Coding Conventions

Details

Written by Hayden Young
Category: Technical (/organizational/technical)
Published: 07 April 2015
Hits: 27

HTML, (/component/tags/tag/2-html) **CSS, (/component/tags/tag/3-css)** **LESS, (/component/tags/tag/4-less)**
Javascript, (/component/tags/tag/5-javascript) **coding conventions (/component/tags/tag/6-coding-conventions)**

This document covers coding conventions for front-end languages HTML, CSS and Javascript as well as their derivatives such as SASS/LESS and Joomla template overriding.

General Best Practices

- Become very familiar with this guide. This is the way we code at KnowledgeArc,
- DO NOT pollute source code repositories with unnecessary files, such as your development environment's project file or Apple's .DS-Store directories. These add compolexity to code management and, if deployed to the server, present a security issue,
- If developing on Microsoft Windows or Mac, ensure your development environment supports UNIX linefeeds (\n) and that the source code file ends with a new line. Use tools such as dos2unix if there are issues with end-of-line characters. Please be aware that Windows uses carriage return plus linefeed and Mac uses carriage return,
- Commit and push to Github often,
- DO NOT pollute Github with files ending with suffixes such as .old, .bak, .bak.bak, etc. This defeats the purpose of having a source control system,
- Use build scripts to automate common tasks such as packaging software,
- Reuse code wherever possible. Use code libraries such as composer and packagist,
- Document your code; classes, variables, methods, etc. Documentation should be concise and easy to understand,
- Use Gitbook for documenting user guides and and other end-user documentation.

Github

- Commit often,
- Use gitignore to ignore unnecessary files,
- DO NOT force changes. Liaise with other programmers if something needs merging,
- Basic information; installation, configuration, extending; can be defined in the README.md. If the software is large and complex, usability MUST be documented in a Gitbook.

PHP

- KnowledgeArc uses a combination of PHP FIG (http://www.php-fig.org/) PSR-2 (http://www.php-fig.org/psr/psr-2/) and Joomla! coding conventions (http://joomla.github.io/coding-standards/). Where there is a discrepancy between PSR and Joomla!, PSR will take precedence,
- Use Phing to automate build processes,
- Use Composer to manage 3rd party libraries,
- Use PHPDocumentor notation for your code documentation.

HTML

- Use semantic markup. For example, `<aside></aside>` is more semantically correct than `<div class="sidebar"></div>`
- Use semantic CSS ids and classes. id="firstName" or class="footer-modules". id="mb" or class="mb" is semantically vague.
- Limit the number of classes to one (1) for the class HTML element attribute. Use LESS to handle class inheritance. For example:

```
<div class="mb-- controls text-box"></div>
```

is confusing and difficult to maintain. The following is more semantically correct, easier to manage and can be more easily

managed using LESS inheritance:

```
<div class="text-box"></div>
```

Your LESS code would then look something like:

```
.text-box {
    &.field-control;
    &.field;

    border: 1px solid @border-color;
}
```

CSS

These conventions apply to both CSS and LESS. Where possible utilize LESS as it is more re-useable.

- CSS ids must be camelCase, E.g. #mainContainer, #firstName.
- CSS classes must be hyphenated, E.g. .footer-module, .form-field.
- Indentation is 4 spaces. DOT NOT USE tabs.

The following code displays syntactically correct LESS CSS:

```
#firstName {
    font-weight: bold;
}

.container
    > .footer-module {
        background: @module-color;
    }

    @media(min-width: @screen-desktop) {
        .makeColumn(6);
    }

    @media
        (min-width: @tablet),
        (min-width: @mobile) {

    }
}

form {
    .fieldset {
        border: 1px solid @border-color;

        .field {
            float: left;
        }

        input[text],
        textarea,
        .radio {
            font-size: 12pt;
        }
    }
}
```

- CSS structures must name the selector along with a space and then a curly brace on the same line. The closing brace must vertically align with the start of the selector.
- Child selectors must be indented 4 spaces relative to their parent (as seen in the "form" selector definition).
- CSS elements must be indented 4 spaces from the beginning of their selector.
- If more than one selector has the same style, list each selector under the previous (as seen with the selectors input[text], textarea, .radio)
- If there are multiple @media definitions, each one should be indented 4 spaces from the beginning of the @media definition (as seen with @media (min-width: @tablet), (min-width: @mobile).

Software Versioning

- KnowledgeArc uses semantic versioning (http://semver.org/),
- E.g. 1.2.2-dev, 1.2.2-alpha, 1.2.2-beta, 1.2.2-rc1, 1.2.3

Joomla

- Use the Joomla! API for as much development as possible. If something can be done natively in PHP or in Joomla, use the Joomla! method,
- Projects are often made up of multiple extensions; components, modules, plugins, languages and templates. Each extension must

have its own version assigned to it even though each extension's version may differ within the same project.

- When all development has been complete for a particular version, the entire repo at a particular point in time is tagged as a new version,
- Extensions which are packaged (I.e. relate to one another and cannot function without one another) should be housed within the same repo. Supporting extensions which are not required for the core group of extensions to run should be housed in their own repo,
- Packages should be versioned with the latest extension version. E.g. if the package is made up of com_extension v1.2.3, plg_extension v2.3.4 and mod_extension 0.2.1 then the package version will be v2.3.4.