RDA Metadata Standards Catalog Working Group: Final Report

Alex Ball Keith Jeffery Rebecca Koskela

2019

Contents

Executive summary

The Metadata Standards Catalog Working Group continued the work of the Metadata Standards Directory Working Group, to develop a directory that would enable researchers, and those who support them, to discover metadata standards appropriate for documenting their research data. The particular goals of this phase of work were to simplify the process of contributing to the Catalog for users, enrich the information held to support more use cases, provide a machine-to-machine API, and provide a more flexible basis for future enhancements.

The Group collected user stories describing ways the Catalog might be used. These were organized into a requirements specification that was used as the basis for developing the Catalog Web application.

The Catalog is now live and accepting contributions from the community. A Metadata Standards Catalog maintenance activity continues under the umbrella of the Metadata Interest Group. Further developments are planned, including support for analyzing metadata schemes through the lens of the Metadata Interest Group's concept of metadata packages.

1

Summary of working group activity

1.1 Problem addressed

The Metadata Standards Catalog (MSC) Working Group was the third in a series of activities to develop a resource supporting the discovery and use of metadata standards and application profiles appropriate for research data. It followed on from the Metadata Standards Directory Working Group, which in turn extended and built upon work conducted by the UK Digital Curation Centre.

The problem addressed by all these activities is that all too many research datasets are shared without adequate metadata. There may be no documentation at all. The documentation may be provided only as unstructured text, rendering it prone to incompleteness and misinterpretation; furthermore, such documentation demands a level of human attention that does not scale in the context of data-intensive science. Where structured metadata is provided, it may be in a format unrecognized by the rest of the domain, or non-conformant to the specification it claims to follow.[1,2]

The consequence of this is that the value inherent in shared research datasets cannot be realized. High quality metadata is vital if datasets are to be efficiently and robustly identified, discovered, contextualized, interpreted and reused.

In order to combat this, the mission of the Metadata Standards Catalog Working Group and its predecessors was to develop a catalog of metadata standards and profiles ensuring that researchers and research supporters would be able to

1. discover metadata schemes suitable for particular domains and types of data;
2. access specifications and other documentation explaining how those schemes should be used;
3. discover software tools and libraries that help to create and process metadata conforming to those schemes;

[1] C. Tenopir, S. Allard, K. Douglass, A. U. Aydinoglu, L. Wu, E. Read, M. Manoff, and M. Frame, "Data Sharing by Scientists: Practices and Perceptions," *PLoS ONE* 6/6 (2011): e21101. doi: https://doi.org/10.1371/journal.pone.0021101

[2] C. Willis, J. Greenberg, and H. White, "Analysis and Synthesis of Metadata Goals for Scientific Data," *Journal of the American Society for Information Science and Technology* 63/8 (2012): 1505–1520. doi: https://doi.org/10.1002/asi.22683

4. consult example records conforming to those schemes, either as individual files or in the context of a working data repository or other service.

The particular contribution of the Metadata Standards Catalog Working Group was to extend the previous work in various ways:

- to provide a greater level of detail than was previously collected, supporting academic research into the development and history of metadata standards, as well as practical implementation;
- to provide enhanced search capabilities;
- to provide an online editing interface to further reduce barriers to community contributions to the catalog;
- to provide a machine-to-machine interface, allowing information from the catalog to be used by (and potentially updated from) other services in the e-research fabric;
- to provide a more flexible basis for further enhancements and developments.

As an example of the kinds of development this work aims to support, the ultimate vision is for the MSC to be used as the data source, and possibly the interface, for a tool proposed by the Metadata Interest Group for interconverting between arbitrary standards via one or more metadata "packages." These "packages" would be groupings of metadata elements for particular purposes – such as discovery, contextualization, or detailed connection of software to data – with the following characteristics:

- Each element may be a structure instead of single-valued attribute.
- There are relationships between elements including those carrying referential and functional integrity.
- Elements may belong to more than one package.

As well as being useful for constructing converters, the packages may form the basis of novel presentations of schemas and specifications for metadata standards within the MSC. They may also be useful for designing systems and considering new standards.

1.2 Goals

The goal achieved by the Metadata Standards Catalog Working Group was to produce a catalog of metadata standards of relevance to research data. Specifically, the catalog system consists of the following components:

- A set of records describing metadata standards.

