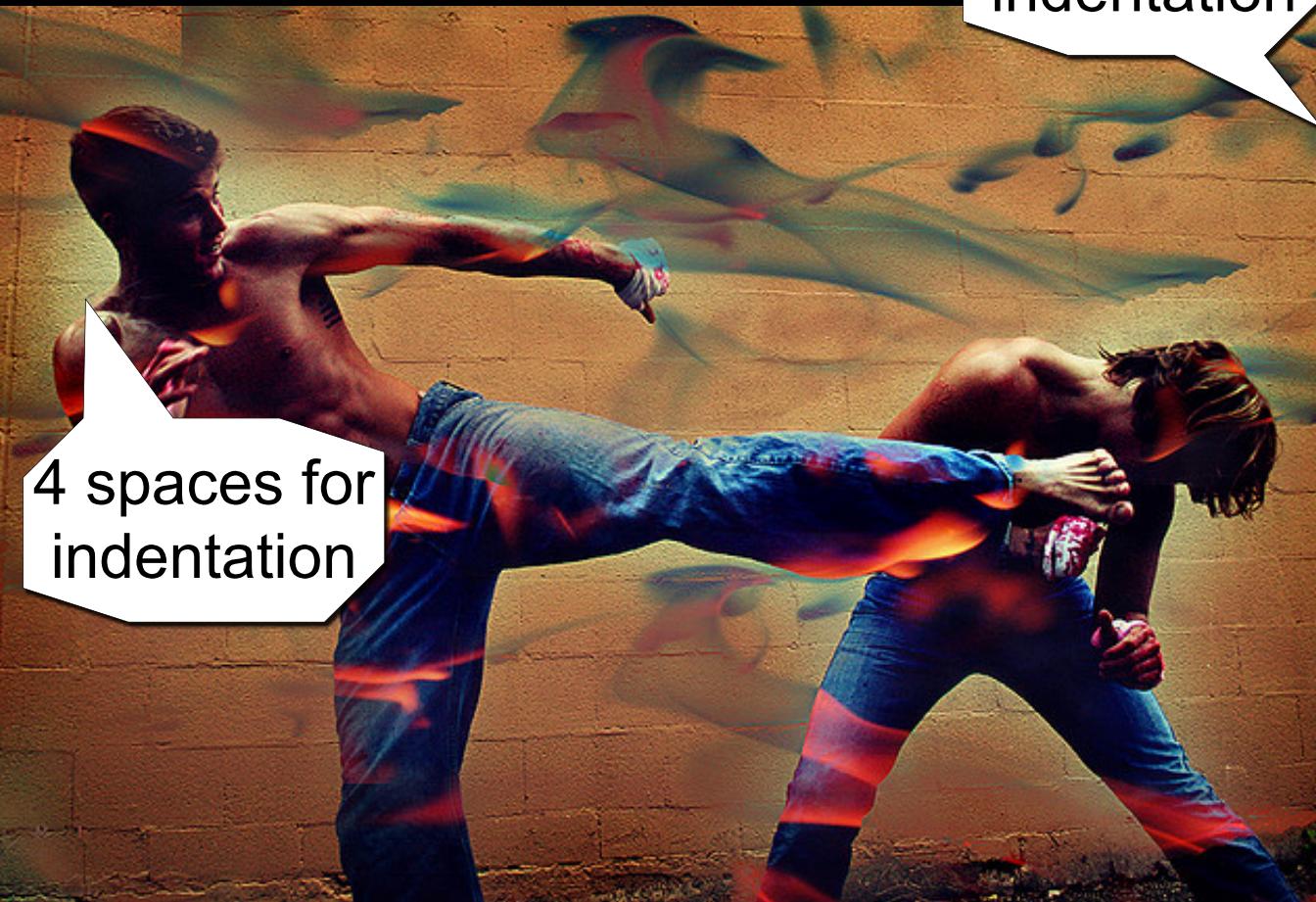
JavaScript is the main client-side scripting language used by many of Google's open-source projects. This style guide is a list of *dos* and *don'ts* for JavaScript programs.

JavaScript Language Rules

<http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml>



Tabs for
indentation



4 spaces for
indentation



```
if (wl && wl.length) {
    for (i = 0, l = wl.length; i < l; ++i) {
        p = wl[i];
        type = Y.Lang.type(r[p]);
        if (s.hasOwnProperty(p)) { if (merge && type
== 'object') {

            Y.mix(r[p], s[p]);
} else if (ov || !(p in r)) {
            r[p] = s[p];
        }
    }
}
```



```
if (wl && wl.length) {
    for (i = 0, l = wl.length; i < l; ++i) {
        p = wl[i];
        type = Y.Lang.type(r[p]);
        if (s.hasOwnProperty(p)) {
            if (merge && type == 'object') {
                Y.mix(r[p], s[p]);
            } else if (ov || !(p in r)) {
                r[p] = s[p];
            }
        }
    }
}
```



Nicholas C. Zakas
@slicknet

The less code on one line, the less likely you'll encounter a merge conflict.

twitter.com/slicknet/status/169903570047614977



```
if (found) { doSomething(); doSomethingElse(); }  
else { doAThirdThing(); doAFourthThing(); }
```



```
if (found) {  
    doSomething();  
    doSomethingElse();  
} else {  
    doAThirdThing();  
    doAFourthThing();  
}
```



Comments



Nicholas C. Zakas
@slicknet

Follow

Self-documenting code is a myth
perpetuated by those who hate to write
documentation.

<https://twitter.com/slicknet/statuses/3559283285>



```
/**  
 * Returns a new object containing all of the properties of  
 * all the supplied objects. The properties from later objects  
 * will overwrite those in earlier objects. Passing in a  
 * single object will create a shallow copy of it. For a deep  
 * copy, use clone.  
 * @method merge  
 * @for YUI  
 * @param {Object*} arguments the objects to merge.  
 * @return {object} the new merged object.  
 */  
Y.merge = function() {  
    var a = arguments, o = {}, i, l = a.length;  
    for (i = 0; i < l; i = i + 1) {  
        Y.mix(o, a[i], true);  
    }  
    return o;  
};
```

Every method



```
if (mode) {  
    switch (mode) {  
        case 1: // proto to proto  
            return Y.mix(r.prototype, s.prototype, ov, wl, 0,  
                         merge);  
        case 2: // object to object and proto to proto  
            Y.mix(r.prototype, s.prototype, ov, wl, 0, merge);  
            break; // pass through  
        case 3: // proto to static  
            return Y.mix(r, s.prototype, ov, wl, 0, merge);  
        case 4: // static to proto  
            return Y.mix(r.prototype, s, ov, wl, 0, merge);  
        default: // object to object is what happens below  
    }  
}
```

