Vocabulary Widget

Introduction

The Vocabulary Widget allows you to **easily add Data Classification capabilities** to your data capture tools by drawing on vocabularies hosted at Research Vocabularies Australia.

The widget has been written in the style of a jQuery plugin, allowing complete control over styling and functionality with just a few lines of JavaScript. The widget also ships with some UI helper modes for:

- searching for vocabulary terms, with autocompletion;
- populating a select list (or autocomplete textbox) with items derived from a base IRI (which is either the broadest term of a hierarchy, or a collection);
- browsing a hierarchical vocabulary as a tree.

It is also possible to use the widget in a more programmatic manner; refer to the Advanced Configuration section below for more details.

How do I use the widget?

Pull in the widget code and styles

The widget requires jQuery; load this, and the plugin itself (and associated CSS styles) in your document's `<head>...</head>` segment:

```
<script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.js"><
/script>
<script type="text/javascript" src="//vocabs.ardc.edu.au/apps/assets
/vocab_widget/js/vocab_widget_v2.js"></script>
<link rel="stylesheet" type="text/css" href="//vocabs.ardc.edu.au/apps
/assets/vocab_widget/css/vocab_widget_v2.css" />
```

If you are using tree mode (explained below), you may wish to display the widget in a tooltip. If so, add this to pull in support for qTip2 tooltips:

```
<script type="text/javascript" src="//vocabs.ardc.edu.au/assets/core/lib
/qtip2/jquery.qtip.js"></script>
<link rel="stylesheet" type="text/css" href="//cdn.jsdelivr.net/qtip2/2.2.1
/basic/jquery.qtip.min.css" media="screen" />
```

> The code generated for tree mode by the Widget Explorer requires qTip2. When incorporating the results of the Widget Explorer, make sure to include the content from both of the code blocks given above.

Instantiate the widget

Add widget input and initialisation code into your HTML document's `<body>...</body>` section. The input/initialisation code selects the vocabulary, input mode, and other parameters. Write the code yourself, based on the instructions in the following sections, or use the Widget Explorer to generate it for you.

> You can combine the two approaches: use the Widget Explorer to generate boilerplate input/initialisation code, then customise it to suit your needs.

Demo

This section gives one demo of each of the main supported modes.

Search mode

Search a vocabulary for matching terms, provided in an autocomplete-style list:

Narrow and collection modes

Narrow and collection modes can be attached to a `select` element to **provide a drop-down**, or a text `input` box for an autocomplete-style list:

- Narrow mode supports selection from the concepts that are narrower than a specified concept (to be specific, the concepts that are `skos:narrower` than a specified `skos:Concept`).
- Collection mode supports selection from the members of a collection (to be specific, the `skos:members` of a specified `skos:Collection`).

Narrow mode example

This demo uses a `<select>` element.

Collection mode example

This demo uses an `<input>` element that supports autocompletion.

Tree mode

Tree mode constructs a clickable vocabulary tree for a given repository. Bind to the `treeselect.vocab.ands` event to handle user selection.

Configuration

The plugin accepts a suite of options, detailed below. Please note that some options are required, and don't have default values (such as `repository`): you must provide values for such options. Incorrectly configured plugins will result in a JavaScript "alert" box being displayed, describing the nature of the configuration problem.

Options are passed into the plugin as a JavaScript object. For example:

```
$("#vocabInput").vocab_widget({cache: false});
```

Be sure to quote strings, and separate multiple options with a comma (`,`).

Alternatively, options can be set after initialisation using the following form:

```
$(...).vocab_widget("[option-name]", [option-value]);
```

This works for all options *except* `mode`, which must be specified at initialisation (or omitted for core usage).

Some options are specific to the chosen mode; the tables below are grouped in a way that makes this easy to comprehend. Core usage of the widget exposes all "common" options.

Note: `tree` mode has no specific configuration other than the widget's common options.

Common options

Property	Defaults	Description
`api_key`	`"public"`	The API key you have previously been issued. Please request an API key by emailing `services@ardc.edu.au`.
`mode`	`""`	REQUIRED Vocab widget mode: • `"tree"`: provides a hierarchical display based on broader/narrower relations • `"search"`: provides an autocomplete widget on an HTML input element • `"narrow"`: populates an HTML select element with appropriate data based on narrower relations. • `"collection"`: populates an HTML select element with data from a collection. • `"advanced"` or `"core"`: exposes the core widget with no UI helpers.
`repository`	`""`	REQUIRED The SISSVoc repository to query. This can be specified either as: • a base repository name, e.g., `"anzsrc-for"`, `"rifcs"` • a URL that is the prefix of the SISSVoc endpoints for the vocabulary, e.g., `"http://vocabs.ardc.edu.au/repository/api/lda/aodn/aodn-discovery-parameter-vocabulary/version-1-2"`
`max_results`	`100`	At most, how many results should be returned? Note: the vocabulary's SISSVoc configuration may override this option by applying a lower (but not a higher) maximum limit. At present, the SISSVoc configuration of most vocabularies specifies a maximum limit of 200 results.
`cache`	`true`	Cache SISSVoc responses?
`error_msg`	`"ANDS Vocabulary Widget service error."`	Message title to display (via a JavaScript `alert()` call) when an error is encountered. Set to `false` to suppress such messages
`endpoint`	`"https://vocabs.ardc.edu.au/api/v1.0/vocab.jsonp/"`	Location (absolute URL) of the "back end" of the widget (an interface to SISSVoc that generates responses in JSONP).