  The group interpreted "metadata standard" loosely, as a defined metadata structure and format used independently by multiple actors within a community. Some metadata standards have formal standardization status but this is not necessarily an indication of quality or utilization.

- A user interface for submitting information, searching, browsing and displaying standards information.

- A machine-to-machine interface (API) allowing automated tools to submit information, perform queries and retrieve information from the catalog.

  In this sense the catalog is "machine readable." The information provided through the API is structured in such a way that the recipient machine will be able to process and act upon it, and in this sense the catalog is also "machine actionable."

The code for the MSC is available from GitHub.[3] The MSC itself was available initially from the Digital Curation Centre domain, but has since been migrated to hosting at the University of Bath.[4]

1.3 Developing the requirements specification

Since the aim of the group was to develop a new electronic resource, it was important to begin by establishing what use cases the MSC should be addressing, and therefore what the requirements for the MSC should be.

The Metadata Standards Directory Working Group had collected a set of use cases as part of its work program.[5] These were adapted into a set of user stories[6] and arranged into four thematic groups:

- Properties by which to select records
- Additional properties to display
- GUI (Web page) functionality
- API functionality

Each user story consisted of a statement of the form, "As a <*stakeholder*>, I would like to <*interaction with the Catalog*>, so that <*benefit*>," and optionally an explanatory note. It was assigned an identifier, title and a suggested priority (must, should, or could).

This list was discussed at Plenaries 6 and 7, and opened out to the whole Working Group to extend and expand. The Working Group co-chairs reviewed the list in mid-2016, combining or splitting use cases as appropriate and confirming the priority each should be given. At the end of this process, a total of 26 user stories were listed.

The user stories were used to develop a formal requirements specification[7] consisting of the following sections:

- System description
- Data model (in outline)
- Functional requirements:

[3] https://github.com/rd-alliance/metadata-catalog-dev

[4] https://rdamsc.bath.ac.uk/

[5] A. Ball, J. Greenberg, K. Jeffery, and R. Koskela, *RDA Metadata Standards Directory Working Group: Final Report* (Research Data Alliance, 2016), https://rd-alliance.org/group/metadata-standards-catalog-wg/outcomes/metadata-standards-directory-wg-recommedations.html

[6] https://goo.gl/wFTwHM

[7] https://goo.gl/7bdSiz

  - Search or browse (9 requirements)
  - Display (10 requirements)
  - Update (2 requirements)
  - Compare (2 requirements)
- Non-functional requirements (4 requirements)

Again, each requirement was given a priority (must, should, or could); the requirements were validated with reference to the corresponding user stories, and dependencies between the requirements were noted.

These requirements were presented to the Working Group at Plenary 8 for approval.

1.4 Developing the Catalog

A source code repository was set up on GitHub[8] as the central focus for developing the Catalog. Requirements from the specification were transferred into the repository's issue tracker so that progress against them could be monitored.

Some efforts were made to attract funding or volunteer effort to help with developing the Catalog. In the end, the University of Bath generously donated the time of one of the co-chairs for this purpose.

The first step was to devise a data model for the Catalog that could satisfy the requirements specification, while also permitting straightforward migration of information from the Metadata Standards Directory database and cohering with the set of recommended metadata elements being developed by the Metadata Interest Group. A summary of the eventual data model is provided in Tables 1.1 and 1.2.

Table 1.1: The entities of the MSC data model and their elements.

Metadata Scheme	Tool	Mapping	Organization	Endorsement
identifiers	identifiers	identifiers	identifiers	identifiers
locations	locations	locations	locations	locations
title	title		name	
data types	types		type	
description	description	description		citation
versions	versions	versions		
keywords	creators	creators		date issued
samples				date valid

It was decided to group all versions of the same entity under the same record, with the aim of reducing human effort in maintaining the records at the expense of some additional computational complexity. Therefore, for the entities with a "versions" element, each version listed would have a version number and a date. Where the

[8] https://github.com/rd-alliance/metadata-catalog-dev

Table 1.2: Relationships between entities in the MSC data model.

Entity	has...	Entity
Metadata Scheme	parent scheme	Metadata Scheme
Metadata Scheme	maintainer	Organization
Metadata Scheme	funder	Organization
Metadata Scheme	user	Organization
Tool	supported scheme	Metadata Scheme
Tool	maintainer	Organization
Tool	funder	Organization
Mapping	input scheme	Metadata Scheme
Mapping	output scheme	Metadata Scheme
Mapping	maintainer	Organization
Mapping	funder	Organization
Endorsement	endorsed scheme	Metadata Scheme

properties of the entity varied between versions, these could be given at the version level instead of the top level.