Difficult-to-understand code



```
while (element &&(element = element[axis])){ //NOTE: assignment
    if ( (all || element[TAG_NAME]) &&
        (!fn || fn(element)) ) {
        return element;
    }
}
```

Code that might seem to be wrong



Naming

• Use logical names for variables and functions

- Naming
 - Don't worry about length
- Variable names should be nouns
- Function names should begin with a verb (i.e. getName())
 - Functions return booleans should begin with "is" or "has", such as isValid() or hasItem()
- Avoid useless names such as foo and temp



```
if (wl && wl.length) {
    for (i = 0, l = wl.length; i < l; ++i) {
        p = wl[i];
        type = Y.Lang.type(r[p]);
        if (s.hasOwnProperty(p)) {
            if (merge && type == 'object') {
                Y.mix(r[p], s[p]);
            } else if (ov || !(p in r)) {
                r[p] = s[p];
            }
        }
    }
}
```

```
// Variables, functions, properties methods  
var myName = "Nicholas";  
  
function sayName() {  
    alert(myName);  
}  
  
var person = {  
    name: "Nicholas",  
    sayName: function() {  
        alert(this.name);  
    }  
};
```

Camel Casing



What about acronyms?

```
var element = document.getElementById("my-div");
```

```
element.innerHTML = "Hello world!";
```

NO!

```
var xhr = new XMLHttpRequest();
```

NO

Camel Casing



```
// Constant-like variables
var HOVER_CLASS = "mouse-over";

// Constructors
function Person(name) {
    this.name = name;
}

var me = new Person("Nicholas");
```

Camel Casing- Exceptions

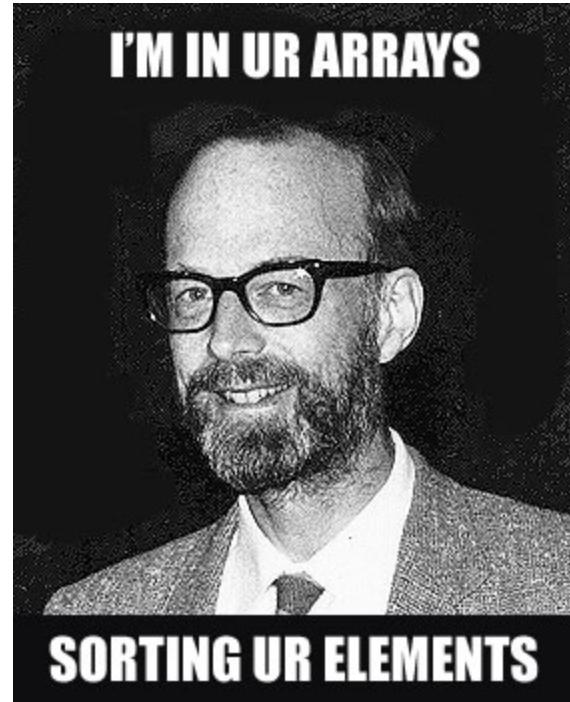


Programming Practices

Small patterns for common problems

There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies.

C.A.R. Hoare,
Quicksort Developer





Presentation
(CSS)

Behavior
(JavaScript)

Data/Structure
(HTML)



Don't cross
the streams



```
<button onclick="doSomething()">Click Me</button>
```

Keep JavaScript out of HTML



```
var element = document.getElementById("container") ;  
element.innerHTML = "<div class=\"popup\"></div>" ;
```

Keep HTML out of JavaScript



```
.foo {  
  width: expression(document.offsetWidth + "px");  
}
```

Keep JavaScript out of CSS



```
var element = document.getElementById("container") ;  
element.style.color = "red";  
element.style.cssText = "background:blue; border:1px solid red";
```

Keep CSS out of JavaScript



```
//the wrong way!!!
function handleClick(event) {

    var popup = document.getElementById("popup");
    popup.style.left = event.clientX + "px";
    popup.style.top = event.clientY + "px";
    popup.className = "reveal";

}
```

Event handlers should only
handle events



```
//better, but still wrong
function handleClick(event) {
    showPopup(event);
}

function showPopup(event) {
    var popup = document.getElementById("popup");
    popup.style.left = event.clientX + "px";
    popup.style.top = event.clientY + "px";
    popup.className = "reveal";
}
```

Don't pass the event object
around



```
//win!!  
function handleClick(event) {  
    showPopup(event.clientX, event.clientY);  
}  
  
function showPopup(x, y) {  
    var popup = document.getElementById("popup");  
    popup.style.left = x + "px";  
    popup.style.top = y + "px";  
    popup.className = "reveal";  
}
```

Properly separated event
handling



```
//don't add new methods
Array.prototype.awYeah = function() {
    alert("Aw yeah!");
};

//don't override methods
YUI.use = function() {
    alert("Aw yeah!");
};
```

Don't modify objects you don't own

If you didn't define the object yourself, you
don't own it



Maintainable JavaScript: Don't modify objects you don't own

Posted at March 2, 2010 09:00 am by Nicholas C. Zakas

Tags: [JavaScript](#), [Maintainable](#)

The first talk I gave after arriving at Yahoo! was entitled [Maintainable JavaScript \(video\)](#). As with most topics I write or speak about, I didn't think it would be terribly controversial. The basis of the talk is that hacking around on your own and writing code in an enterprise environment are two different things. Web developers are truly unique in that none of us learned what we know in school; we all began as hobbyists one way or another and taught ourselves most (if not all) of what we know.

Professionalization

The professionalization of web development has been a difficult journey because of our disparate beginnings. Even those who end up at large companies such as Yahoo! inevitably began on their own, hacking around. Perhaps you were even "the web guy" at a small company and could do pretty much whatever you wanted. When the large companies started tapping this previously undiscovered resource, it brought a lot of hackers into a corporate environment where they were met with constraints. No longer a lone soldier in a small battle, all of these self-taught, self-directed individuals had to figure out how to work together as a team.

At the time that I gave the talk (2007), web development was evolving into front-end engineering and people were having trouble with the transition. Smart folks like Nate Koechley talked about the [professionalization of front-end engineering \(video\)](#) and how our discipline was evolving. My talk was aimed at the same goal: helping front-end engineers adapt to JavaScript development in a team environment by making sure that their code was as maintainable as possible.

Why can't I modify objects I don't own?

I still get email and comments about Maintainable JavaScript, and the most popular question is, "why can't I modify objects I don't own?" JavaScript is, of course, a dynamic language that allows you to add and remove objects and their members at any point in time. For many, this is precisely why they enjoy the language: there are very few constraints imposed by the language. And I was telling them not to do this. Why?

