GUIDELINES: HTML + CSS + FILES & FOLDERS + JAVASCRIPT

V03042013

Contents:

1. GENERAL RULES FOR ALL CODE:

1. *Source: http://bit.ly/Y6pOG2 (digitalpulp / developer-handbook / text file conventions / code formatting)*
2. EVERYTHING should be in lowercase.
    a. Use text-transform in CSS to achieve all lowercase/uppercase if needed.

```
USE THIS STYLE OF COMMENTING THROUGHOUT YOUR FILE:
// Single line comments <-- NOTE: No ending.

USE THIS STYLE OF COMMENTING AT THE TOP OF YOUR FILE ONLY:
/* Multi-line comments */
```

3. 2 space indentation only.
4. New line at end of file, always
    a. For when casting to console
    b. Also "finishes" file ("punctuation"-wise)
5. Avoid trailing whitespace
6. Apply blank lines liberally
    a. Group code into paragraphs/verses
7. You should be aware of, but you don't need to strictly follow 78/80 char line length.
    a. In other words, be considerate of line lengths, but don't let them rule your world.
8. Constantly look for and remove UNUSED or COMMENTED OUT code that is no longer needed.
9. Commenting in CSS and Javascript:

2. FOLDER STRUCTURE & FILE NAMES

PRDXN_GUIDELINES: FOLDER STRUCTURE & FILE NAMES: http://bit.ly/VZgTtI
AS/4/23: For wordpress, the css file needs to be in root directory of theme folder

3. HTML:

Audit your PLAN and CODE against the following guidelines FREQUENTLY throughout development.

1. Use PRDXN's standard HTML template. As found here:

https://github.com/prdxn/prdxn-dev-framework

   a. *(For login assistance) PRDXN_FRAMEWORK: http://bit.ly/XBo1c0*
2. See GENERAL RULES FOR ALL CODE above.
3. Will the next person using your code, be able to **EASILY UNDERSTAND** what you've done, and WHY? Is it **STRUCTURED** in a **LOGICAL** fashion in comparison to what can be seen in the psd (or browser)?
4. **Can it EASILY "GROW"?**
   a. **Will it break (will the front-end get messed up) if an additional x (Title, Paragraph, Image, Page) is added?**
      i. If an additional Title, Paragraph, Image, Page, etc. is needed - will the code easily adapt/allow for that, or will the visual/front-end break? If it will break, you have not structured it correctly - it MUST be structured for growth.
   b. **Will it break (will the front-end get messed up) if x (Title, Paragraph, Image, etc.) grows beyond the current psd indication of x chars or x height/width?**
      i. If the Title, Paragraph, Image, etc. needs to grow beyond what's indicated in the current psd in terms of chars or height/width - will the code easily adapt/allow for that or will the visual/front-end break? If it will break, you have not structured it correctly - it MUST be structured for growth.
   c. **Can it be easily CMS enabled - if required in the future?**
      i. If the Client or PRDXN decides to make the site CMS-enabled, how much "re-work" of the HTML will be required to do this? More rework = NOT good.
5. PRDXN_IMAGES & IMAGE NAMING CONVENTIONS: http://bit.ly/VZcTcw
6. Your HTML mark-up MUST represent the ACTUAL content that is being seen - readability-related (those with eyesight restrictions, using screen readers, will "read" (hear) "hidden" code). Implications:
   a. You should never hide HTML. If needed be, append it via Javascript.
   b. Use more than image path/source, to avoid hiding images with media queries.
7. ***(Also mentioned in: DP Developer Handbook, Folder Structure & File Names:)***
   a. All unknown/TBD URLs should be FIXME (as in href="FIXME"). DO NOT USE href="#"; it's ambiguous as to the intended functionality.
   b. Keep all pages in the same folder (the docroot); do not put pages into subfolders.
   c. *(Specifically for Digital Pulp projects, or projects that are to-be-CMS-enabled by a 3rd party-NOT PRDXN)* Do not link pages together. Though it is helpful, it creates a maintenance nightmare for static sites.
8. **HTML Antipatterns:**
   a. *From: DP Developer Handbook, Source: http://bit.ly/YOXVVu*
   b. **Don't use the javascript: psuedo-protocol:**
      i. *javascript: should not be in either the href or onclick attributes. see http://stackoverflow.com/questions/10068781/ for discission.*
   c. **Avoid Inline JavaScript:**
      i. *bind event handlers via script blocks or external .js files instead.*
   d. (DP Developer Handbook/HTML Anitpatterns) **Avoid all-upper/lowercase content in HTML:**
      i. *Avoid using all lowercase or all uppercase in HTML files. case is usually used for design effect. instead use text-transform in CSS to achieve all*

lowercase/uppercase. The exception, of course may be legal copy (usually provided in all uppercase).