Another issue to consider was the controlled vocabulary used for domain classification. The Metadata Standards Directory had used a scheme based closely on that used by the DCC Disciplinary Metadata Catalogue, which was in turn based on the UK Joint Academic Coding System (JACS)[9] with some custom high-level groupings. The Working Group co-chairs felt that it would be better to use a vocabulary was international in origin, and after examining some alternatives decided on the UNESCO Thesaurus.[10] The particular advantages of this were as follows:

- It had good coverage across a wide range of academic domains.
- It had terms at approximately the same granularity as JACS, making migration from one to the other relatively straightforward.
- The terms had already been translated into various languages, potentially aiding clarity internationally.
- Being a UN product, it avoided national bias.
- It had an RDF representation, making it easy to load into the Catalog system with the hierarchy of terms intact.

The one adaptation made to the Thesaurus was to combine the groups, subgroups and concepts into a single hierarchy.

Given that the Metadata Standards Directory was effectively using a NoSQL database, in the form of four directories of YAML files, and the Catalog data model saw

[9] https://www.hesa.ac.uk/support/documentation/jacs/jacs3-detailed

[10] http://vocabularies.unesco.org/browser/thesaurus/en/

version-level records nesting inside records at the intellectual entity level, it made sense to continue with the NoSQL approach. A script was written, therefore, to migrate the Directory YAML files into Catalog YAML files, and then compile the Catalog YAML files into a (JSON-based) NoSQL database. Only a small amount of human intervention was needed to handle cases where JACS terms did not map cleanly to the UNESCO Thesaurus.

For the the development of the Catalog Web application itself, it was decided to use the Flask[11] microframework for Python.[12] Flask is well suited to rapid development and is supported by a large number of plugins for specific tasks such as authentication, authorization, and form handling. The Developer Guide below lists the Flask-specific and generic Python modules chosen as dependencies.

As well as the formal requirements specification, development of the Catalog was guided by the following considerations:

- To encourage as much community participation as possible, the barriers to contribution should be kept low.
- Since this implies the barrier to abuse of the system would also be low, the database should be version controlled so that vandalism can be reverted and offending user accounts identified. There should also be a mechanism for blocking users.
- To reduce security risks, the Catalog should hold a minimum level of personal information about users, and should hold no user passwords or password hashes. Authentication should be delegated to existing OpenID providers such as the RDA Website.

Authenticating API clients via OpenID did not turn out to be practicable, so the Catalog does handle password hashes for this subset of users. Version control of the database turned out to be relatively straightforward since the database is stored on disk as JSON.

The initial development of the Catalog application was completed in July 2017. Subsequent to this, development effort concentrated on moving the Catalog into production. The DCC kindly agreed to host the catalog, and the Working Group co-chairs would like to acknowledge the assistance and support of Jimmy Angelakos and Sam Rust in setting up and maintaining the first live instance. In early 2019, the live instance was moved to hosting generously provided by the University of Bath Library in order to simplify its ongoing maintenance.

The following User Guide and Developer Guide refer to the Catalog as it was at the time of the migration. Since further developments are planned as part of the maintenance activity, please check GitHub[13] for the latest documentation.

[11] http://flask.palletsprojects.com/

[12] https://www.python.org/

[13] https://github.com/rd-alliance/metadata-catalog-dev

2

User Guide to the Catalog

2.1 Consulting the Catalog

It is possible to look up records for metadata standards, profiles and schemes in three ways: (*a*) browsing an alphabetical, hierarchical list of schemes; (*b*) browsing an alphabetical, hierarchical list of subject terms drawn from the UNESCO Thesaurus; or (*c*) searching.

The corresponding three pages can all be reached from links on the front page. The search page is can also be reached from the navigation bar at the top of each screen.

An alphabetical list of the metadata-related tools in the Catalog is also provided, but there is no corresponding search function. It is anticipated that most users will come to the tools via the metadata scheme records, or else can use their browser's "Find in page" facility to search for tools by name on the index page, but feedback on this point is welcome.