Dependability

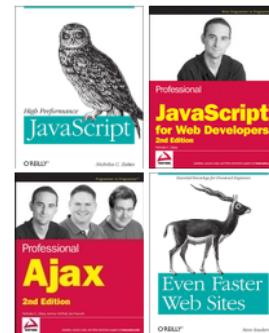
The simple explanation is that an enterprise software product needs a consistent and dependable execution environment to be maintainable. In other languages, you consider already-existing objects as libraries for you to use in order to complete your task. In JavaScript, people saw already-existing objects as a playground in which you could do anything you wanted. My point was that you should treat the already-existing JavaScript objects as you would a library of utilities. Don't override methods, don't add new methods, don't remove existing methods.

When you're the only one working on a project, it's easy to get away with these types of modifications because you know them and expect them. When working with a team on a large project, making changes

More of Me

Twitter: [@slicknet](#)
LinkedIn: [Resume](#)
GitHub: [Projects](#)
Slideshare: [Presentations](#)

My Books



<http://nczonline.net/blog/2010/03/02/maintainable-javascript-don-t-modify-objects-you-down-own/>

was an incredible mess and I'd be very happy if no engineers ever had to go through a similar exercise.



```
function handleClick(event) {  
    showPopup(event.clientX, event.clientY);  
}  
  
function showPopup(x, y) {  
    var popup = document.getElementById("popup");  
    popup.style.left = x + "px";  
    popup.style.top = y + "px";  
    popup.className = "reveal";  
}
```

Avoid global functions and
variables



```
var Controller = {  
    handleClick: function(event) {  
        this.showPopup(event.clientX, event.clientY);  
    },  
  
    showPopup: function (x, y) {  
        var popup = document.getElementById("popup");  
        popup.style.left = x + "px";  
        popup.style.top = y + "px";  
        popup.className = "reveal";  
    }  
};
```

Avoid global functions and variables

Create a single global (if necessary) and attach everything to it



```
var Controller = {  
  addClass: function(element, className) {  
    element.className += " " + className;  
  }  
};
```

Throw your own errors
When you know a function will fail



```
var Controller = {  
    addClass: function(element, className) {  
        if (!element) {  
            throw new Error("addClass: 1st argument missing.");  
        }  
        element.className += " " + className;  
    }  
};
```

Throw your own errors

When you know a function will fail



```
var Controller = {  
  process: function(items) {  
    if (items != null) {  
      items.sort();  
      items.forEach(function(item) {  
        //do something  
      }) ;  
    }  
  } ;
```

Avoid null comparisons



```
var Controller = {  
  process: function(items) {  
    if (items instanceof Array) {  
      items.sort();  
      items.forEach(function(item) {  
        //do something  
      }) ;  
    }  
  } ;
```

Avoid null comparisons

Test for precisely what you want to know if it matters

• Use **instanceof** to test for specific object types

Avoid null comparisons

- **object instanceof MyType**

• Use **typeof** to test for primitive types

- **typeof value == "string"**
- BEWARE: **typeof null == "object"**



```
function validate(value) {  
    if (!value) {  
        alert("Invalid value");  
        location.href = "/errors/invalid.php";  
    }  
}
```

Separate config data

```
var config = {  
    urls: {  
        invalid: "/errors/invalid.php"  
    },  
    strs: {  
        invalidmsg: "Invalid value"  
    }  
};
```

Separate config data

```
function validate(value) {  
    if (!value) {  
        alert(config.strs.invalidmsg);  
        location.href = config.urls.invalid;  
    }  
}
```



- All URLs needed by the JavaScript
- Any strings that are displayed to the user
- Any HTML that needs to be created from JavaScript
- Settings (i.e., items per page)
- Repeated unique values
- Any value that may change in the future



github

Search... Explore Gist Blog Help nzakas 322 Admin Unwatch Fork Pull Request 13 1

nzakas / props2js

Code Network Pull Requests 0 Issues 1 Wiki 0 Graphs

Tool to convert Java properties files into JavaScript — [Read more](#)

ZIP SSH HTTP Git Read-Only git@github.com:nzakas/props2js.git Read+Write access

branch: master Files Commits Branches 1 Tags Downloads 1

Latest commit to the master branch

Updated readme and changelog

nzakas authored 5 months ago commit 87e22d3ed6

props2js /

name	age	message	history
lib	5 months ago	First commit [nzakas]	
src	5 months ago	First commit [nzakas]	
tests	5 months ago	First commit [nzakas]	
.gitignore	2 years ago	Small bug fixes [nzakas]	
CHANGELOG	5 months ago	Updated readme and changelog [nzakas]	
LICENSE	5 months ago	First commit [nzakas]	
README.md	5 months ago	Updated readme and changelog [nzakas]	

<https://github.com/nzakas/props2js>

README.md

Props2js



Now, you can see:

04 Using debuggers

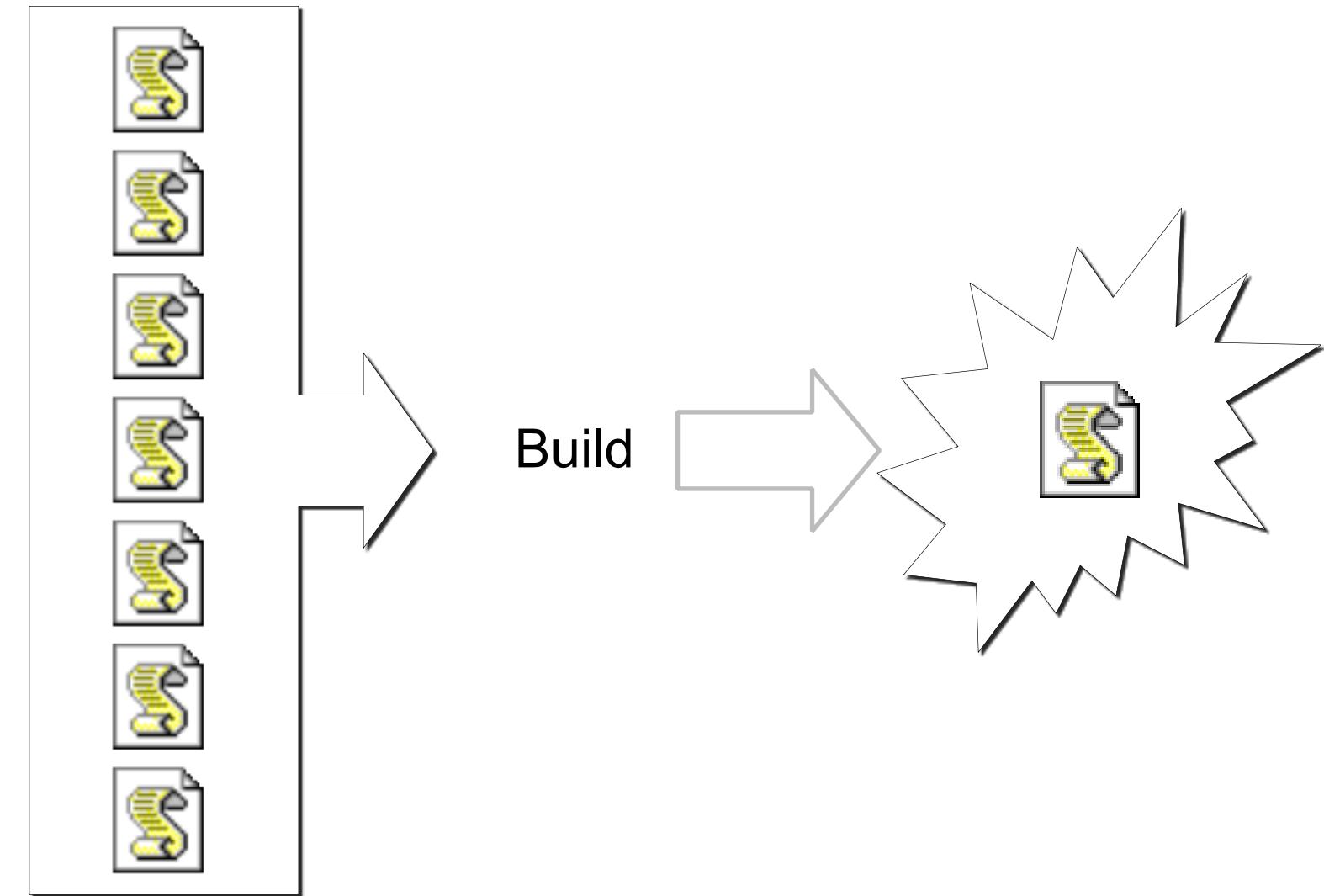
<http://youtu.be/SqBEu-W7AXg>



Automation

Make everyone's life easier

Build Process





Build

Add/Remove
Debugging

Validate
Code

Concatenate
Files

Test Code

Generate
Documentation

Minify Files

Deploy
Files

[Search projects](#)[Project Home](#) [Downloads](#) **Wiki** [Issues](#) [Source](#)Search for

★ AntTasks

*Ant tasks currently available*Updated Aug 13, 2009 by someone@gmail.com

This page shows a list of ant task currently available in the JS Build Tools package.

preprocess

The JS Build Tools package comes with a Javascript preprocessor engine. This Ant task processes statements like #ifdef, #ifndef or #endif.

Parameters

- infile - Input file to process.
- outfile - Output file to write the processed contents to.
- defines - Comma separated list of defined values to check for.

Here is an example on how to use the preprocessor in a JS file:

```
// #ifdef somevalue1
someFunction();
// #endif
// #ifndef somevalue1
someOtherFunction();
```

<https://github.com/moxiecode/js-build-tools>

The preprocessor enables you to include/exclude code depending in defined values. This is very useful if you want to for example remove all debug code when making a distribution package.

yuicompress

This task compresses the specified script using the YUICompressor package. The YUICompressor will remove all unneeded white space and obfuscate long variable names.

Parameters

Add/Remove Debugging



Index

DRAFT

Getting Started

[An introduction to JSDoc 3](#)

A quick-start to documenting JavaScript with JSDoc.

JSDoc 3 Tag Dictionary

[@augments or @extends](#)

This object adds onto a parent object.

[@borrows](#)

This object uses something from another object.

[@constructor or @class](#)

This function is intended to be called with the "new" keyword.

[@classdesc](#)

Use the following text to describe the entire class.

[@constant or @const](#)

Document an object as a constant.

[@copyright](#)

[todo] Document some copyright information.

[@constructs](#)

[todo] This function member will be the constructor for the previous class.

[@default](#)

Document the default value.

[@deprecated](#)

[todo] This is no longer the preferred way.

<http://usejsdoc.org>

[@event](#)

[todo] Document an event.

[@example](#)

[todo] Provide an example.

[@throws or @exception](#)

[todo] Describe what errors could be thrown.

[@exports](#)

Generate
Documentation

[Register](#) | [Login](#)

Search docs and APIs

[Home](#) | [Quick Start](#) | [Documentation](#) | [Community](#) | [Contribute](#) | [Other Projects](#)[YUI](#) > [Other Projects](#) > [YUI Doc](#)

YUI Doc

[Project Home](#) | [Wiki](#) | [Download](#) | [Forum](#) | [Calendar](#) | [Browse Source](#) | [Pull Request](#) | [View Tickets](#) | [New Ticket](#)

YUI Doc is a tool written in Python that generates beautiful, organized, searchable API documentation for your JavaScript code. You will typically use YUI Doc as part of your build process. YUI Doc is comment-driven and is compatible with a variety of coding styles. You do not need to be using YUI Library JavaScript in order to use YUI Doc.

Reporting Defects / Making Enhancement Requests

YUILibrary.com is the proper location for reporting defects found in the YUI Doc code as well as for logging enhancement requests for consideration for future updates. You can review existing tickets filed for YUI Doc by clicking the [View Tickets](#) link at the top of this page. Please review the [YUI guidelines for filing defects and making enhancement requests](#) before adding new tickets to the YUI Doc project using the [New Ticket](#) link above.

***Only logged in users can submit bugs and feature requests.**

Generate Documentation

Product Information

Current Version: 1.0.0b1

Release Date: 12/05/2008

- [Download Release](#)
- [Getting Started](#)
- [License](#)
- [File a Ticket](#)
- [View Tickets](#)
- [Project Forum](#)

Developer Information

Development Version: NEXT

Tentative Release Date: 0

- [Dashboard](#)
- [Browse Source](#)
- [Recent Commits](#)
- [Issue a Pull Request](#)

Recent Forum Posts

Re: yuidocjs and extended yui3 classes

ed yui3
12

Re: yuidocjs and extended yui3 classes

Peter Peterson on 02/28/12
Re: yuidocjs and extended yui3

<http://yuilibrary.com/projects/yuidoc/>



docco.coffee

¶ Docco is a quick-and-dirty, hundred-line-long, literate-programming-style documentation generator. It produces HTML that displays your comments alongside your code. Comments are passed through [Markdown](#), and code is passed through [Pygments](#) syntax highlighting. This page is the result of running Docco against its own source file.

If you install Docco, you can run it from the command-line:

```
docco src/*.coffee
```

...will generate linked HTML documentation for the named source files, saving it into a `docs` folder.

The [source for Docco](#) is available on GitHub, and released under the MIT license.

