

Welcome to Cybage



Document Name	JavaScript Style Guide
Version No.	V.1
Release Date	13 th May 2013

This document of Cybage Software Pvt. Ltd. is for restricted circulation. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means – recording, photocopying, electronic and mechanical, without prior written permission of Cybage Software Pvt. Ltd.

Document Template History

Version No.	Authored / Modified by	Reviewed by, Date	Approved by, Date	Date Remark / Change History
V0.1	Yogesh Gaikwad			Compiled as on 11 th May 2013 Contributors: Jatin Patel, Ajay Pawar
V0.2				

This document contains guidelines for web applications. This document's primary motive is to implement code consistency and best practices to all UI projects. By maintaining consistency in coding styles and conventions, we can ease the burden of code maintenance, and mitigate risk of breakage in the future.

By adhering to best practices, we ensure optimized page loading, performance and maintainable code.

Contents

1. OBJECTIVE	4
2. JAVASCRIPT FILES	4
3. INDENTATION & WHITESPACES	4
Indentation Settings for Your Tools	4
Line Length	5
4. COMMENTS	5
5. LANGUAGE RULES	6
Variable Declaration	6
Function and method names	6
Always Use Semicolons	6
Use literal expressions	7
=== and !== Operators	7
String Concatenation	7
6. JavaScript Tricks	8
True and False Boolean Expressions	8
&& and	9
Conditional (Ternary) Operator (?:)	10
7. TOP UP: PERFORMANCE OPTIMIZATION	10
Compression	10
Firebug's Timer Feature	11
Code Writing Techniques	11
8. Test Page Performance	12
ySlow for Firefox	12
PageSpeed for Chrome and Firefox	12
9. Code Debugging Tool	13
Firebug	13
Chrome DevTool	13
10. Online Code Development & Sharing Tool	13
Easy Web Development with jsFiddle	13

1. OBJECTIVE

JavaScript is the main client-side scripting language used in Cybage Front-end projects by many of UI Engineers. This is a set of coding conventions and rules for use in JavaScript programming.

These Standards are inspired by the Sun document Code Conventions for the Java Programming Language. Guidelines are heavily modified because *JavaScript is not Java*.

Style Guide can help in reducing the instability of JavaScript programs. It is all about, “How to make it more efficient” rather than “How to write JavaScript”?

2. JAVASCRIPT FILES

99% of code should be stored in and delivered as external .js files.

The primary goal is to make the page load as quickly as possible for the user. If the script file is added in the beginning of the page, the browser can't continue on until the entire file has been loaded. Thus, the user will have to wait longer before noticing any progress.

If JS files are added for some functionality like, on button click perform some action — go ahead and place those files at the bottom, just before the closing body tag. This will allow browser to render HTML objects while .js files are loading.

```
<p>Lorem Ip sum </p>
<script type="text/javascript" src="js/main.js"></script>
<script type="text/javascript" src="js/carousal.js"></script>
</body>
</html>
```

3. INDENTATION & WHITESPACES

Use an indent of 2 spaces, with no tabs. No trailing whitespace. The use of spaces can impact filesize. Difference can be eliminated by *minification*.

Indentation Settings for Your Tools

In Dreamweaver

1. Select Edit > Preferences.
2. Select Code Format from the Category list on the left.
3. Set Tab Size

In Sublime

1. Select Preferences > Settings – Default
2. It will open Preferences.sublime-settings file. Search for *tab_size*
3. Set value

In Notepad++

1. Select Settings > Preferences
2. Go to Tab Language Menu/Tab Settings
3. Update Tab Size in Tab Settings section

Line Length

Try to keep lines to 80 characters or less. Lines should not contain trailing spaces, even after binary operators, commas or semicolons. While wrapping lines, try to indent to line up with a related item on the previous line.

```
var result = prompt(aMessage,  
                    aInitialValue,  
                    aCaption);
```

4. COMMENTS

It is highly suggested to leave information regarding your code that will be read at a later time by people to understand, what is been done. The comments should be well-written, meaningful and clear to save reader's time.

