HTML, CSS Coding Guidelines

Contributors

Souvik Basu

[souvikbasu@gmail.com](mailto:souvikbasu@gmail.com)

@souvikbasu

References

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

<http://primercss.io/guidelines/>

<https://github.com/necolas/idiomatic-css>

Tools

Gulp SCSS Lint: <https://www.npmjs.com/package/gulp-scss-lint>

Sprite creation: <http://spritegen.website-performance.org/>

W3C Markup Validation Service: [https://validator.w3.org/#validate\_by\_uri+with\_options](https://validator.w3.org/" \l "validate_by_uri+with_options)

Intro

Most of the guidelines mentioned here adhere to Google Coding standards for HTML and CSS. Wherever needed we have modified the rules to suit our needs. This document contains generic CSS guideline as well as some specific SCSS guidelines.

Be consistent in styling the document. Even though many developers worked on the same piece of code, it should appear as if one person was coding the whole project. Whenever in doubt, look at how the existing code looks like and style accordingly.

HTML

Omit Protocol

Omit the protocol portion (http:, https:) from URLs pointing to images and other media files, style sheets, and scripts unless the respective files are not available over both protocols.

Omitting the protocol—which makes the URL relative—prevents mixed content issues and results in minor file size savings.

<!-- Not recommended -->

<script src="http://www.google.com/js/gweb/analytics/autotrack.js"></script>

<!-- Recommended -->

<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>

/\* Not recommended \*/

.example {

background: url(http://www.google.com/images/example);

}

/\* Recommended \*/

.example {

background: url(//www.google.com/images/example);

}

Spacing

Use space instead tabs. Indent using 4 spaces.

<ul>

<li>Fantastic

<li>Great

</ul>

Use a line limit of 120 characters.

Naming convention

Use only lowercase for both HTML and CSS.

All code has to be lowercase: This applies to HTML element names, attributes, attribute values (unless text/CDATA), CSS selectors, properties, and property values (with the exception of strings).

<A HREF="/">Home</A>

<img src="google.png" alt="Google">

color: #E5E5E5;

color: #e5e5e5;

Use hyphen for user-defined tags and attributes

tag-name-like-this

attribute-name-like-this

Prefix custom attribute with “data-“

data-my-attr

Id used should be unique across the page. Id, if present, should be the first attribute of an element followed by class (if present)

<input maxlength="20" id="email" type="email" class="login-box">

<input id="email" class="login-box" maxlength="20" type="email">

Trailing Whitespaces

Remove trailing white spaces.

Trailing white spaces are unnecessary and can complicate diffs.

Encoding

Use UTF-8 (no BOM).

Make sure your editor uses UTF-8 as character encoding, without a byte order mark.

Specify the encoding in HTML templates and documents via <meta charset="utf-8">

Comments

Avoid writing closing tag comments, like <!-- /.element -->. This just adds to page load time. Plus, most editors have indentation guides and open-close tag highlighting.

General Formatting

Paragraphs of text should always be placed in a <p> tag. Never use multiple <br> tags.

Items in list form should always be in <ul>, <ol>, or <dl>. Never use a set of <div> or <p>.

Every form input that has text attached should utilize a <label> tag. Especially radio or checkbox elements.

Even though quotes around attributes is optional, always put quotes around attributes for readability.

Avoid trailing slashes in self-closing elements. For example, <br>, <hr>, <img>, and <input>.

Don’t set tabindex manually—rely on the browser to set the order.

Many attributes don’t require a value to be set, like disabled or checked, so don’t set them.

Lean markup

Whenever possible, avoid superfluous parent elements when writing HTML. Many times this requires iteration and refactoring, but produces less HTML. For example:

<span class="avatar">

<img src="...">

</span>

<img class="avatar" src="...">

Tables

Make use of <thead>, <tfoot>, <tbody>, and <th> tags (and scope attribute) when appropriate. (Note: <tfoot> goes above <tbody> for speed reasons. You want the browser to load the footer before a table full of data.)

<table summary="This is a chart of invoices for 2011.">

<thead>

<tr>

<th scope="col">Table header 1</th>

<th scope="col">Table header 2</th>

</tr>

</thead>

<tfoot>

<tr>

<td>Table footer 1</td>

<td>Table footer 2</td>

</tr>

</tfoot>

<tbody>

<tr>

<td>Table data 1</td>

<td>Table data 2</td>

</tr>

</tbody>

</table>

Document Type

Use Html5

<!DOCTYPE html>.