Browsing by scheme name

The browse list for metadata schemes is arranged so that application profiles are shown below the scheme(s) on which they are based in a lower-order list. This means that the schemes listed at the top level are independently defined (i.e., not derived or adapted from another scheme), and that profiles based on multiple schemes may appear in several places.

Browsing by subject term

The first term in the list is "Multidisciplinary," capturing those schemes without a particular disciplinary focus. The remaining top level terms correspond to the UNESCO Thesaurus "groups"; these are not filtered, so following some of these links may result in an empty list of schemes.

The second-level terms correspond with UNESCO Thesaurus sub-groups, and lower level terms with UNESCO Thesaurus concepts. These are filtered so terms with no associated schemes are not shown.

Searching

It is possible to search on the following fields:

- title or name of the metadata scheme;
- subject keyword (i.e., group, sub-group, or concept from the UNESCO Thesaurus);
- identifier (internal or external);
- funder;
- data type (keyword or identifying URL).

In the screenshot (Figure 2.1), the latter two are hidden since at the time of writing data types and funding information had not been added to the Catalog database.

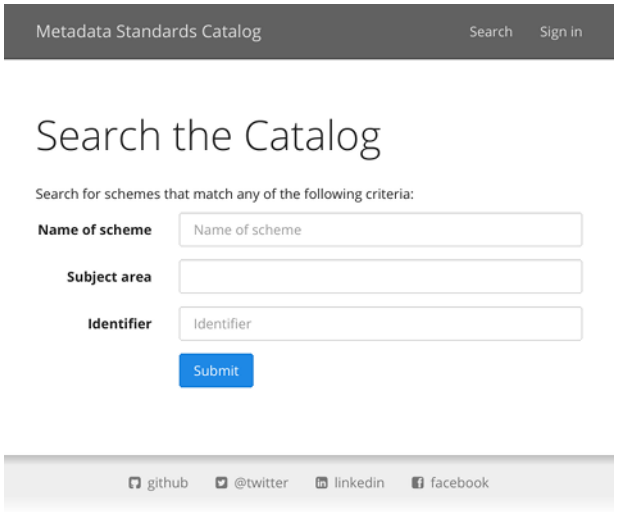

Figure 2.1: Search form

All the above fields autocomplete with values present in the database. If more than one field is used, the search is broadened: the search criteria are joined with a Boolean "OR." Within a given field it is possible to use wildcard syntax, i.e., "*" for 0 or more missing characters, "?" for one missing character. It is not currently possible, however, to search for more than one value at once within a given field. Again, feedback is welcome on whether to support more advanced searches.

Viewing

When searching or browsing by subject, the results come back as a series of panels showing the name and description of each matching scheme (see Figure 2.2). Select the name to navigate to the record.

The records themselves start with the name and description of scheme, followed by the other information collected about it (see Figure 2.3).

- You can search for other schemes tagged with the same subject term or data type by selecting the appropriate button.
- Selecting buttons such as "View specification" or "View website" will take you off-site to the respective resource.