Comments should be kept up-to-date.

- Single line comments

```
// Write to a heading:  
document.getElementById("pageH1").innerHTML="Welcome to my Homepage";  
// Write to a paragraph:  
document.getElementById("detailsP").innerHTML="This is my first paragraph.";
```

- Multi line comments: Multi line comments start with `/*` and end with `*/`. The following example uses a multi-line comment to explain the code:

```
/*  
 * The code below will write to a  
 * heading and to a paragraph,  
 * and will represent the start of my homepage:  
 */  
document.getElementById("pageH1").innerHTML="Welcome to my Homepage";  
document.getElementById("detailsP ").innerHTML="This is my first paragraph.";
```

**** You can take it to next level by using [JSDoc](#) rules**

5. LANGUAGE RULES

Here is the list of best practices and preferred ways of developing JavaScript code. Specific projects may follow their own standards as per the requirements.

Variable Declaration

It is preferred that each variable be given its own line and comment. They should be listed in alphabetical order. All variables should be declared with **var** before they are used and should only be declared once.

```
var areaSqFt;           // area in square feet
var countryName;       // currently selected country name
var perCapitalInc;     // per capita income of the country
```

No use of **var** will place your variable in global context. It's hard to tell in what scope a variable lives (e.g., it could be in the Document or Window just as easily as in the local scope)

Should not use alphabets as variable names like, **a, b, c, x, y, z**. It can be used in exceptional cases, in **for** loop where the scope is limited to iteration only.

Use of global variables should be minimized

Function and method names

- Functions and methods should be named in lowerCamelCase.
- Function names should begin with the name of the module or theme declaring the function to avoid collisions
- Should be logical

e.g.

popUpWindowForAd

instead of:

myWindow

Always Use Semicolons

JavaScript requires statements to end with a semicolon so all statements should be followed by **;** except for the following:

for, function, if, switch, try, while

Semicolons should be included at the end of function expressions, but not at the end of function declarations:

```
var foo = function() {  
    return true;  
}; // semicolon here.
```

```
function foo() {  
    return true;  
} // no semicolon here.
```

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`.

=== and !== Operators

It is almost always better to use the `===` and `!==` operators for faster type-based comparison. The `==` and `!=` operators do type coercion.

String Concatenation

Use `Array.prototype.join()` for string concatenation inside IE. Joining strings using the plus sign (i.e. `var ab = 'a' + 'b';`) creates performance issues in IE when used within an iteration.

6. JavaScript Tricks

True and False Boolean Expressions

The following are all **false** in boolean expressions:

- null
- undefined
- "" the empty string
- 0 the number

But be careful, because these are all **true**:

- '0' the string
- [] the empty array
- {} the empty object

This means that instead of this:

```
while (x != null) {
```

you can write this shorter code (as long as you don't expect x to be 0, or the empty string, or false):

```
while (x) {
```

And if you want to check a string to see if it is null or empty, you could do this:

```
if (y != null && y != "") {
```

But this is shorter and nicer:

```
if (y) {
```


&& and ||

These binary boolean operators are short-circuited, and evaluate to the last evaluated term.

"||" has been called the 'default' operator, because instead of writing this:

```
/** @param {*=} opt_win */  
function foo(opt_win) {  
  var win;  
  if (opt_win) {  
    win = opt_win;  
  } else {  
    win = window;  
  }  
  // ...  
}
```

you can write this:

```
/** @param {*=} opt_win */  
function foo(opt_win) {  
  var win = opt_win || window;  
  // ...  
}
```

"&&" is also useful for shortening code. For instance, instead of this:

```
if (node) {  
  if (node.kids) {  
    if (node.kids[index]) {  
      foo(node.kids[index]);  
    }  
  }  
}
```

you could do this:

```
if (node && node.kids && node.kids[index]) {  
  foo(node.kids[index]);  
}  
or this:  
var kid = node && node.kids && node.kids[index];  
if (kid) {  
  foo(kid);  
}
```