IE Edge case

Use IE edge mode in meta tag

<meta http-equiv="X-UA-Compatible" content="IE=edge">

If this is note used, IE can use compatibility mode and render the page using an older version of IE such as IE7. Place this immediately after <title> tag

Semantics

Use html tag which are semantic to its purpose

<div id="header"></div>

<header></header>

Sprite

Use sprite for images instead of individual image files. This improves performance as there is only a single network call made for all images on the site. Go to <http://spritegen.website-performance.org/> to create sprite images. This site creates a single image for all image files and also provides css ruleset to apply to each image

If you are using Web Essentials, you can create a sprite bundle using Web Essentials. This is recommended as it integrates well with Visual Studio.

Multimedia formats

Preferably use multiple source types for multimedia files like video and audio. Every browser supports a different audio/video codec. e.g. WebM video format is supported in FireFox but not in Safari. So to cater to all browsers use the following:

<video controls>

<source src="somevideo.webm" type="video/webm">

<source src="somevideo.mp4" type="video/mp4">

I'm sorry; your browser doesn't support HTML5 video in WebM with VP8 or MP4 with H.264.

</video>

For more info on supported media formats: <https://developer.mozilla.org/en-US/docs/Web/HTML/Supported_media_formats>

Accessibility

Use alt tag to specify the purpose of media files (image, audio, video) so that on browsers (or screen readers) where the media does not play or takes time to render, the user can figure out the purpose of the content:

<img src="spreadsheet.png">

<img src="spreadsheet.png" alt="Spreadsheet screenshot.">

Separation of Concerns

Keep structure (HTML) separate from presentation (Styling) separate from behavior (Script).

Avoid inline styling and scripting in HTML tag.

<div style="background-color:#000" onclick="alert('I am clicked!')"></div>

<style>

.hightlight {

background-color: #000;

}

</style>

<div class="highlight"></div>

<script>

$('.highlight').click(function(){

alert('I am clicked');

});

<script>

CSS, HTML, JS can be kept in the same file if it’s a component (like Web Components). Otherwise keep them in separate files so that they can be minimized.

type attribute

Omit type attributes for style sheets and scripts.

Do not use type attributes for style sheets (unless not using CSS) and scripts (unless not using JavaScript).

Specifying type attributes in these contexts is not necessary as HTML5 implies [text/css](http://www.whatwg.org/specs/web-apps/current-work/multipage/semantics.html" \l "attr-style-type) and [text/javascript](http://www.whatwg.org/specs/web-apps/current-work/multipage/scripting-1.html" \l "attr-script-type) as defaults. This can be safely done even for older browsers.

<link rel="stylesheet" href="//www.google.com/css/maia.css" type="text/css">

<link rel="stylesheet" href="//www.google.com/css/maia.css">

<script src="//www.google.com/js/gweb/analytics/autotrack.js"

type="text/javascript"></script>

<script src="//www.google.com/js/gweb/analytics/autotrack.js"></script>

Quotation mark

When quoting attributes values, use double quotation marks.

<a class='maia-button maia-button-secondary'>Sign in</a>

<a class="maia-button maia-button-secondary">Sign in</a>

CSS

Spacing

Use space instead of tab. Use 4 spaces for indentation. Spaces are the only way to guarantee code renders the same in any person’s environment.

Put spaces after : in property declarations.

Put spaces before { in rule declarations.

Put line breaks between rulesets.

When grouping selectors, keep individual selectors to a single line.

Place closing braces of declaration blocks on a new line.

.add-section {

margin: 20px;

}

input:focus, textarea:focus, select:focus{

border-top-width: 2px; border-bottom-width:2px; }

.add-section {

margin: 20px;

}

input:focus,

textarea:focus,

select:focus {

border-top-width: 2px;

border-bottom-width: 2px;

}

Comments

Use // for comment blocks (instead of /\* \*/).

/\* Do not comment like this \*/

.add-section {

margin: 20px;

}

// Some comment here

.add-section {

margin: 20px;

}

Zero Values

Avoid specifying units for zero values

.add-section {

margin: 0px;

}

.add-section {

margin: 0;

}

Omit leading “0”s in values.