9. **Does it PASS by Google's HTML Style and Formatting Rules?** http://google-styleguide.googlecode.com/svn/trunk/htmlcssguide.xml#HTML_Style_Rules
    a. Focus on:
        i. HTML STYLE RULES
        ii. HTML FORMATTING RULES.
10. **Make your HTML, SEO Friendly from the START:**
    a. PRDXN_SEO GUIDELINES: http://bit.ly/ZT8Y4E
        i. *Follow steps 1 to 6 in relation to your HTML.*
11. Remove:
    a. *unused code; commented-out code; hard-coded links; references to non-relevant sites.*
12. Perform (frequently) HTML Validation: http://validator.w3.org/

4. CSS:

1. Use PRDXN's standard CSS template. As found here: https://github.com/prdxn/prdxn-dev-framework
    a. (For login assistance) PRDXN_FRAMEWORK: http://bit.ly/XBo1c0
2. Audit your PLAN and CODE against the below - "Proposed CSS Style Guide" - aka "CSS GUIDELINES" FREQUENTLY throughout development. (Also audit against GENERAL RULES FOR ALL CODE above.)

"Proposed CSS STYLE GUIDE" v1.1:
Source - GitHub URL: http://bit.ly/14NqIxn
Source - G'docs URL: http://bit.ly/Vd6V7o

Proposed CSS Style Guide

Revision 1.1
This document defines formatting and style rules for CSS. It aims at improving collaboration and code quality of Digital Pulps internal and external development teams.
Based on the above inspiration. All parts attributable to Google's CSS Rules. Exceptions noted. (Some examples abbreviated)

1. Indent by 2 spaces at a time.

Don't use tabs or mix tabs and spaces for indentation.

```
.example {
  color: blue;
}
```

2. Indent all block content.

Indent all block content, that is rules within rules as well as declarations, so to reflect hierarchy and improve understanding.

```
@media screen, projection {
 html {
  background: #fff;
  color: #444;
 }
}
```

3. Use only lowercase.

(Edited to be specific to CSS)
All code has to be lowercase: This applies to selectors, properties, and property values.

```
.the-example h2 {
 color: #333;
}
```

4. Use valid CSS. (Edited to use absolute language)

Validate your CSS with the W3C CSS validator: http://jigsaw.w3.org/css-validator/. Vendor specific prefixes for new CSS properties are the only acceptable exception to this rule.

5. Use meaningful or generic ID and class names.

Always use ID and class names that reflect the purpose of the element in question, or that are otherwise generic.
Names that are specific and reflect the purpose of the element should be preferred as these are most understandable and the least likely to change.
Generic names are simply a fallback for elements that have no particular or no meaning different from their siblings. They are typically needed as "helpers."
Using functional or generic names reduces the probability of unnecessary document or template changes.

```
/* Recommended: specific */
#gallery {}
#login {}
.video {}

/* Recommended: generic */
.aux {}
.alt {}
```

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

Try to convey what an ID or class is about while being as brief as possible. Using ID and class names this way contributes to acceptable levels of understandability and code efficiency.

```
/* Not recommended */
#navigation {}
.atr {}

/* Recommended */
```

```
#nav {}
.author {}
```

7. Don't qualify ID and class names with type selectors.

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

```
/* Not recommended */
ul#example {}
div.error {}

/* Recommended */
#example {}
.error {}
```

8. Don't use IDs to apply style. Do use IDs for javascript and jQuery.

(This is from my experience with supporting articles. Absolute language used for effect)
In CSS IDs offer lowered performance and increased issues with specificity. See Don’t use IDs in CSS selectors?
jQuery uses the native getElementByID() for accessing DOM elements. This is the fastest method for accessing DOM elements. See 10 Ways to Instantly Increase Your jQuery Performance.