Conditional (Ternary) Operator (?:)

Use short form of value assignment on condition check:

```
var foo = (empName == "YG") ? "Access Granted" : "Access Denied";
```

Instead of:

```
var foo;  
if(empId == "YG")  
    foo = "Access Granted";  
else  
    foo = "Access Denied";
```

7. TOP UP: PERFORMANCE OPTIMIZATION

The THUMB RULE is "Don't Optimize without Measuring"

Techniques use for enhancing the performance of JavaScript to improve the responsiveness of websites and web applications.

Compression

Compression is the process of removing all unnecessary characters from source code, without changing its functionality. These unnecessary characters usually include white space characters, new line characters, comments, and sometimes block delimiters, which are used to add readability to the code but are not required for it to execute. Most of the minification tools typically reduce the size of the file by 30-90%.

Why should I Want to Compress JavaScript?

- Quicker download times for users.
- Reduced bandwidth consumption of the website.
- Reduced number of HTTP requests on server when combining many JavaScript files into one compressed file, thus reducing the server load and allowing more visitors to access the website.

Tools to be used for compression

- **WebHawk** (code validation and compression tool developed at Cybage Software Pvt. Ltd.) can be used to validate HTML/CSS, JS, LESS and Compress files. Tool is placed on *UI Server*. *Highly Recommended for UI Devs!*
- **YUI Compressor** can be used to compress files. Latest standalone package can be downloaded from <https://github.com/yui/yuicompressor/downloads>

Firebug's Timer Feature

Easy way to determine how long an operation takes? Use Firebug's "timer" feature to log the results.

```
function TimeTracker(){
  console.time("MyTimer");
  for(x=5000; x > 0; x--){}
  console.timeEnd("MyTimer");
}
```

Code Writing Techniques

Avoid eval() : While sometimes it may bring some time efficiency, it's definitely wrong practice. It makes your code look dirtier and it crashes out most of the compressors.

GET for Ajax Requests: A **POST** type request takes two TCP packets to send (headers sent first, data next). **GET** type request takes only one packet to send. So while your URL length is less than 2K and you want to request some data use **GET**.

Don't use the with() statement: with() appends an extra set of variables to the beginning of the scope chain. The basic concept is that they can be used to provide shorthand for accessing deeply nested objects.

Use:

```
var o = being.person.man.bodyparts;
o.arms = true;
o.legs = true;
```

Instead of:

```
with (being.person.man.bodyparts) {
  arms = true;
  legs = true;
}
```

De-reference Unused Objects: Do not occupy browser memory. Remove un-used objects using **null** or **delete**. The **delete** keyword should be avoided except when it is necessary to remove a property from an object's iterated list of keys, or to change the result of **if (key in obj)**.

Prefer:

```
this.foo = null;
```

Instead of:

```
delete this.property_;
```

Declare Variables Outside of the for Statement:

Use:

```
var container = document.getElementById('container');
for(var i = 0, len = someArray.length; i < len; i++) {
  container.innerHTML += 'my number: ' + i;
  console.log(i);
}
```

Instead of: (Notice how many times it will traverse the DOM to find the container element on every

iteration)

```
for(var i = 0; i < someArray.length; i++) {  
    var container = document.getElementById('container');  
    container.innerHTML += 'my number: ' + i;  
    console.log(i);  
}
```

8. Test Page Performance

ySlow for Firefox

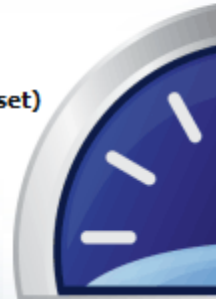
<http://developer.yahoo.com/yslow/>

Grade your web pages with YSlow

YSlow gives you:

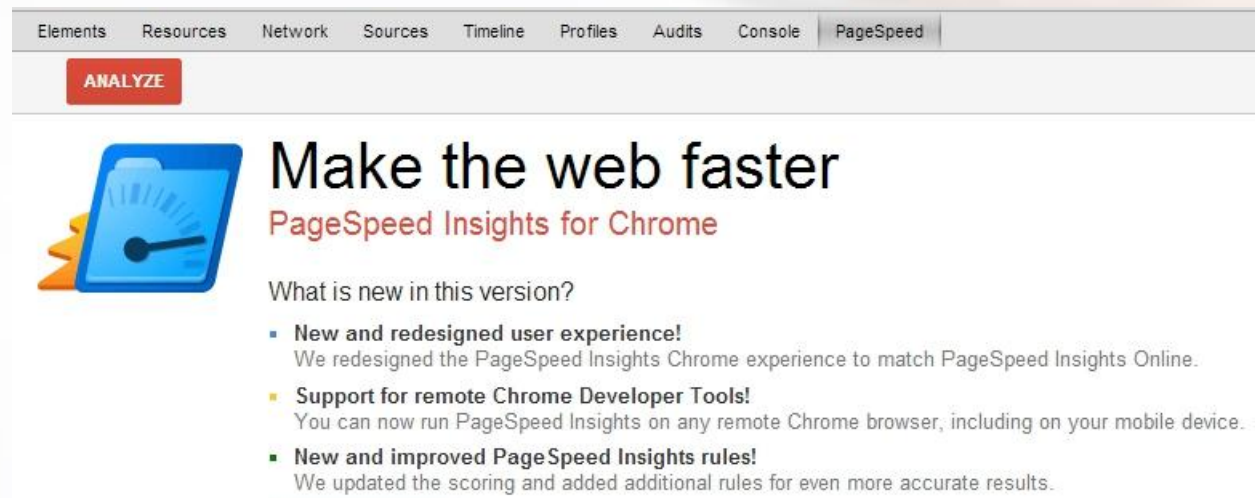
- Grade based on the performance of the page (you can define your own ruleset)
- Summary of the page components
- Chart with statistics
- Tools for analyzing performance, including Smush.it™ and JSLint

☐ Autorun YSlow each time a web page is loaded



PageSpeed for Chrome and Firefox

https://developers.google.com/speed/docs/insights/using_chrome



The screenshot shows the PageSpeed Insights for Chrome interface. At the top, there is a navigation bar with tabs: Elements, Resources, Network, Sources, Timeline, Profiles, Audits, Console, and PageSpeed. Below the navigation bar is a red button labeled "ANALYZE". The main content area features a large blue icon of a speedometer with a needle pointing to the right. To the right of the icon, the text "Make the web faster" is displayed in a large, bold, black font, followed by "PageSpeed Insights for Chrome" in a smaller, red font. Below this, the text "What is new in this version?" is followed by a list of three bullet points:

- **New and redesigned user experience!**
We redesigned the PageSpeed Insights Chrome experience to match PageSpeed Insights Online.
- **Support for remote Chrome Developer Tools!**
You can now run PageSpeed Insights on any remote Chrome browser, including on your mobile device.
- **New and improved PageSpeed Insights rules!**
We updated the scoring and added additional rules for even more accurate results.

9. Code Debugging Tool

Firebug

Developer can edit, debug, and monitor CSS, HTML, and JavaScript live in any web page.

<https://addons.mozilla.org/en-US/firefox/addon/firebug/>

Chrome DevTool

The DevTools, bundled in Chrome, provide web developers deep access into the internals of the browser and their web application. <https://developers.google.com/chrome-developer-tools/>

10. Online Code Development & Sharing Tool

Easy Web Development with jsFiddle

jsFiddle is a free code-sharing tool that allows you to edit, share, execute and debug Web code within a browser.

jsFiddle color-codes the syntax and offers a nice "tidy up" feature that reformats the code using appropriate indentation and spacing. You can also use the JSLint button to check your JavaScript syntax

<http://jsfiddle.net/>