Do not use put 0s in front of values or lengths between -1 and 1.

font-size: 0.8em;

font-size: .8em;

Short hands

Use short hands in places where you need to specify all the values explicitly

.add-section {

margin-top: 10px;

margin-right: 10px;

margin-bottom: 10px;

margin-left: 10px;

border-width: 10px;

border-color: #000;

border-style: solid;

}

.add-section {

margin: 10px 20px 10px 30px;

border: 20px #000 solid;

}

Avoid Short hands if you do not have explicit values for all available values.

Specificity

Use classes instead of id as selector in css

#the\_specific\_section {

margin: 10px 20px 10px 30px;

border: 20px #000 solid;

}

.section {

margin: 10px 20px 10px 30px;

border: 20px #000 solid;

}

Use id only if the style is to be applied to a single element. If you have id in css, make sure not to have more than one # in a selector.

#header .search #quicksearch { ... }

If using scss-lint to validate the file, use disabled comment to ignore rule to not use id in css. Remember to enable it after the block. IdSelector checks for no usage of id in css file, SelectorFormat checks for hyphenated selectors by default which is needed in class names but not id.

// scss-lint:disable IdSelector SelectorFormat

#template\_admin\_pages {

height: 100%;

width: 100%;

}

// scss-lint:enable IdSelector SelectorFormat

The class names disabled, mousedown, danger, hover, selected, and active should always be namespaced by a class.

disabled { ... }

button.disabled { ... }

Avoid qualifying ID and class names with type selectors.

Unless necessary (for example with helper classes), do not use element names in conjunction with IDs or classes.

Avoiding unnecessary ancestor selectors is useful for [performance reasons](http://www.stevesouders.com/blog/2009/06/18/simplifying-css-selectors/).

ul#example {}

div.error {}

#example {}

.error {}

Naming convention

Use ID and class names that are as short as possible but as long as necessary.

E.g. #nav not #navigation, .author not .atr

Do not concatenate words and abbreviations in selectors by any characters (including none at all) other than hyphens, in order to improve understanding and scannability.

E.g. .demo-image not .demoimage or .demo\_image

class-names-like-this

ids\_like\_this

mixin-names-like-this

Use class names for what the element is for and not how it looks like

.red-box {

color: #f00;

}

.error {

color: #f00;

}

Units

Unit-less line-height is preferred because it does not inherit a percentage value of its parent element, but instead is based on a multiplier of the font-size.

Avoid use of em. Use rem instead with a fall back of pixel unit

font-size: 32px;

font-size: 2rem;

Declaration order

Put declarations in alphabetical order in order to achieve consistent code in a way that is easy to remember and maintain.

Ignore vendor-specific prefixes for sorting purposes. However, multiple vendor-specific prefixes for a certain CSS property should be kept sorted (e.g. -moz prefix comes before -webkit).

background: fuchsia;

border: 1px solid;

-moz-border-radius: 4px;

-webkit-border-radius: 4px;

border-radius: 4px;

color: black;

text-align: center;

text-indent: 2em;

Semicolon

Use semicolon after every declaration

.test {

display: block;

height: 100px

}

.test {

display: block;

height: 100px;

}

Quotation marks

Use single ('') rather than double ("") quotation marks for attribute selectors or property values. Do not use quotation marks in URI values (url()).

@import url("//www.google.com/css/maia.css");

html {

font-family: "open sans", arial, sans-serif;

}

@import url(//www.google.com/css/maia.css);

html {

font-family: 'open sans', arial, sans-serif;

}

SCSS

Selector Hierarchy

Use selector hierarchy instead of flat structure. This created better readability and is more maintainable

.ui-widget-content {

background: $secondary-theme-color-2;

color: $primary-theme-text-on-color-2;

}

.ui-widget-content a {

color: $primary-theme-text-on-color-2;

}

.ui-widget-content {

background: $secondary-theme-color-2;

color: $primary-theme-text-on-color-2;

a {

color: $primary-theme-text-on-color-2;

}

}

Avoid nesting more than 4 levels. Also break a block into multiple blocks it exceeds more than 50 lines. A long block that does not fit the screen height can nullify the understandability advantage that nesting of rulesets brings in.

Color

Use hex color codes #000 instead of color names. SCSS’ rgba() function is overloaded to accept hex colors as a param, e.g., rgba(#000, .5)

.my-class {

color: orange;

}

.my-class {

color: #ffa500;

}

Use 3 character hexadecimal notation where possible.

