| | |
|---|---|
| Document name | **Mobile Web – Coding Standards and Guidelines** |
| Version no. | 1.0 |
| Release date | 15-Mar-2012 |

# Mobile Web – Coding Standards and Guidelines

## Table of Contents

# 1. Introduction

This will be a reference document for the HTML, CSS, JavaScript coding standards & best practices.

# 2. Coding Standards

## 2.1 HTML – Coding Standards

- HTML code – maintain proper indenting in the code ('Tabbing')
- There should be not too much nesting of elements and different components should be independent
    - E.g. if certain element is required to be removed, the one below it should move up
- Elements styling should not be outer-most parent related. Every module should have its corresponding parent and child elements' styles accordingly
- The HTML structure should be simple, not complex and easy to understand
- Include all script files at the end of the page, and not in <head> tag. Except for the typekit.js and the Modernizr library
- Maximum 2 CSS files and maximum 3 JS files are preferred
- Strictly NO inline CSS and JS
- No 'title' for links (anchor tag)
- h1, h2, h3, …… h6 tags should be used in the content part only and these should not be in header or the footer markup. 'span' or 'div' can be used at such places
- combine all JavaScript files to a single file and call it as global.js
- Include external CSS before external JavaScript
- Create Sprite for icon images. There should be 1 image for all icons throughout the site
- Image(s) should not be missing width and height attributes
- Do not scale images in HTML (i.e. width and height values to the <img> tag should be the actual image dimensions only)
- All styles on the page should match comps(PSDs) precisely
- Maintain proper hierarchy on headers. h1 need not be unique on a page, but should be used for important content

## 2.2 CSS – Coding Standards

- Write **clean CSS code**
- Write **multiple selectors** (IDs or classes) **on different line each**

e.g.:
```
#id1,
#id2,
#id3 {
```

```
                    styles
               }
```

- Use **space after the property, before the value**

  e.g.:

  ```
  #id {
  property:(space)value
  }
  ```

- Follow either of the standards of camelCasing or hyphen (-) separated, while writing multiple words selectors. **Do not have a mix of both in a CSS file**.
  - **Note:** The hyphen separated method is preferred; it eases readability of the code.
  - E.g.

    **.class-name {}** instead of **.className {}**

## 2.3 JavaScript coding conventions

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.

### 2.3.1 Comments

Be generous with comments. It is useful to leave information that will be read at a later time by people (possibly yourself) who will need to understand what you have done. The comments should be well-written and clear, just like the code they are annotating.

It is important that comments be kept up-to-date. Erroneous comments can make programs even harder to read and understand.
Make comments meaningful. Focus on what is not immediately visible. Don't waste the reader's time with stuff like

```
    i = 0; // Set i to zero.
```

Generally use line comments. Save block comments for formal documentation and for commenting out.

### 2.3.2 Variable Declarations

All variables should be declared before used. JavaScript does not require this, but doing so makes the program easier to read and makes it easier to detect undeclared variables that may become implied global. Implied global variables should never be used.

The var statements should be the first statements in the function body.

It is preferred that each variable be given its own line and comment.

```
var currentEntry; // currently selected table entry
var level;        // indentation level
var size;         // size of table
```

JavaScript does not have block scope, so defining variables in blocks can confuse programmers who are experienced with other C family languages. Define all variables at the top of the function.

Use of global variables should be minimized.

### 2.3.3 Function Declarations

All functions should be declared before they are used. Inner functions should follow the var statement. This helps make it clear what variables are included in its scope.

There should be no space between the name of a function and the ( (left parenthesis) of its parameter list. There should be one space between the ) (right parenthesis) and the { (left curly brace) that begins the statement body. The body itself is indented. The } (right curly brace) is aligned with the line containing the beginning of the declaration of the function.

```
function outer(c, d) {
    var e = c * d;

    function inner(a, b) {
        return (e * a) + b;
    }
    return inner(0, 1);
}
```

This convention works well with JavaScript because in JavaScript, functions and object literals can be placed anywhere that an expression is allowed. It provides the best readability with inline functions and complex structures.

```
function getElementsByClassName(className) {
    var results = [];
    walkTheDOM(document.body, function (node) {
        var a;                  // array of class names
        var c = node.className; // the node's classname
        var i;                  // loop counter
```

// If the node has a class name, then split it into a list of simple names.
// If any of them match the requested name, then append the node to the set of results.

```
            if (c) {
                a = c.split(' ');
                for (i = 0; i < a.length; i += 1) {
                    if (a[i] === className) {
                        results.push(node);
                        break;
                    }
                }
            }
        });
        return results;
  }
```

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 function, which is an incorrect reading.