9. Use shorthand properties where possible.

CSS offers a variety of shorthand properties (like font) that should be used whenever possible, even in cases where only one value is explicitly set.

```
/* Not recommended */
border-top-style: none;
font-family: palatino, georgia, serif;
font-size: 100%;
line-height: 1.6;
padding-bottom: 2em;
padding-left: 1em;
padding-right: 1em;
padding-top: 0;

/* Recommended */
border-top: 0;
font: 100%/1.6 palatino, georgia, serif;
padding: 0 1em 2em;
```

10. Omit unit specification after "0" values.

Do not use units after 0 values unless they are required.

```
margin: 0;
padding: 0;
```

11. Omit leading "0"s in values.

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

```
font-size: .8em;
```

12. Use 3 character hexadecimal notation where possible.

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

```
/* Not recommended */
color: #eebbcc;

/* Recommended */
color: #ebc;
```

13. Prefix selectors with an application-specific prefix.

(optional)

In large projects as well as for code that gets embedded in other projects or on external sites use prefixes (as namespaces) for ID and class names. Use short, unique identifiers followed by a dash. Using namespaces helps preventing naming conflicts and can make maintenance easier, for example in search and replace operations.

```
.adw-help {} /* AdWords */
#maia-note {} /* Maia */
```

14. Separate words in ID and class names by a hyphen.

(Altered example to be more literal)

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.

```
/* Not */
.demoimage {}
.error_status {}

/* Recommended */
.demo-image {}
.error-status {}
```

15. Avoid user agent detection as well as CSS "hacks"-- try a different approach first.

It's tempting to address styling differences over user agent detection or special CSS filters, workarounds, and hacks. Both approaches should be considered last resort in order to achieve and maintain an efficient and manageable code base. Put another way, giving detection and hacks a free pass will hurt projects in the long run as projects tend to take the way of least resistance. That is, allowing and making it easy to use detection and hacks means using detection and hacks more frequently--and more frequently is too frequently.

16. Order Vendor Prefixes

(Reference Wordpress Vendor Prefixes)
Vendor prefixes should go longest (-webkit-) to shortest (unprefixed). Values should be right aligned with spaces after the colon provided that all the values are the same across all prefixes.

```
/* Preferred method: */
.koop-shiny {
 -webkit-box-shadow: inset 0 0 1px 1px #eee;
 -moz-box-shadow:    inset 0 0 1px 1px #eee;
 box-shadow:         inset 0 0 1px 1px #eee;

 -webkit-transition: border-color 0.1s;
 -moz-transition:    border-color 0.1s;
 -ms-transition:     border-color 0.1s;
 -o-transition:      border-color 0.1s;
 transition:         border-color 0.1s;
}

/* Special case for CSS gradients: */
.gradient {
 background: #fff;
 background-image: -webkit-gradient(linear, left bottom, left top, from(#fff), to(#000));
 background-image: -webkit-linear-gradient(bottom, #fff, #000);
 background-image:    -moz-linear-gradient(bottom, #fff, #000);
 background-image:      -o-linear-gradient(bottom, #fff, #000);
 background-image: linear-gradient(to top, #fff, #000);
}
```

17. Use a semicolon after every declaration.

End every declaration with a semicolon for consistency and extensibility reasons.

```
/* Not recommended */
.test {
 display: block;
 height: 100px
}

/* Recommended */
.test {
 display: block;
 height: 100px;
}
```

18. Use a space after a property name's colon.

Always use a single space between property and value (but no space between property and colon) for consistency reasons.

```
/* Not recommended */
h3 {
 font-weight:bold;
```

```
}

/* Recommended */
h3 {
  font-weight: bold;
}
```

19. Separate selectors and declarations by new lines.

Always start a new line for each selector and declaration.

```
/* Not recommended */
a:focus, a:active {
  position: relative; top: 1px;
}

/* Recommended */
h1,
h2,
h3 {
  font-weight: normal;
  line-height: 1.2;
}
```

20. Separate rules by new lines.

Always put a line between rules.

```
html {
  background: #fff;
}

body {
  margin: auto;
  width: 50%;
}
```

21. Use single quotation marks for attribute selectors and property values.

Use single (*) rather than double ("") quotation marks for attribute selectors or property values. Do not use quotation marks in URI values (url()). *
Exception: If you do need to use the @charset rule, use double quotation marks--single quotation marks are not permitted.

```
/* Not recommended */
@import url("//www.google.com/css/maia.css");
html {
  font-family: "open sans", arial, sans-serif;
}

/* Recommended */
@import url(//www.google.com/css/maia.css);
html {
```

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

22. Group sections by a section comment.

(optional)
If possible, group style sheet sections together by using comments. Separate sections with new lines.

```
/* Header */

#adw-header {}

/* Footer */

#adw-footer {}

/* Gallery */

.adw-gallery {}
```