For color values that permit it, 3 character hexadecimal notation is shorter and more succinct.

color: #eebbcc;

color: #ebc;

Use SCSS variables

Using variables enables consistent styling and more maintainable code.

It also allows easy switching of themes.

Use a separate file to store variables and import it in all scss files as needed.

\_theme.scss:

$font-stack: "Segoe UI", Helvetica;

$light-font-weight: 300;

$semi-light-font-weight: 300;

$normal-font-weight: 400;

$heavy-font-weight: 500;

// primary theme

$primary-theme-color-1: #0082c8;

$primary-theme-color-2: #fff;

$primary-theme-text-on-color-1: #fff;

$primary-theme-text-on-color-2: #222;

...

neonlite.scss:

@import 'theme';

button {

border: solid 1px $primary-theme-border-color;

color: $primary-theme-color-1;

}

Add all theme related properties as variables in \_theme.scss file

font

color

background-color

Mixins

Use mixins for commonly used styles like font size

@mixin fontSize($size) {

font-size: $size; //Fallback in px

font-size: #{$size/16px}rem;

}

.page-title {

font-weight: $light-font-weight;

@include fontSize(30px);

}

any time you'd use a mixin with no parameter, an extend will be more efficient

.foo {

color: #f00;

}

.bar {

@extend .foo;

}

This outputs to following css:

.foo, .bar {

color: #f00;

}

@mixin stuff {

color: #f00;

}

.foo {

@include stuff;

}

.bar {

@include stuff;

}

This will get those styles into both selectors. This is often easier to understand and has less "gotchas," so is more common to see. Note that the output moves the styles into both places:

.foo {

color: red;

}

.bar {

color: red;

}

Indentation

Avoid indentation of more than 4 levels

.class-1 {

&.class-2 {

&.class-3 {

}

}

}

.class-1 {

&.class-2 {

&.class-3 {

&.class-4 {

&.class-5 {

...

}

}

}

}

}

Vendor specific properties

Do not manually add vendor specific settings in SCSS. Instead set Autoprefixer to true in Web Essentials to auto create vendor specific settings

\* {

-moz-box-sizing: border-box;

-webkit-box-sizing: border-box;

box-sizing: border-box;

}

\* {

box-sizing: border-box;

}

Validation tool

Use validation tools to make sure the CSS syntax and best practices are used. In our projects we will use gulp-scss-lint to validate SCSS file. For validating css files, you can use <http://jigsaw.w3.org/css-validator/>. See Appendix for details.

Shame css

Create a shame.scss to keep all styling hacks that have been added due to time constrained but which the developer decides to come back at a later point to fix it.

e.g. If you cannot figure out why the button background is not changing using color property, you can use a !important attribute for the style but keep it in shame.css so that you can come back to it at a later point and put a clean fix.

neonlite.scss:

button {

background-color: #fff;

}

shame.scss:

// Provide enough comment on why you did this

// and what is a cleaner fix if you know

.disabled-button {

background-color: #0e0e0e; !important

}

At a later point you come back and remove the setting from shame.scss and include a clean fix in neonlite.scss

button {

background-color: #fff;

&:disabled {

background-color: #0e0e0e;

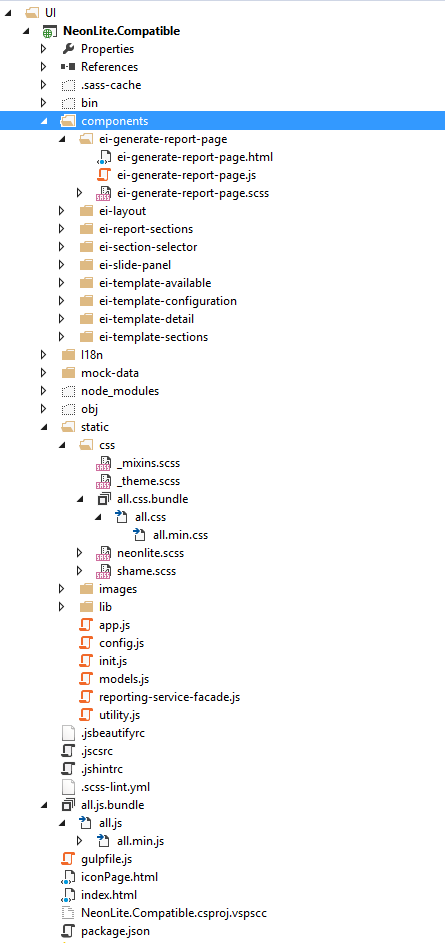
}

}

Appendix

File structure

We use the following file structure for our backbone app. Notice creation of UI components under the components folder. Each component folder has a html, js, scss files thus creating a separation of concern yet encapsulation.