To install Docco, first make sure you have [Node.js](#), [Pygments](#) (install the latest dev version of Pygments from [its Mercurial repo](#)), and [CoffeeScript](#). Then, with NPM:

```
sudo npm install docco
```

Partners in Crime:

- If Node.js doesn't run on your platform, or you'd prefer a more convenient package, get [Ryan Tomayko's Rocco](#), the

<http://jashkenas.github.com/docco/>

- If Python's more your speed, take a look at [Nick Fitzgerald's Pycco](#).
- For Clojure fans, [Fogus's Marginalia](#) is a bit of a departure from "quick-and-dirty", but it'll get the job done.
- Lua enthusiasts can get their fix with [Robert Gieseke's Locco](#).

Generate Documentation



JSHint (and JSLint) allow you to perform an analysis on your code, without it actually running.

It spots what look like errors in your code so you can fix them early before they cause any issues - things like undeclared variables, or variables declared but not used (good for spotting spelling mistakes!).



JSHint, a JavaScript Code Quality Tool

SonarQube™ » Discussing Cyclomatic Com... Grunt: The JavaScript Task Runner introduction-1.3.1.js JSHint, a JavaScript Code Quality Tool

Home About Docs Install Hack Blog

JS Hint

Create a Website and an Online Store with LightCMS. Sign Up For Free.

ads via Carbon

JSHint has found potential problems in your code. See [detailed report](#).

```
1 /**
2  Utility methods for handling dates
3
4  @class Dates
5  @static
6 */
7
8 var Dates = (function($) {
9   "use strict";
10
11 /**
12  Lets you know if a supplied date is a Monday
13
14  @method isMonday
15  @param {Date} dateObj date to test
16  @return {Boolean} true if supplied date is a Monday
17 */
18
19 function isMonday(dateObj) {
20   var inputDayOfTheWeek = dateObj.getDay(),
21       mondayDayOfTheWeek = 1;
22   return (inputDayOfTheWeek === mondayDayOfTheWeek);
23 }
24
25 return {
26   isMonday: isMonday
27 };
28 })(jQuery);
```

Validate Code



JSLint

The [JavaScript](#) Code Quality Tool

Edition 2012-04-15

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

Validate Code

[JSLint](#) [Syntax Tree](#) [clear](#)

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

Warning! JSLint will hurt your feelings.

<http://jslint.com>

Assume [Node.js](#)

Assume [Rhino](#)

Assume a [Yahoo Widget](#)

Assume Windows

Stop on first error

Safe Subset

Tolerate debugger statements

Tolerate == and !=

Tolerate ES5 syntax

Tolerate eval

Tolerate [unfiltered](#) for in

Tolerate uncapitalized constructors

Tolerate dangling _ in identifiers

Tolerate ++ and --

Tolerate missing 'use strict' pragma

Tolerate stupidity

Tolerate inefficient subscripting

Tolerate many var statements per function

Tolerate messy white space

Tolerate CSS workarounds

Tolerate HTML case

Tolerate HTML event handlers

JSHint is a tool to detect errors and potential problems in JavaScript code.

```
1 // Your code goes here.
```

Validate Code

<http://jshint.com>

Lint

Warn

- About debugging code
- About unsafe `for..in`
- About `== null`
- About `eval`
- About unsafe line breaks
- When bitwise operators are used

Assume

- Browser
- Development (`console`, etc.)
- jQuery

[Register](#) | [Login](#)

Search docs and APIs

[Home](#) | [Quick Start](#) | [Documentation](#) | [Community](#) | [Contribute](#) | [Other Projects](#)

YUI > Other Projects > YUI Compressor

YUI Compressor

[Project Home](#) | [Wiki](#) | [Download](#) | [Forum](#) | [Calendar](#) | [Browse Source](#) | [Pull Request](#) | [View Tickets](#) | [New Ticket](#)

YUI Compressor is an open source tool that supports the compression of both JavaScript and CSS files. The JavaScript compression removes comments and white-spaces as well as obfuscates local variables using the smallest possible variable name. CSS compression is done using a regular-expression-based CSS minifier.

Current Production Release: 2.4.7

Next Release:

The next Compressor release is in the initial planning stage.

Reporting Defects / Making Enhancement Requests

YUILibrary.com is the proper location for reporting defects found in the YUI Compressor code as well as for logging enhancement requests for consideration for future updates. You can review existing tickets filed for YUI Compressor by clicking the [View Tickets](#) link at the top of this page. Please review the [YUI guidelines for filing defects and making enhancement requests](#) before adding new tickets to the YUI Compressor project using the [New Ticket](#) link above.

*Only logged in users can submit bugs and feature requests.

<http://yuilibrary.com/projects/yuicompressor/>

Minify Files

Product Information

Current Version: 2.4.7

Release Date: 11/14/2011

- [Download Release](#)
- [License](#)
- [File a Ticket](#)
- [View Tickets](#)
- [Project Forum](#)

Developer Information

Development Version: in planning

Tentative Release Date: Not Planned

- [Dashboard](#)
- [Browse Source](#)
- [Recent Commits](#)
- [Issue a Pull Request](#)

Recent Forum Posts

- [Source maps?](#)
Matt Snider on 03/27/12
- [Re: Why YUI Compressor does not minimize Leading zero...](#)
Tubal Martin on 03/20/12
- [Re: error: __YUICSSMIN_PRESERVE_CANDIDATE_C](#)
Tim Berman on 03/19/12
- [Re: error:](#)

github Search... Explore Gist Blog Help nzakas 322

mishoo / UglifyJS Watch Fork 2,548 168

Code Network Pull Requests 26 Issues 103 Graphs

JavaScript parser / mangler / compressor / beautifier library for NodeJS — [Read more](#)
<http://marijn.haverbeke.nl/uglifyjs>

[ZIP](#) [HTTP](#) Git Read-Only <https://github.com/mishoo/UglifyJS.git> Read-Only access

branch: master [Files](#) Commits Branches 2 Tags 19 Downloads

Latest commit to the **master** branch
Merge pull request #387 from rvanvelzen/master + ...
mishoo authored 2 hours ago

[UglifyJS /](#)

name	age	message	history
bin	8 days ago	Merge pull request #342 from int3/master [mishoo]	
lib	2 hours ago	Fix for issue #386 [rvanvelzen]	
test	6 hours ago	Implement #368 (too many parentheses) [Robert Gust-Bardon]	
tmp	8 months ago	Added an utility that allows walking depth-first easily. [mishoo]	

<https://github.com/mishoo/UglifyJS/>

docstyle.css	a year ago	NPM changes [mishoo]
package.json	a month ago	move consolidator tests to test/ dir and run them via 'npm test' [paulbaumgart]
uglify-js.js	2 months ago	Consolidate null, Boolean, and String values [Robert Gust-Bardon]



Closure Tools



747

Minify Files

Closure Tools

Compiler

▶ Documentation

Project on Google Code

FAQ

Community

Library

Templates

Linter

FAQ

Blog

What is the Closure Compiler?

