Accessing the Vocabulary Registry API

« Vocabulary Registry model entities   Vocabulary Registry data »

This page sets out a variety of ways in which humans and programs may interact with the Registry API.

The Portal

It is worth noting again that the Vocabulary Portal (i.e., the web-based "front end" accessible at https://vocabs.ardc.edu.au/) is a client of the Registry API. In general, what you can do in the Portal, you can also do yourself using the Registry API. (Really the only exception is to do with authentication. The Portal provides a means of "logging in" as certain classes of user (e.g., social) that the Registry does not.)

The Swagger UI

RVA hosts an instance of Swagger UI that is already configured to connect to the Registry. You can access it here: https://vocabs.ardc.edu.au/registry/swagger-ui/

(There is also an instance on the demo server: https://demo.vocabs.ardc.edu.au/registry/swagger-ui/)

Direct access to API URLs

The API is accessible through HTTPS. This means that you can access API methods directly using a web browser and using command-line tools such as `wget` and `curl`.

Here's an example of a URL that will provide the top-level description of one vocabulary, in XML format: https://vocabs.ardc.edu.au/registry/api/resource/vocabularies/1.

The Swagger UI (described in the previous section) shows you the URL for each API method, and shows how to send requests using the `curl` tool.

REST tools

The API is accessible through HTTPS. This means that you can access API methods using REST tools, whether standalone or embedded in web browsers. Some convenient tools for API access include:

- Postman (https://www.getpostman.com/): commercial product, with free-of-charge version
- Paw (https://paw.cloud/): commercial product, with free-of-charge, open source version (MIT License)
- Insomnia (https://insomnia.rest/): commercial product
- Advanced REST Client (https://install.advancedrestclient.com/): open source (Apache 2.0 License)

(This list of tools is provided only for the convenience of API users; it should not be interpreted as a recommendation of any particular tool.)

With such tools, it is easy to use API methods that require authentication, and you can select the request and response content types. The following section shows how to create a Postman Collection by direct import of the Swagger description; a similar process is available in some of the other tools.

Postman collection

Postman supports directly importing an API description in Swagger format and creating a Collection containing the API methods. To do this:

1. Click the **Import** button at the top of the Postman window.
2. Click to select the **Import From Link** tab.
3. In the text box, enter the URL for the Swagger description: `https://vocabs.ardc.edu.au/registry/swagger.json`
    1. NB: We recommend that you test out your ideas using the demo environment first. The Swagger description for demo is available at: `https://demo.vocabs.ardc.edu.au/registry/swagger.json`
4. Click the **Import** button.
5. If it is not already at the front, click to select the Collections tab. You now have a collection with the name "Vocabulary Registry API".

Please note that this import functionality is limited; Postman does not show the detailed help that is available in the Swagger UI interface. For example, it does not show the documentation for method parameters, and it does not offer sample XML or JSON data. Similar limitations may (or may not) apply to the import functionality of other REST tools.

Generated API client libraries

The API is specified in OpenAPI format, also known as Swagger. You can use the API specification with the `swagger-codegen` tool to generate an API client library in the language of your choice.

To use swagger-codegen, you will need:

- a Java run-time environment, either Java 7 or 8
- the `swagger-codegen-cli` JAR file

Depending on the language, some generated client libraries may have additional requirements for building the library. (For example, if you generate a client library for Java, you will get a generated `pom.xml` file for use with Maven.)

You can find the original download and installation instructions for `swagger-codegen` here: https://github.com/swagger-api/swagger-codegen

The following is a summary of what you need to know to get started.

ARDC uses the stable release 2.2.3 to generate the PHP and JavaScript libraries used by the Portal. Later versions in the 2.x branch may also work, but we have not tested them. The releases in the 3.x branch are still of "pre-release" quality and are not recommended for now.

To download release 2.2.3 of the swagger-codegen JAR, you can use the following direct link:

https://repo1.maven.org/maven2/io/swagger/swagger-codegen-cli/2.2.3/swagger-codegen-cli-2.2.3.jar

If you are using Linux or Unix, you can download the JAR from the command line using `wget`:

```
wget https://repo1.maven.org/maven2/io/swagger/swagger-codegen-cli/2.2.3
/swagger-codegen-cli-2.2.3.jar \
  -O swagger-codegen-cli.jar
```

This `wget` command saves the file as `swagger-codegen-cli.jar`. The following instructions assume you have named the JAR file in this way. If you downloaded the JAR by clicking on the link above, we assume that you will then manually rename the downloaded JAR file as `swagger-codegen-cli.jar`.

You can now use `swagger-codegen` to generate code. Help is available with the command:

```
java -jar swagger-codegen-cli.jar help
```

The `generate` command is used to generate client library code. You probably wish to see the command-line options first:

```
java -jar swagger-codegen-cli.jar help generate
```

To see the list of supported languages and environments, use:

```
java -jar swagger-codegen-cli.jar langs
```

(Please note that some of the supported languages do not in fact generate *client* libraries, but generate code stubs for *servers* instead. For example, the "language" `jaxrs` generates server stubs that could be used as a framework for re-implementing the Registry API. You probably don't want to do that!)

The following sections give particular examples of code generation. In each case, we use the Swagger description hosted on RVA at `https://vocabs.ardc.edu.au/registry/swagger.json`.

Example: Java

For some target languages, such as Java, you have a choice of support library for networking, and you make your choice by specifying a command-line option. To see the list of language-specific options, including the list of support libraries, run:

```
java -jar swagger-codegen-cli.jar config-help -l java
```

The end of the output gives the list of supported library options.

Let's choose `jersey2`. To generate code and to store it in the directory `jersey_client`, use this command:

```
java -jar swagger-codegen-cli.jar \
  generate -i https://vocabs.ardc.edu.au/registry/swagger.json \
  -l java -Dlibrary=jersey2 -o jersey_client
```

You now have a directory `jersey_client` containing the generated source code for the API client library. The library uses the Jersey 2 framework (https://jersey.github.io) to communicate with the API. Please read the generated `jersey_client/README.md` for instructions about how to use the library.

Example: JavaScript

In this case, you can choose between JavaScript and TypeScript. If you use TypeScript, you can also generate code for integrating with frameworks such as JQuery and AngularJS.

In this example, we choose JavaScript. First, view the list of language-specific options:

```
java -jar swagger-codegen-cli.jar config-help -l javascript
```

For the RVA portal, we use Promises for AJAX calls, and we request model methods. To generate code in this way and to store it in the directory `js_client`, use this command:

```
java -jar swagger-codegen-cli.jar \
  generate -i https://vocabs.ardc.edu.au/registry/swagger.json \
  -l javascript -DusePromises=true -DemitModelMethods=true -o js_client
```

You now have a directory `js_client` containing the generated source code for the API client library. Please read the generated `js_client/README.md` for instructions about how to use the library. Tip: we use the `browserify` package to create an all-in-one file that is included by the Portal pages.

« Vocabulary Registry model entities   Vocabulary Registry data »