Config file for JSBeautify

Create .jsbeautifyrc at project level folder as follows. This can be used to beautify JS, HTML and CSS files.

{

"braceStyle": "collapse",

"breakChainedMethods": false,

"e4x": false,

"evalCode": false,

"indentChar": " ",

"indentLevel": 0,

"indentSize": 4,

"indentWithTabs": false,

"jslintHappy": false,

"keepArrayIndentation": false,

"keepFunctionIndentation": false,

"maxPreserveNewlines": 3,

"preserveNewlines": true,

"spaceBeforeConditional": true,

"spaceInParen": false,

"unescapeStrings": false,

"wrapLineLength": 120

}

Config node package

Create package.json at project level folder as follows. Add/delete modules as needed by the project

{

"name": "mean-seed",

"version": "0.0.1",

"description": "Seed project for MEAN stack",

"keywords": [

"MeanHub",

"Mean-seed",

"MongoDB",

"ExpressJS",

"AngularJS",

"Node.js"

],

"author": {

"name": "MeanHub",

"email": "info@meanhub.com",

"url": "http://meanhub.com/"

},

"homepage": "http://meanhub.com/",

"scripts": {

"start": "node main.js"

},

"main": "main.js",

"dependencies": {

"body-parser": "~1.13.1",

"cookie-parser": "~1.3.5",

"debug": "~2.2.0",

"ejs": "~2.3.2",

"express": "~4.13.0",

"morgan": "~1.6.1",

"serve-favicon": "~2.3.0"

},

"engines": {

"node": ">= 0.6.0",

"npm": ">= 1.0.0"

},

"devDependencies": {

"browser-sync": "^2.8.2",

"gulp": "^3.9.0",

"gulp-autoprefixer": "^2.3.1",

"gulp-concat": "^2.6.0",

"gulp-if": "^1.2.5",

"gulp-jscs": "^2.0.0",

"gulp-jshint": "^1.11.2",

"gulp-minify-css": "^1.2.0",

"gulp-nodemon": "^2.0.3",

"gulp-notify": "^2.2.0",

"gulp-notify-growl": "^1.0.2",

"gulp-processhtml": "^1.1.0",

"gulp-rename": "^1.2.2",

"gulp-sass": "^2.0.4",

"gulp-uglify": "^1.2.0",

"rimraf": "^2.4.2",

"run-sequence": "^1.1.2",

"yargs": "^3.18.0"

}

}

Run npm install to install the packages mentioned in the package.json file.

Config gulp file

Create gulpfile.js at project level folder as follows. Modify file paths as per project structure

var gulp = require('gulp');

var prettify = require('gulp-jsbeautifier');

var scsslint = require('gulp-scss-lint');

var jsFiles = [

'your-js-files.js'

];

var htmlFiles = [

'index.html',

'components/\*\*/\*.html'

];

var scssFiles = [

'static/\*\*/\*.scss',

'components/\*\*/\*.scss'

];

gulp.task('format-js', function () {

gulp.src(jsFiles, { base: './' })

.pipe(prettify({ config: '.jsbeautifyrc', mode: 'VERIFY\_AND\_WRITE' }))

.pipe(gulp.dest('./'));

});

gulp.task('prettify-html', function () {

gulp.src(htmlFiles, { base: './' })

.pipe(prettify({ indentSize: 4 }))

.pipe(gulp.dest('./'));

});

gulp.task('scss-lint', function () {

gulp.src(scssFiles)

.pipe(scsslint());

});

Run gulp scss-lint to validate SCSS file.

