**Front end style guidelines - draft**

Guidelines for HTML, CSS and JavaScript coding.

# General guidelines

1. Always use spaces to indent. Don't mix tabs and spaces.
2. Be consistent. Try to follow the style of the document being edited.
3. Save all files with UTF-8 encoding to avoid encoding issues.
4. Validate HTML, CSS using [w3c validators](http://www.w3.org/QA/Tools/). Check for errors in JavaScript code using [JSHin](http://www.jshint.com/about/)t.
5. Refer to Google’s [html/css](http://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml) and [JavaScript](http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml) style guides for understanding best practices.

# Tools

1. JSHint to lint JS code.
2. LESSC for coding CSS.

# HTML

1. Indent child nodes with two spaces.
2. All tags and attributes must be in lowercase.
3. Use double quotations for specifying attributes.
4. Always use HTML5 doctype.   
   <!DOCTYPE html>
5. Specify meta tag for charset,   
   <meta charset="utf-8">
6. Add all necessary stylesheets in the head of html document.
7. Elements should not have style properties assigned directly. Use classes for styling.
8. Assign meaningful CSS class names to elements based on structure and presentation.
9. Use IDs only for unique elements used as containers. Do not repeat DOM IDs for elements!
10. Plan and modularize html. Think of UI in terms of modular components that can be cloned or reused.
11. Include JavaScript towards the bottom of the page in the body of html.
12. JavaScript code should reside in external script files. Only JavaScript initialization calls can be placed in html.
13. Event handlers should not be added in html code. Use JavaScript to attach events.

<a href="#" onclick="clickHandler()">click</a> is bad.   
Use something like:

 <a class="new-tab" href="#" >click</a>   
and in JavaScript,

 $(container).delegate('click', 'a.new-tab', clickHandler);.

1. Specify 'alt' attribute for images.
2. Specify 'title' attribute for links and buttons.
3. Use [semantic HTML](http://www.adobe.com/devnet/html5/articles/semantic-markup.html). Avoid using unnecessary markup for maintaining state. Use JavaScript variables for maintaining state.

**CSS**

1. Use 2 spaces to indent code. Line should not exceed 80 characters.
2. Use semicolon after every declaration.
3. Organize CSS rules into groups based on structure and features.
4. Use meaningful class and ID names based on structure, presentation and functionality.
5. Separate words in class names, ID with hyphens.

#asset-container {}

.case-row {}

1. Do not over-qualify selectors.

#container #main { } // BAD

ul#list {} // BAD

#main { } //GOOD

#list {} //GOOD

1. Have one selector per line. Have one rule per line. Have space after property name's colon.

.profile-sidebar,

.meta-sidebar {

font-family: Arial;

color: #333;

}

1. Add the .ir class to any element you are applying image-replacement to.
2. Adding .clearfix to an element will ensure that it always fully contains its floated children.
3. Use multi or single line comments to explain about the defined styles, variables etc., if required.

Eg:

/\* Hello, I'm a CSS-style comment \*/

// Hi, I'm a silent comment, I won't show up in your CSS

**JavaScript**

1. Use 2 spaces to indent code.
2. Use curly braces on the same line as the openers.

function clickHandler() {

// function body

}

1. Always use curly braces for conditional statements.

if( isActive )

process(); **// is bad**

moreProcess();

if( isActive ) {

process(); **// Is good.**

}

moreProcess();

1. Use single quotes for strings.
2. Follow [these](http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml?showone=Code_formatting#Code_formatting) style guidelines to the spirit.
3. Declare all variables before use.
4. Declare variables keeping scope in mind. Be mindful of closures.
5. Do not corrupt global namespace with variable and function declarations. Organize and maintain variables within application namespace.

var app = {"SidebarWidget": {} };

app.SidebarWidget.status = 'open';

1. Organize code in terms of reusable widgets and modules. Avoid hardcoding DOM IDs, class names in functions; instead pass DOM element references to functions.

function slideContainer(){

$(‘#slider’).slide(); // **is bad**

// more code

}

function slideContainer(el){

$(el).slide(); **// is good**

// more code

}

slideContainer( ‘#slider’ );

1. Define all utility functions in a separate JS file. Do not add utility functions to global namespace.
2. Camel case variable names. Classes and constructors should start with capital case; other variables and functions with small case.

**JSP**

1. Pass messages and other application constants in a settings or constants JSP and assign them to objects/variables with proper namespacing.
2. Use TRIM-DIRECTIVE-WHITESPACES to avoid unnecessary whitespace in html markup generated**.**
3. **Have one layout/template for the main page. All ‘complete’ pages should extend this template.**
4. **Have one layout/template for ajax responses. All ajax responses containing HTML should use this template.**
5. **Extend templates with care to avoid repeat including DOM elements and JavaScript multiple times in pages.**