**HTML coding standards**

Formatting

All HTML documents must use two spaces for indentation and there should be no trailing whitespace.

Doctype and layout

All documents must be using the HTML5 doctype and the <html> element should have a "lang" attribute. The <head> should also at a minimum include "viewport" and "charset" meta tags.

Forms

Form fields must always include a <label> element with a "for" attribute matching the "id" on the input. This helps accessibility by focusing the input when the label is clicked, it also helps screen readers match labels to their respective inputs.

Each <input> should have an "id" that is unique to the page. It does not have to match the "name" attribute.

Forms should take advantage of the new HTML5 input types where they make sense to do so, placeholder attributes should also be included where relevant. Including these can provided enhancements in browsers that support them such as tailored inputs and keyboards.

Including meta data

Classes should ideally only be used as styling hooks. If you need to include additional data in the HTML document, for example to pass data to JavaScript, then the HTML5 data- attributes should be used.

These can then be accessed easily via jQuery using the .data() method.

One thing to note is that the JavaScript API for datasets will convert all attribute names into camelCase. So "data-file-format" will become fileFormat.

Ideally you should be using CKAN’s JavaScript module format for defining how JavaScript is initiated and interacts with the DOM.

Targeting Internet Explorer

Targeting lower versions of Internet Explorer (IE), those below version 9, should be handled by the stylesheets. Small fixes should be provided inline using the .ie specific class names. Larger fixes may require a separate stylesheet but try to avoid this if at all possible.

**CSS coding standards**

Formatting

All CSS documents must use two spaces for indentation and files should have no trailing whitespace. Other formatting rules:

Use soft-tabs with a two space indent.

Use double quotes.

Use shorthand notation where possible.

Put spaces after : in property declarations.

Put spaces before { in rule declarations.

Use hex color codes #000 unless using rgba().

Always provide fallback properties for older browsers.

Use one line per property declaration.

Always follow a rule with one line of whitespace.

Always quote url() and @import() contents.

Do not indent blocks.

Naming

All ids, classes and attributes must be lowercase with hyphens used for separation.

Comments

Comments should be used liberally to explain anything that may be unclear at first glance, especially IE workarounds or hacks.

Modularity and specificity

Try keep all selectors loosely grouped into modules where possible and avoid having too many selectors in one declaration to make them easy to override.

Instead here we would create a dataset “module” and styling the item outside of the container allows you to use it on it’s own

In the same vein use classes make the styles more robust, especially where the HTML may change.

You may use pseudo selectors to keep the HTML clean:

However this will break any time the HTML changes for example if an item is added or removed. Instead we can use class names to ensure the icons always match the elements (Also you’d probably sprite the image :).

Avoid using tag names in selectors as this prevents re-use in other contexts.

Also ids should not be used in selectors as it makes it far too difficult to override later in the cascade.

**JavaScript coding standards**

Formatting

All JavaScript documents must use two spaces for indentation. This is contrary to the OKFN Coding Standards but matches what’s in use in the current code base.

Coding style must follow the idiomatic.js style but with the following exceptions.

White space

Two spaces must be used for indentation at all times. Unlike in idiomatic whitespace must not be used \_inside\_ parentheses between the parentheses and their Contents.

Quotes

Single quotes should be used everywhere unless writing JSON or the string contains them. This makes it easier to create strings containing HTML.

Object properties need not be quoted unless required by the interpreter.

Variable declarations

One var statement must be used per variable assignment. These must be declared at the top of the function in which they are being used.

Declare variables at the top of the function in which they are first used. This avoids issues with variable hoisting. If a variable is not assigned a value until later in the function then it it okay to define more than one per statement.

Naming

All properties, functions and methods must use lowercase camelCase:

Constructor functions must use uppercase CamelCase:

Constants must be uppercase with spaces delimited by underscores:

Event handlers and callback functions should be prefixed with “on”:

Boolean variables or methods returning boolean functions should prefix the variable name with “is”:

Private methods should be prefixed with an underscore:

Functions should be declared as named functions rather than assigning an anonymous function to a variable.

Named functions are generally easier to debug as they appear named in the debugger.

Comments

Comments should be used to explain anything that may be unclear when you return to it in six months time. Single line comments should be used for all inline comments that do not form part of the documentation.

JSHint

All JavaScript should pass JSHint before being committed. This can be installed using npm (which is bundled with node) by running:

Each project should include a jshint.json file with appropriate configuration options for the tool. Most text editors can also be configured to read from this file.

Documentation

For documentation we use a simple markup format to document all methods. The documentation should provide enough information to show the reader what the method does, arguments it accepts and a general example of usage. Also for API’s and third party libraries, providing links to external documentation is encouraged.

Testing

For testing we use Cypress.

Tests are run from the cypress directory. We use the BDD interface (describe(), it() etc.).

Generally we try and have the core functionality of all libraries and modules unit tested.

Best practices

Forms

All forms should work without JavaScript enabled. This means that they must submit application/x-www-form-urlencoded data to the server and receive an appropriate response. The server should check for the X-Requested-With: XMLHTTPRequest header to determine if the request is an ajax one. If so it can return an appropriate format, otherwise it should issue a 303 redirect.

The one exception to this rule is if a form or button is injected with JavaScript after the page has loaded. It’s then not part of the HTML document and can submit any data format it pleases.

Ajax

Ajax requests can be used to improve the experience of submitting forms and other actions that require server interactions. Nearly all requests will go through the following states.

User clicks button.

JavaScript intercepts the click and disables the button (add disabled attr).

A loading indicator is displayed (add class .loading to button).

The request is made to the server.

On success the interface is updated.

On error a message is displayed to the user if there is no other way to resolve the issue.

The loading indicator is removed.

The button is re-enabled.

This covers possible issues that might arise from submitting the form as well as providing the user with adequate feedback that the page is doing something. Disabling the button prevents the form being submitted twice and the error feedback should hopefully offer a solution for the error that occurred.

Event handlers

When using event handlers to listen for browser events it’s a common requirement to want to cancel the default browser action. This should be done by calling the event.preventDefault() method:

It is also possible to return false from the callback function. Avoid doing this as it also calls the event.stopPropagation() method which prevents the event from bubbling up the DOM tree. This prevents other handlers listening for the same event. For example an analytics click handler attached to the <body> element.

Also jQuery (1.7+) now provides the .on() and .off() methods as alternatives to .bind(), .unbind(), .delegate() and .undelegate() and they should be preferred for all tasks.

Templating

Small templates that will not require customisation by the instance can be placed inline. If you need to create multi-line templates use an array rather than escaping newlines within a string:

Always localise text strings within your template. If you are including them inline this can be done with jQuery:

Larger templates can be loaded in using the CKAN snippet API. Modules get access to this functionality via the sandbox.client object:

The primary benefits of this is that the localisation can be done by the server and it keeps the JavaScript modules free from large strings.