To run gulp tasks from Visual Studio, install VS extension at <https://visualstudiogallery.msdn.microsoft.com/8e1b4368-4afb-467a-bc13-9650572db708>

Config SCSS lint file

Create config file .scss-lint.yml at project root folder

# Default application configuration that all configurations inherit from.

plugin\_directories: ['.scss-linters']

# List of gem names to load custom linters from (make sure they are already

# installed)

plugin\_gems: []

linters:

BangFormat:

enabled: true

space\_before\_bang: true

space\_after\_bang: false

BemDepth:

enabled: false

max\_elements: 1

BorderZero:

enabled: true

convention: zero # or `none`

ColorKeyword:

enabled: true

ColorVariable:

enabled: true

Comment:

enabled: true

DebugStatement:

enabled: true

DeclarationOrder:

enabled: true

DuplicateProperty:

enabled: true

ElsePlacement:

enabled: true

style: same\_line # or 'new\_line'

EmptyLineBetweenBlocks:

enabled: true

ignore\_single\_line\_blocks: true

EmptyRule:

enabled: true

ExtendDirective:

enabled: false

FinalNewline:

enabled: true

present: true

HexLength:

enabled: true

style: short # or 'long'

HexNotation:

enabled: true

style: lowercase # or 'uppercase'

HexValidation:

enabled: true

IdSelector:

enabled: true

ImportantRule:

enabled: true

ImportPath:

enabled: true

leading\_underscore: false

filename\_extension: false

Indentation:

enabled: true

allow\_non\_nested\_indentation: false

character: space # or 'tab'

width: 4

LeadingZero:

enabled: true

style: exclude\_zero # or 'include\_zero'

MergeableSelector:

enabled: true

force\_nesting: true

NameFormat:

enabled: true

allow\_leading\_underscore: true

convention: hyphenated\_lowercase # or 'camel\_case', or 'snake\_case', or a regex pattern

NestingDepth:

enabled: true

max\_depth: 4

ignore\_parent\_selectors: false

PlaceholderInExtend:

enabled: true

PropertyCount:

enabled: false

include\_nested: false

max\_properties: 10

PropertyUnits:

enabled: true

global: [

'ch', 'em', 'ex', 'rem', # Font-relative lengths

'cm', 'in', 'mm', 'pc', 'pt', 'px', 'q', # Absolute lengths

'vh', 'vw', 'vmin', 'vmax', # Viewport-percentage lengths

'deg', 'grad', 'rad', 'turn', # Angle

'ms', 's', # Duration

'Hz', 'kHz', # Frequency

'dpi', 'dpcm', 'dppx', # Resolution

'%'] # Other

properties: {}

PropertySortOrder:

enabled: true

ignore\_unspecified: false

min\_properties: 2

separate\_groups: false

PropertySpelling:

enabled: true

extra\_properties: []

QualifyingElement:

enabled: true

allow\_element\_with\_attribute: true

allow\_element\_with\_class: false

allow\_element\_with\_id: false

SelectorDepth:

enabled: true

max\_depth: 4

SelectorFormat:

enabled: true

convention: hyphenated\_lowercase # or 'strict\_BEM', or 'hyphenated\_BEM', or 'snake\_case', or 'camel\_case', or a regex pattern

Shorthand:

enabled: true

allowed\_shorthands: [1, 2, 3]

SingleLinePerProperty:

enabled: true

allow\_single\_line\_rule\_sets: true

SingleLinePerSelector:

enabled: true

SpaceAfterComma:

enabled: true

SpaceAfterPropertyColon:

enabled: true

style: one\_space # or 'no\_space', or 'at\_least\_one\_space', or 'aligned'

SpaceAfterPropertyName:

enabled: true

SpaceBeforeBrace:

enabled: true

style: space # or 'new\_line'

allow\_single\_line\_padding: false

SpaceBetweenParens:

enabled: true

spaces: 0

StringQuotes:

enabled: true

style: single\_quotes # or double\_quotes

TrailingSemicolon:

enabled: true

TrailingWhitespace:

enabled: true

TrailingZero:

enabled: false

UnnecessaryMantissa:

enabled: true

UnnecessaryParentReference:

enabled: true

UrlFormat:

enabled: true

UrlQuotes:

enabled: false

VariableForProperty:

enabled: false

properties: []

VendorPrefix:

enabled: true

identifier\_list: base

additional\_identifiers: []

excluded\_identifiers: []

ZeroUnit:

enabled: true

Compass::\*:

enabled: false

Web Essentials settings

Install Web Essentials for Visual Studio.

In Visual Studio go to Tools > Options > Web Essentials > CSS

Set Enable Autoprefixer to True

Set Disallow Universal selector to False as this was a problem with IE6 and most modern browsers have negligible performance hit.