Search mode options

In this mode, the target element should be `<input type="text" ...>`.

Property	Defaults	Description
`min_chars`	`3`	How many characters are required before a search is executed?
`delay`	`500`	How long to wait (after initial user input) before executing the search? Provide in milliseconds

`nohits_msg`	`"No matches found"`	Message to display when no matching concepts are found. Set to `false` to suppress such messages
`list_class`	`"vocab_list"`	CSS ‘class’ references for the dropdown list. Separate multiple classes by spaces
`fields`	`["label", "notation", "about"]`	Which fields do you want to display? Available fields are defined by the chosen repository.
`target_field`	`"notation"`	What data field should be stored upon selection?

Narrow and collection mode options

In these modes, the target element should be `<select ...>`.

Property	Defaults	Description
`mode_params`	`""`	REQUIRED In these modes mode, `mode_params` defines the vocabulary item upon which to narrow. The value is the IRI of the vocabulary item.
`fields`	`["label", "notation", "about"]`	In these modes, if the target element is a `<select>`, this option *must be overridden* to be an array of *one* string (e.g., `["label"]`). This selection defines the label for the select list options.
`target_field`	`"notation"`	What data field should be stored upon selection? This field is used as the `value` attribute for the select list options.

Advanced Configuration

Core mode

Invoking the plugin with no `mode` argument, or with the `mode` set either to `"core"` or `"advanced"` exposes core functionality, without having to use form input (text, select) elements or the like. Instead, you hook into JavaScript Events, building the UI as best fits your needs. A very basic example is shown below: it constructs a list of RIF-CS identifier types.

This form section is using the widget with no helpers; it outputs a list of known RIF-CS identifier types

Core mode exposes the functionality that underpins the other modes, making it accessible through calls to the `vocab_widget()` function. Having initialised the widget in core mode, you call `vocab_widget()` again, passing in one of the following values as the first parameter:

- `"search"`
- `"narrow"`
- `"collection"`
- `"top"`

You also pass in a second parameter, which can look like either of the following:

- a string: search term (for a `"search"` call) or narrow URI (for a `"narrow"` call)
- an object `{uri: "...", callee: my_callback_object }`:
    - The `uri` provides the same data as per the description of the string value in the point above.
    - The `callee` defines the object against which the subsequent JavaScript event will be triggered. Defaults to the containing element (i.e., on which you invoked `vocab_widget()`).

When you call the `vocab_widget()` function in this way, a request is made to the widget `endpoint` (the "back end" of the widget) to fetch data from the vocabulary. When the `endpoint` replies, an event is generated and the response data passed in as the event's data. For example, if you call `vocab_widget("narrow", ...)` there will be a `narrow.vocab.ands` event generated when the `endpoint` replies.

Events

Events are fired to allow you to hook into the widget workflow and implement your customisations as you see fit. Each of the widget's modes supports configuration of event handlers, but only certain events are *fired* in each mode. For example, the `treeselect.vocab.ands` event will only be fired when the widget has been configured in tree mode. For modes other than core mode, there are default event handlers already defined, and you can override them. For core mode, there are no default handlers; you need to define handlers for each back end function you are going to request using the `vocab_widget()` function as explained in the previous section.

> Widget events are in the `vocab.ands` namespace.

Event Name	Parameters	Description
`search.vocab.ands`	1. JS Event object 2. SISSVoc data object	Hook into the plugin's `search` core function; `data` is the search response.
`narrow.vocab.ands`	1. JS Event object 2. SISSVoc data object	Hook into the plugin's `narrow` core function; `data` is the search response.
`collection.vocab.ands`	1. JS Event object 2. SISSVoc data object	Hook into the plugin's `collection` core function; `data` is the search response.
`top.vocab.ands`	1. JS Event object 2. SISSVoc data object	Hook into the plugin's `top` core function; `data` is the search response.
`searchselect.vocab.ands`	1. JS Event object	(Fired in search mode, and in narrow and collection modes when the target is an `<input>` element.) Fired when a search result item is clicked. The selected item is the event target. The target will have a `vocab` data object, containing all the details found in a SISSVoc data object.
`treeselect.vocab.ands`	1. JS Event object	(Fired in tree mode.) Fired when a tree item is clicked. The selected item is the event target. The target will have a `vocab` data object, containing all the details found in a SISSVoc data object.
`error.vocab.ands`	1. JS Event object 2. XMLHttpRequest	This event is fired whenever there is a problem communicating with the plugin's endpoint (the widget back end). > If the error occurred during an AJAX call, the object will be a bona fide XMLHttpRequest object. Otherwise, a plain object with `status` and `responseText` properties will be available.

Data

The SISSVoc `data` object returned by the above events (and also attached to the `searchselect` and `treeselect` event's `vocab` data object) is a plain object with the following properties:

- `status`: `OK` if all good, something else (most likely `ERROR`) if not
- `message`: description of the underlying system call by default, or information on status when something went wrong
- `limit`: the maximum number of records requested

- `items`: an array of SISSVoc vocabulary items:
    - `definition`: item description
    - `label`: item label
    - `about`: item definition / URL
    - `broader`: parent term (if it exists)
    - `narrower`: child terms (if they exist, otherwise boolean false)
    - `count`: frequency of use among ARDC registry objects (experimental; requires endpoint to be set to `"http://researchdata.edu.au/apps/vocab_widget/proxy/"`; works best on ANZSRC-FOR, not so well on RIF-CS)
- `count`: the number of items returned