23. Declaration order matters.

(Reference Idiomatic; revised to my preferred order)
Declarations are to be consistently ordered; box-model, display, positioning and etc.

```
.selector {
  /* Box Model */
  box-sizing: border-box;
  width: 100px;
  height: 100px;
  padding: 10px;
  border: 10px solid #333;
  margin: 10px;

  /* Display */
  display: inline-block;
  overflow: hidden;

  /* Positioning */
  position: absolute;
  z-index: 10;
  top: 0;
  right: 0;
  bottom: 0;
  left: 0;

  /* Other */
  background: #000;
  color: #fff;
  font-family: sans-serif;
  font-size: 16px;
  text-align: right;
}
```

24. Exceptions and slight deviations

(Reference Idiomatic)

Large blocks of single declarations can use a slightly different, single-line format. In this case, a space should be included after the opening brace and before the closing brace.

```
.selector-1 { width: 10%; }
.selector-2 { width: 20%; }
.selector-3 { width: 30%; }
```

25. Comment your code

(Reference Idiomatic

Well commented code is extremely important. Take time to describe components, how they work, their limitations, and the way they are constructed. Don't leave others in the team guessing as to the purpose of uncommon or non-obvious code.

- Place comments on a new line above their subject.
- Keep line-length to a sensible maximum, e.g., 80 columns.
- Make liberal use of comments to break CSS code into discrete sections.
- Use "sentence case" comments and consistent text indentation.

```
/* ============================================================================
  Section comment block
  ============================================================================ */

/* Sub-section comment block
  ============================================================================ */

/**
 * Short description using Doxygen-style comment format
 *
 * The first sentence of the long description starts here and continues on this
 * line for a while finally concluding here at the end of this paragraph.
 *
 * The long description is ideal for more detailed explanations and
 * documentation. It can include example HTML, URLs, or any other information
 * that is deemed necessary or useful.
 *
 * @tag This is a tag named 'tag'
 *
 * @todo This is a todo statement that describes an atomic task to be completed
 *   at a later date. It wraps after 80 characters and following lines are
 *   indented by 2 spaces.
 */

/* Basic comment */
```

26. Define a table of contents

(Reference Smashing)

To keep an overview of the structure of your code, you might want to consider defining a table of contents in the beginning of your CSS-files. One possibility of integrating a table of contents is to display a tree overview of your layout with IDs and classes used in each branch of the tree. You may want to use some keywords such as header-section or content-group to be able to jump to specific code immediately.
You may also select some important elements you are likely to change frequently — after the project is released. These classes and IDs may also appear in your table of contents, so once you'll need to find them you'll find them immediately — without scanning your whole code or remembering what class or ID you once used.

```
/*-----------------------------------------------------------------
 [Table of contents]

 1. Body
   2. Header / #header
     2.1. Navigation / #navbar
   3. Content / #content
     3.1. Left column / #leftcolumn
     3.2. Right column / #rightcolumn
     3.3. Sidebar / #sidebar
       3.3.1. RSS / #rss
       3.3.2. Search / #search
       3.3.3. Boxes / .box
       3.3.4. Sideblog / #sideblog
       3.3.5. Advertisements / .ads
   4. Footer / #footer
 -----------------------------------------------------------------*/
```

OR

```
/*-----------------------------------------------------------------
[Table of contents]

1. Body
2. Header / #header
3. Navigation / #navbar
4. Content / #content
5. Left column / #leftcolumn
6. Right column / #rightcolumn
7. Sidebar / #sidebar
8. RSS / #rss
9. Search / #search
10. Boxes / .box
11. Sideblog / #sideblog
12. Advertisements / .ads
13. Footer / #footer
-----------------------------------------------------------------*/
```

AND

```
/*-----------------------------------------------------------------
[8. RSS / #rss]
*/
#rss { ... }
#rss img { ... }
```

27. Define your colors and typography

(Reference Smashing)

Since we don't have CSS constants yet, we need to figure out some way to get a quick reference of "variables" we are using. In web development colors and typography can often be considered as "constants" — fixed values that are used throughout the code multiple times.

As Rachel Andrew states, "one way to get round the lack of constants in CSS is to create some definitions at the top of your CSS file in comments, to define constants. A common use for this is to create a color glossary. This means that you have a quick reference to the colors used in the site to avoid using alternates by mistake and, if you need to change the colors, you have a quick list to go down and do a search and replace."

```
/*-------------------------------------------------------------------
 # [Color codes]

 # Dark grey (text): #333333
 # Dark Blue (headings, links) #000066
 # Mid Blue (header) #333399
 # Light blue (top navigation) #CCCCFF
 # Mid grey: #666666
 # */
```