The Closure Compiler is a tool for making JavaScript download and run faster. It is a true compiler for JavaScript. Instead of compiling from a source language to machine code, it compiles from JavaScript to better JavaScript. It parses your JavaScript, analyzes it, removes dead code and rewrites and minimizes what's left. It also checks syntax, variable references, and types, and warns about common JavaScript pitfalls.

The Closure Compiler has been integrated with [Page Speed](#), which makes it easier to evaluate the performance gains you can get by using the compiler.

How can I use the Closure Compiler?

You can use the Closure Compiler as:

- An open source Java application that you can run from the command line.
- A simple web application.
- A RESTful API.

To get started with the compiler, see "How do I start" to the right.

What are the benefits of using Closure Compiler?

<https://developers.google.com/closure/compiler/>



Unit tests are a fairly new concept to JS developers, but familiar to those of many other languages.

A unit test is a small function that calls one of the functions in your code with a set of known inputs and checks that the output of each call is what was expected.



We use Jasmine, but there are many other unit test frameworks out there.

You create a HTML file, include the Jasmine library, the JS file you wrote that you wish to test, and the JS file you wrote your unit tests in.

It then runs the tests automatically and shows you the results.



The screenshot shows a web browser window with the following details:

- Title Bar:** The title bar says "introduction-1.3.1.js".
- Address Bar:** The address bar shows the URL "pivotal.github.io/jasmine/".
- Header:** The header includes links for "SonarQube™ » Discussing Cyclomatic Complexity" and "Grunt: The JavaScript Task Runner".
- Content Area:** The main content area displays the Jasmine logo (a green flower) and the text "Jasmine".
- Section Headings:** Below the logo, there are sections titled "introduction-1.3.1.js", "Suites: `describe` Your Tests", and "Specs".
- Text Content:** The "introduction-1.3.1.js" section describes Jasmine as a behavior-driven development framework for testing JavaScript code. It mentions that it does not depend on any other frameworks, does not require a DOM, and has a clean, obvious syntax.
- Code Snippet:** The "Suites: `describe` Your Tests" section shows a snippet of JavaScript code for defining a test suite:

```
describe("A suite", function() {
  it("contains spec with an expectation", function() {
    expect(true).toBe(true);
  });
});
```

The right side of the browser window has a dark teal sidebar containing the text "Powered by PIVOTAL LABS" next to the Pivotal Labs logo.



Now, you can see:

05 JavaScript Testing With Jasmine Introduction

<http://youtu.be/2CiKlcABwW0>



Now, you can see:

06 JavaScript Testing With Jasmine The First Application

http://youtu.be/_e53fKh1XuQ



Now, you can see:

07 JavaScript Testing With Jasmine Bundles for textmate

<http://youtu.be/sHn2VcYKIFs>



Assuming the tests pass, you will see the green bar at the top of the screen, together with a list of the individual tests that ran and their results.

Jasmine Spec Runner
file:///Users/dennis.odell/Documents/Git/akqa-360-image-navigator-component/src/build/_SpecRunner.html
Jasmine 1.3.1 revision 1354556913
finished in 0.045s
No try/catch
Passing 32 specs
AKQA 360 Image Component
refresh method
works
showThreesixty method
works
reveal method
works
imageLoaded method
works
loadImage method
works
hidePreviousFrame method
works
showCurrentFrame method
works
getNormalizedCurrentFrame method
works
render method
works
getPointerEvent method
works
ignores mouse move events
supports originalEvent
works with no arguments
trackPointer method
works
disableSelection method
works
move method
works
works with 1
handles inputs less than 0
handles inputs greater than defaults.maxImg



Here's an example unit test for our isMonday method.

Groups of tests in Jasmine are wrapped in a call to its 'describe' method, and individual tests are wrapped in a call to its 'it' method. You can then describe each test with a string before containing the test itself within a function.

The unit tests execute the isMonday method with known inputs, and check the output is as expected with the 'expect' and 'toBe' methods of Jasmine.

Unit tests prove that your code works as expected, which gives you and your team confidence in it.

Example

```
describe('Calculator', function () {
  var calc;
  beforeEach(function () {
    calc = new Calculator();
  });
  it('should multiply', function () {
    expect(calc.multiply(6, 7)).toBe(42);
  });
}) ;
```



Now, you can see:

08 JavaScript Testing With Jasmine Matchers

<http://youtu.be/TpDXZ5Bpf9k>



We want to test code in isolation
here the code is the 'keypress' event handler
and isolation means not invoking the getMatch() method

```
'keypress': function (element, event) {
    var pattern = this.element.val();
    pattern += String.fromCharCode(event.charCodeAt);
    var match = this.getMatch(pattern);
    if (match) {
        event.preventDefault();
        this.element.val(match);
    }
}
```



We can mock the `getMatch()` method
decide how the mock should behave
verify that the mocked method was called correctly

```
spyOn(autoComplete, 'getMatch').andReturn('monique');

$('#name').trigger($.Event('keypress', {charCode: 109}));

expect(autoComplete.getMatch).toHaveBeenCalledWith('m');

expect($('#name')).toHaveValue('monique');
```



Global functions are properties of the window object

```
openPopup: function (url) {
    var popup = window.open(url, '_blank', 'resizable');
    popup.focus();
}
```

```
var popup;
spyOn(window, 'open').andCallFake(function () {
    popup = {
        focus: jasmine.createSpy()
    };
    return popup;
});

autoComplete.openPopup('zealake.com');

expect(window.open).toHaveBeenCalledWith('zealake.com', '_blank', 'resiza
expect(popup.focus).toHaveBeenCalled();
```



Constructors are functions with this being the object to construct

```
this.input = new window.AutoComplete(inputElement, {  
    listUrl: this.options.listUrl  
});  
this.input.focus();
```

```
spyOn(window, 'AutoComplete').andCallFake(function () {  
    this.focus = jasmine.createSpy();  
});  
  
expect(window.AutoComplete.callCount).toBe(1);  
var args = window.AutoComplete.mostRecentCall.args;  
expect(args[0]).toBe('#name');  
expect(args[1]).toEqual({listUrl: '/someUrl'});  
  
var object = window.AutoComplete.mostRecentCall.object;  
expect(object.focus).toHaveBeenCalled();
```



We want the tests to be fast
So don't use Jasmine waitsFor()
But we often need to wait

- For animations to complete
- For AJAX responses to return

```
delayHide: function () {  
    var self = this;  
    setTimeout(function () {  
        self.element.hide();  
    }, this.options.hideDelay);  
}
```



Use Jasmine's mock clock

Control the clock explicitly

Now the test completes in milliseconds without waiting

```
jasmine.Clock.useMock();  
  
autoComplete.delayHide();  
  
expect($('#name')).toBeVisible();  
  
jasmine.Clock.tick(500);  
  
expect($('#name')).not.toBeVisible();
```



`new Date()` tends to return different values over time

Actually, that's the whole point :)

But how do we test code that does that?