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

    that = {
        method: function () {
            return this.datum;
        },
        datum: 0
    };
```

Use of global functions should be minimized.

When a function is to be invoked immediately, the entire invocation expression should be wrapped in parens so that it is clear that the value being produced is the result of the function and not the function itself.

```
var collection = (function () {
    var keys = [], values = [];

    return {
        get: function (key) {
            var at = keys.indexOf(key);
            if (at >= 0) {
                return values[at];
            }
        },
        set: function (key, value) {
            var at = keys.indexOf(key);
            if (at < 0) {
                at = keys.length;
            }
            keys[at] = key;
            values[at] = value;
        },
        remove: function (key) {
            var at = keys.indexOf(key);
            if (at >= 0) {
                keys.splice(at, 1);
                values.splice(at, 1);
            }
        }
    };
}());
```

### 2.3.4 Names

Names should be formed from the 26 upper and lower case letters (A .. Z, a .. z), the 10 digits (0 .. 9), and _ (underbar). Avoid use of international characters because they may not read well or be understood everywhere. Do not use $ (dollar sign) or \ (backslash) in names.

Do not use _ (underbar) as the first character of a name. It is sometimes used to indicate privacy, but it does not actually provide privacy.

Most variables and functions should start with a lower case letter.

Constructor functions which must be used with the new prefix should start with a capital letter.

Global variables should be in all caps.

### 2.3.5 Simple Statements

Each line should contain at most one statement. Put a ; (semicolon) at the end of every simple statement. Note that an assignment statement which is assigning a function literal or object literal is still an assignment statement and must end with a semicolon.

JavaScript allows any expression to be used as a statement. This can mask some errors, particularly in the presence of semicolon insertion. The only expressions that should be used as statements are assignments and invocations.

### 2.3.6 Compound Statements

Compound statements are statements that contain lists of statements enclosed in { } (curly braces).

The enclosed statements should be indented.
The { (left curly brace) should be at the end of the line that begins the compound statement.
The } (right curly brace) should begin a line and be indented to align with the beginning of the line containing the matching { (left curly brace).
Braces should be used around all statements, even single statements, when they are part of a control structure, such as if or for statement. This makes it easier to add statements without accidentally introducing bugs.
Ex:

```
    if (condition) {
        statements
    } else if (condition) {
        statements
    } else {
        statements
    }
```

### 2.3.7 Whitespace

Blank lines improve readability by setting off sections of code that are logically related.

Blank spaces should be used in the following circumstances:

A keyword followed by ( (left parenthesis) should be separated by a space.

```
    while (true) {
```

A blank space should not be used between a function value and its ( (left parenthesis). This helps to distinguish between keywords and function invocations.
All binary operators except . (period) and ( (left parenthesis) and [ (left bracket) should be separated from their operands by a space.
No space should separate a unary operator and its operand except when the operator is a word such as typeof.
Each ; (semicolon) in the control part of a for statement should be followed with a space.
Whitespace should follow every , (comma).

## 3. Best Practices

### 3.1 HTML - Best Practices

#### 3.1.1 Redirect mobile devices to a dedicated mobile version of your web site

There are several ways you can redirect requests to the mobile version of your web site, using server-side redirects. Most often, this is done by "sniffing" the User Agent string provided by the web browser. To determine whether to serve a mobile version of your site, you should simply look for the "mobile" string in the User Agent, which matches a wide variety of mobile devices. If necessary, you can also identify the specific operating system in the User Agent string (such as "Android 2.1").

#### 3.1.2 Always Declare a Doctype

The doctype declaration should be the first thing in your HTML documents.
The most common markup language used for mobile web sites is XHTML Basic. This standard ensures specific markup for your web site that works best on mobile devices. For instance, it does not allow HTML frames or nested tables, which perform poorly on mobile devices. Along with the DOCTYPE, be sure to declare the appropriate character encoding for the document (such as UTF-8).

For example:

```
<?xml version="1.0" encoding="UTF-8"?><!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML Basic 1.1//EN" "http://www.w3.org/TR/xhtml-basic/xhtml-basic11.dtd">
```

Also be sure that your web page markup is valid against the declared DOCTYPE. Use a validator, such as the one available at http://validator.w3.org.

#### 3.1.3 Use viewport meta data to properly resize your web page

In your document <head>, you should provide meta data that specifies how you want the browser's viewport to render your web page. For example, your viewport meta data can specify the height and width for the browser's viewport, the initial web page scale and even the target screen density.

For example:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, user-
scalable=no">
```

#### 3.1.4 Use Descriptive Meta Tags

Meta tags make your web page more meaningful for user agents like search engine spiders.

***Description Meta Attribute***

The `description` meta attribute describes the basic purpose of your web page (a summary of what the web page contains).

For example, this description:

```
<meta name="description" content="Details Description of the site for Search
Engine." />
```

### 3.1.5 Use a vertical linear layout

Avoid the need for the user to scroll left and right while navigating your web page. Scrolling up and down is easier for the user and makes your web page simpler.

### 3.1.6 Use Meaningful Title Tags

The `<title>` tag helps make a web page more meaningful and search-engine friendly. For example, the text inside the `<title>` tag appears in Google's search engine results page, as well as in the user's web browser bar and tabs.

### 3.1.7 Separate Content from Presentation

Your HTML is your content. CSS provides your content's visual presentation. Never mix both.
Don't use inline styles in your HTML. Always create a separate CSS file for your styles. This will help you and future developers that might work on your code make changes to the design quicker and make your content more easily digestible for user agents.

### 3.1.8 Minify and Unify CSS

A simple website usually has one main CSS file and possibly a few more for things like CSS reset and browser-specific fixes.
But each CSS file has to make an HTTP request, which slows down website load times.
A solution to this problem is to minify (take out unneeded characters such as spaces, newlines, and tabs) all your code and try to unify files that can be combined into one file.
Also, always put your stylesheet reference link inside the `<head></head>` tags because it will help your web page feel more responsive while loading.

### 3.1.9 Minify, Unify and Move Down JavaScript

Like CSS, never use inline JavaScript and try to minify and unify your JavaScript libraries to reduce the number of HTTP requests that need to be made in order to generate one of your web pages.

But unlike CSS, there is one really bad thing about external JavaScript files: browsers do not allow parallel downloads, which means the browser cannot download anything while it's downloading JavaScript, resulting in making the page feel like it's loading slowly.
So, the best strategy here is to load JavaScript last (i.e. after your CSS is loaded). To do this, place JavaScript at the bottom of your HTML document where possible. Best practice recommends doing this right before the closing `<body>` tag.

### 3.1.10 Close Your Tags

Closing all your tags is a W3C specification. Some browsers may still render your pages correctly (under Quirks mode), but not closing your tags is invalid under standards.

### 3.1.11 Use Lower Case Markup

It is an industry-standard practice to keep your markup lower-cased. Capitalizing your markup will work and will probably not affect how your web pages are rendered, but it does affect code readability.

### 3.1.12 Use Alt Attributes with Images

Using a meaningful `alt` attribute with `<img>` elements is a must for writing valid and semantic code.

### 3.1.13 Use Title Attributes with Links (When Needed)

Using a `title` attribute in your anchor elements will improve accessibility when used the right way. It is important to understand that the `title` attribute should be used to increase the meaning of the anchor tag.

### 3.1.14 Write Consistently Formatted Code

A cleanly written and well-indented code base shows your professionalism, as well as your consideration for the other people that might need to work on your code.

Write properly indented clean markup from the start; it will increase your work's readability.

### 3.1.15 Avoid Excessive Comments

HTML is very much self-explanatory and commenting every line of code does not make sense in HTML.

## 4. References

| | |
|---|---|
| **HTML5 compatibility table** | HTML5 compatibility table.doc |
| **Mobile Web iOS Web Applications**<br>- Configuring iOS Web Applications for native look and feel<br>- iPhone 4 Retina Display | MobileWeb_iOS Web Applications.doc |
| **Targeting Different device densities in android devices** | MobileWeb_Android Web Applications.doc |