OR

```
/*-------------------------------------------------------------------
 [Color codes]

 Background: #ffffff (white)
 Content:   #1e1e1e (light black)
 Header h1:  #9caa3b (green)
 Header h2:  #ee4117 (red)
 Footer:    #b5cede (dark black)

 a (standard):   #0040b6 (dark blue)
 a (visited):   #5999de (light blue)
 a (active): #cc0000 (pink)
-------------------------------------------------------------------*/
```

AND

```
/*-------------------------------------------------------------------
 [Typography]

 Body copy:     1.2em/1.6em Verdana, Helvetica, Arial, Geneva, sans-serif;
 Headers:       2.7em/1.3em Helvetica, Arial, "Lucida Sans Unicode", Verdana, sans-serif;
 Input, textarea:   1.1em Helvetica, Verdana, Geneva, Arial, sans-serif;
 Sidebar heading:   1.5em Helvetica, Trebuchet MS, Arial, sans-serif;

 Notes:  decreasing heading by 0.4em with every subsequent heading level
 -------------------------------------------------------------------*/
```

28. Be consistent.

If you're editing code, take a few minutes to look at the code around you and determine its style. If they use spaces around all their arithmetic operators, you should too. If their comments have little boxes of hash marks around them, make your comments have little boxes of hash marks around them too.
The point of having style guidelines is to have a common vocabulary of coding so people can concentrate on what you're saying rather than on how you're saying it. We present global style rules here so people know the vocabulary, but local style is also important.
If code you add to a file looks drastically different from the existing code around it, it throws readers out of their rhythm when they go to read it. Avoid this.

/*-------- END OF CSS GUIDELINES --------*/

5. JAVASCRIPT:

1. Basically, see Airbnb JavaScript Style Guide. We like that: https://github.com/airbnb/javascript
    a. *Audit your PLAN and CODE against these (Airbnb) guidelines FREQUENTLY throughout development.*

OTHER JAVASCRIPT "TO DO'S" AND "DO NOT DO'S":

1. See: GENERAL RULES FOR ALL CODE
    a. Use 2-space indentation everywhere
    b. Apply correct indentation everywhere; not all functions/blocks are indented properly
    c. Remove any errant trailing spaces/tabs ("show invisibles" in your IDE to find them)
    d. 'Space out' the JS file a bit more; put some empty lines between each 'stanza' of code
2. Use correct comment form: comments need spaces after comment delimiters:
    a. WRONG: //Add Article page Scrolling jQuery
    b. CORRECT: // Add Article page Scrolling jQuery
3. Avoid setting CSS dimensions in JS functions
    a. *https://github.com/digitalpulp/plos-alm-templates/issues/85

```
WRONG:
$("div").css("margin", "0");

CORRECT:
In JS:
// Class: position: This class is being used to manipulate the position of x.
// Reference CSS file: style.css
$("div").addClass("position");

In CSS:
.position {
  margin: 0;
}
```

4.

**** IGNORE THE BELOW FOR NOW. THIS INFORMATION NEEDS TO BE SORTED AND PRIORITIZED ****

ADDITIONAL HTML GUIDELINES:

1. Understand and use the following where most appropriate:
    1. Header:
        a. <header> *HTML5 tag
    2. Container:
        a.
    3. Main Navigation without drop-down:
        a. <nav> *HTML5 tag
    4. Main Navigation with drop-down:
        a. <nav> *HTML5 tag
    5. Sub-Navigation without drop-down:
        a. <nav class="subnav"> *HTML5 tag
    6. Sub-Navigation with drop-down:
        a. <nav class="subnav">
    7. Sidebar "Block":
        a. <aside class="sidebar"> *HTML5 tag
    8. Main Content:
        a. <section class="main"> *HTML5 tag
    9. Footer:
        a. <footer> *HTML5 tag

3. See what the experts do ("View Source" for these websites.)
    a. view-source:http://www.bigspaceship.com/
    b. view-source:http://twitter.com/
    c. view-source:http://www.akqa.com/
    d. view-source:http://www.b-reel.com/

**** IGNORE THE BELOW FOR NOW. THIS INFORMATION NEEDS TO BE SORTED AND PRIORITIZED ****

1. IGNORE THIS FOR NOW:
http://thesassway.com/articles/sass-doesnt-create-bad-code-bad-coders-do

2. IGNORE THIS FOR NOW: http://www.youtube.com/watch?v=ZpFdyfs03Ug

Will there be any instances where you will be applying a class within a class? AKA NESTING.

No need to make a brand new class name here, just apply a new class right in the class attribute. You can use the class name "widget" as your hook to apply the same set of styling to each one of these. And you can use the class name "big" to assign some slightly different styling rules to the widget that is bigger than the others.