We cannot expect on a value that changes on every run

We can mock the `Date()` constructor!

```
var then = new Date();
jasmine.Clock.tick(42000);
var now = new Date();

expect(now.getTime() - then.getTime()).toBe(42000);
```



Keep Date() and setTimeout() in sync

```
jasmine.GlobalDate = window.Date;

var MockDate = function () {
    var now = jasmine.Clock.defaultFakeTimer.nowMillis;
    return new jasmine.GlobalDate(now);
};
MockDate.prototype = jasmine.GlobalDate.prototype;
window.Date = MockDate;
jasmine.getEnv().currentSpec.after(function () {
    window.Date = jasmine.GlobalDate;
});
```



To test in isolation

To vastly speed up the tests

Many options

- can.fixture
- Mockjax
- Sinon

```
can.fixture('/getNames', function (original, respondWith) {
    respondWith({list: ['rachel', 'lakshmi']});
});
autoComplete = new AutoComplete('#name', {
    listUrl: '/getNames'
});
jasmine.Clock.tick(can.fixture.delay);
```

```
    respondWith(500); // Internal server error
```



Supply the markup required by the code
Automatically cleanup markup after every test

Various solutions

- Built into QUnit as #qunit-fixture
- Use jasmine-jquery

```
var fixtures = jasmine.getFixtures();
fixtures.set(fixtures.sandbox());

$('<input id="name">').appendTo('#sandbox');

autoComplete = new AutoComplete('#name');
```



How do we test that an event was triggered?
Or prevented from bubbling?
Use jasmine-jquery!

```
'keypress': function (element, event) {
    var pattern = this.element.val() +
                  String.fromCharCode(event.charCode);
    var match = this.getMatch(pattern);
    if(match) {
        event.preventDefault();
        this.element.val(match);
    }
}
```

```
keypressEvent = spyOnEvent('#name', 'keypress');

$('#name').trigger($.Event('keypress', {charCode: 109}));

expect(keypressEvent).toHaveBeenCalled();
```



Now, you can see:

09 JavaScript Testing With Jasmine JavaScript Koans

<http://youtu.be/it4BXV49B9g>



Now, you can see:

10 JavaScript Automated Testing Ruby Gem

http://youtu.be/PWUJHN_0L4g



Now, you can see:

11 JavaScript Automated Testing Using LiveReload

http://youtu.be/PWUJHN_0L4g



Now, you can see:

12 JavaScript Automated Testing jasmine headless webkit

<http://youtu.be/DohifjP2ThQ>



We use three main tools for testing. The first is Grunt, a JS task runner built on Node.js.

It can be configured to run JSHint and Jasmine unit tests automatically via plugin ‘tasks’.

This gets run on the developer’s local machine before their code is committed.



Grunt: The JavaScript Task Runner

SonarQube™ » Discussing Cyclomatic Complexity

Grunt: The JavaScript Task Runner

Getting Started Plugins Documentation API



GRUNT

The JavaScript Task Runner

Latest Version

- Stable: v0.4.1
- Development: N/A

The conference for common sense JavaScript applications, July 31-August 1 in Boston.

Ads by [Bocoup](#).

Latest News

[Grunt 0.4.1 released](#)
March 13, 2013

[Grunt 0.4.0 released](#)
February 18, 2013

Why use a task runner?

In one word: automation. The less work you have to do when performing repetitive tasks like minification, compilation, unit testing, linting, etc, the easier your job becomes. After you've configured it, a task runner can do most of that mundane work for you—and your team—with basically zero effort.

Why use Grunt?

The Grunt ecosystem is huge and it's growing every day. With literally hundreds of plugins to choose from, you can use Grunt to automate just about anything with a minimum of effort. If someone hasn't already built what you need, authoring and publishing your own Grunt plugin to npm is a breeze.

Available Grunt plugins

Many of the tasks you need are already available as Grunt Plugins, and new plugins are published every day. While the [plugin listing](#) is more complete, here's a few you may have heard of.

Build



The second tool we use for testing is BrowserStack.

This is a site that spins up virtual machines on the fly which contain only browsers and dev tools.

You can select an OS and a browser, give it a URL, and it'll spin up a VM which you can then interact with as if you were using the real thing.



Desktop & mobile OSes are supported, and it allows the creation of a ‘tunnel’ through which you can run local sites from your machine within their VMs.

Very handy for mobile and old IE testing!



Dashboard

SonarQube™ » Discussing Cyclomatic Complexity

www.browserstack.com/start#os=Windows&os_version=8&browser=IE&browser_version=10.0&zoom_to_fit=true&url=http://www.datsun.com&resolution=1024x768&s

Reader

BrowserStack Live

Windows 8

IE 10.0

1024 x 768

Update

Full Screen Zoom to Fit

Local Testing

Test your local server or HTML designs in our remote browser.

Web tunnel

or

Command line

Dashboard

Resources Sign out Get Help Your account

HOME ABOUT DATSUN OUR MODELS DATSUN NEWS FAQS GLOBAL

DAT SUN RETURNS

Discover Datsun

SPOTLIGHT

http://www.datsun.com/

The screenshot displays the BrowserStack Live interface. On the left, there's a sidebar titled 'Local Testing' with options for 'Web tunnel' and 'Command line'. The main area shows a preview of the Datsun website ('DAT SUN RETURNS') on a desktop setup with Windows 8, Internet Explorer 10.0, and a resolution of 1024x768. The preview includes a 'Discover Datsun' button and a 'SPOTLIGHT' section with images of a car's front grille and headlight. At the bottom, there's a navigation bar with icons for back, forward, search, and refresh, along with the URL 'http://www.datsun.com/'.



BrowserStack also have an automated service API for testing JavaScript code with.

It allows you to script up a number of VMs that connect to your unit tests for running automatically across a wide selection of browsers. They then pass you back the results for you to parse and check for errors.

This is good for running on a build server.



The screenshot shows a web browser displaying the "Browser API for Automated Cross-Browser Testing" page from www.browserstack.com/automated-browser-testing-api. The page title is "SonarQube™ » Discussing Cyclomatic Complexity". The main content area features a large heading "Automated JS Testing*" and a sub-headline "Instantly test JavaScript across 200+ browsers and mobile devices." Below this, a note states: "* The Automated Testing API is free for open source projects. View [pricing plans](#) for other projects." A detailed description follows, mentioning the API's capabilities and supported environments. To the left, a sidebar navigation menu includes sections for "COMMON", "LIVE TESTING", "AUTOMATE" (which is currently selected), and "SCREENSHOTS". Under "AUTOMATE", there are links for "Getting Started", "Capabilities & Timeouts", "Browsers & Platforms", "JavaScript Testing" (with sub-links for "Languages & Frameworks" including Python, Ruby, Java, C#, PHP, and Perl), and "SCREENSHOTS". At the bottom of the page, there is a section titled "Language Bindings".

Build



Finally, the third tool we use for testing is Selenium - this allows you to script interaction testing in a browser, allowing you to test behaviour at a high level.

E.g. if I click in one place, does a modal window popup somewhere else?

This is best run on a build and/or staging server.



Selenium – Web Browser Automation

docs.seleniumhq.org

SonarQube™ » Discussing Cyclomatic Complexity

Selenium – Web Browser Automation

SeleniumHQ Browser Automation

edit this page search selenium: Go Projects Download Documentation Support About

What is Selenium?

Selenium automates browsers. That's it. What you do with that power is entirely up to you. Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) also be automated as well.

Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

Which part of Selenium is appropriate for me?

	
If you want to <ul style="list-style-type: none">• create quick bug reproduction scripts• create scripts to aid in automation-aided exploratory testing	If you want to <ul style="list-style-type: none">• create robust, browser-based regression automation• scale and distribute scripts across many environments
Then you want to use Selenium IDE ; a Firefox add-on that will do simple record-and-playback of interactions with the browser	Then you want to use Selenium WebDriver ; a collection of language specific bindings to drive a browser -- the way it is meant to be driven. Selenium WebDriver is the successor of Selenium Remote Control which has been officially deprecated. The Selenium Server (used by both WebDriver and Remote Control) now also includes built-in grid capabilities.

Selenium is a suite of tools to automate web browsers across many platforms.

Selenium...

- runs in [many browsers](#) and [operating systems](#)
- can be controlled by many [programming languages](#) and [testing frameworks](#).

[Download Selenium](#)

Donate to Selenium
with Google Checkout
\$ Enter Amount
[Donate](#)

[Sponsors](#)

with PayPal

Build



So we use Grunt, BrowserStack and Selenium in addition to manual, cross-browser checks to create a quadruple-lock on our testing process, right from the local development stage all the way up to directly on the pre-release server.

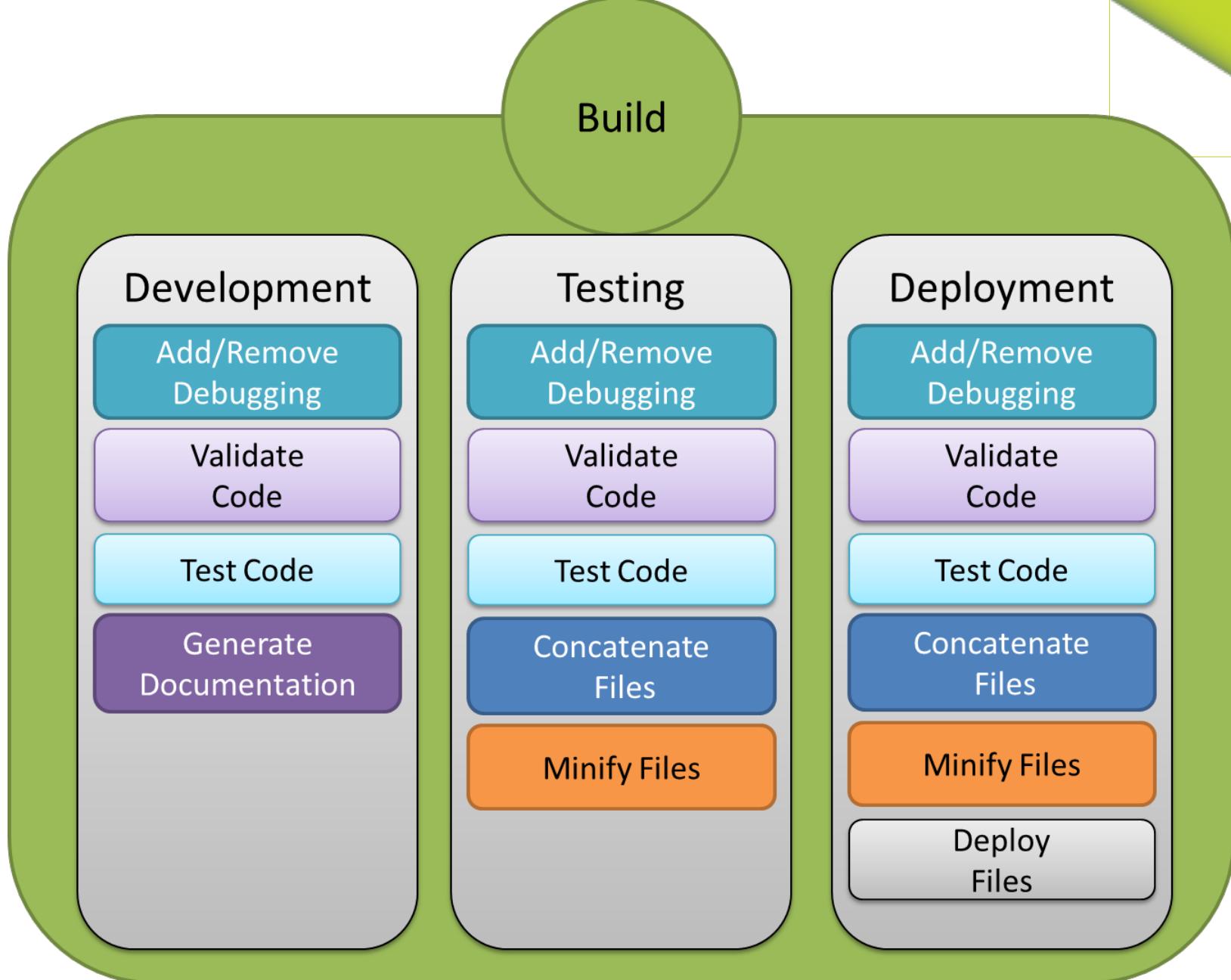


Configure Grunt To Run Static Code Analysis and Unit Tests

Run Unit Tests Cross-Browser Via BrowserStack API

Use Selenium For Automated Integration Testing

Perform Manual, Cross-Browser Testing





Recap



- Code style guidelines ensure everyone's speaking the same language
- Loose coupling of layers make changes and debugging easier
- Good programming practices allow for easier debugging
- Code organization and automation help to bring sanity to an otherwise crazy process



 **Kyle Richter**
@kylerichter

 Follow 

Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

<https://twitter.com/kylerichter/status/15101151292694529>

Thank
you!