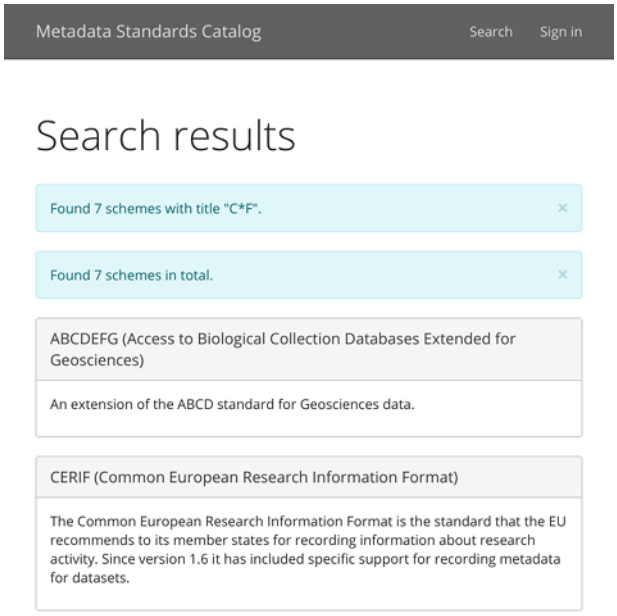

Figure 2.2: Search results

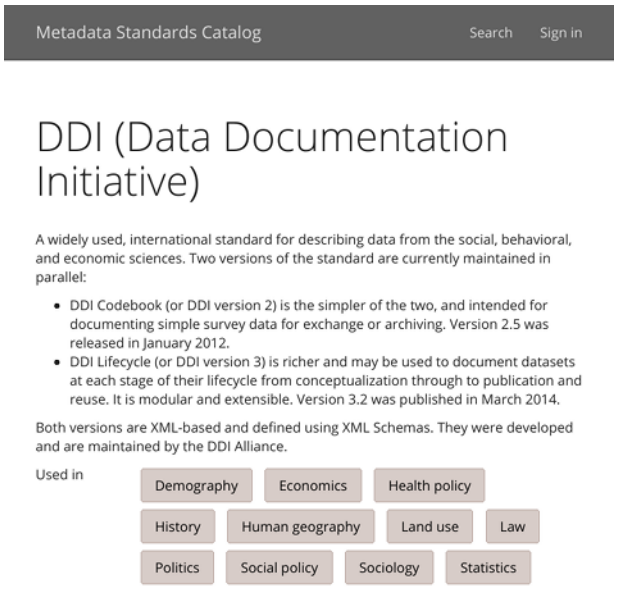

Figure 2.3: Catalog page for the DDI standard

- You can select to name of a maintaining organization, funding body or known (corporate) user of a scheme to search for other schemes maintained, funded or used by that organization, respectively.
- In the section "Relationships to other metadata standards," hyperlinked names of standards take you to the Catalog page for that standard.
- Similarly, in the "Tools" section, the hyperlinked names take you to the Catalog page for the respective tool.

2.2 Contributing to the Catalog

Signing in

To make contributions to the Catalog, you must first sign in using the link in the top navigation bar (see Figure 2.4).

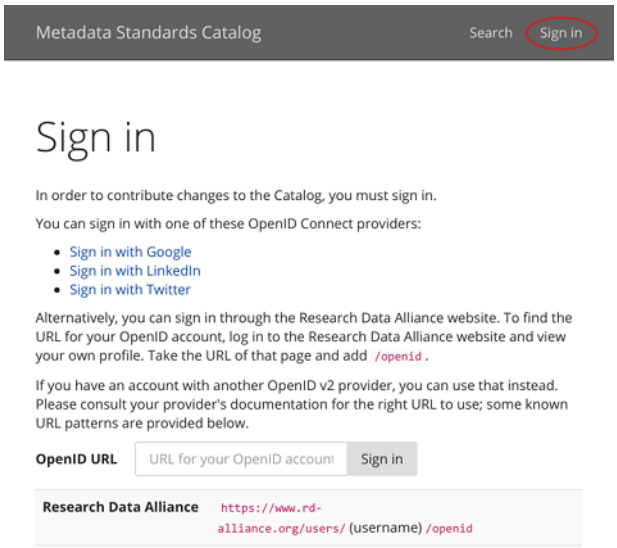

Figure 2.4: Page for signing in to the Catalog

The preferred way of signing in is via the RDA website.

1. Go to https://rd-alliance.org/user and sign in there.
2. On successfully signing in, you will be redirected to your profile page. In the address bar of your browser, you should see a URL like this: `https://rd-alliance.org/users/jane-doe`. The final portion (`jane-doe` in this case) is your OpenID username. Copy the whole URL.
3. On the Catalog "Sign in" page, paste the URL into the "OpenID URL" box and add `/openid` to the end, e.g., `https://rd-alliance.org/users/jane-doe/openid`.
4. Select the "Sign in" button.

The first time you do this, you will be redirected to the RDA website, where you will be asked if you want to use your credentials to log into the site. Once you have accepted, you be asked to set up your account for the Catalog. Your account details consist of your OpenID, and your name and email address. This is the minimum needed to

be able to track your contributions in the database log. Since these details will be permanently attached to any edits you make, it is recommended that you do not use a personal email address.

On subsequent occasions you will be redirected to the front page after signing in. You will see an additional link in the top navigation bar, "My Profile," which you can follow should you wish to update your name or email address.

The Catalog also has the ability to work with OpenID Connect providers, meaning you could sign in using a Google, LinkedIn or Twitter account. This ability is switched off currently in the live version to reduce the potential for spam accounts, but could be enabled if demand is sufficient.

Adding new records

On signing in, you will see an additional section on the front page headed "Make changes" (see Figure 2.5).

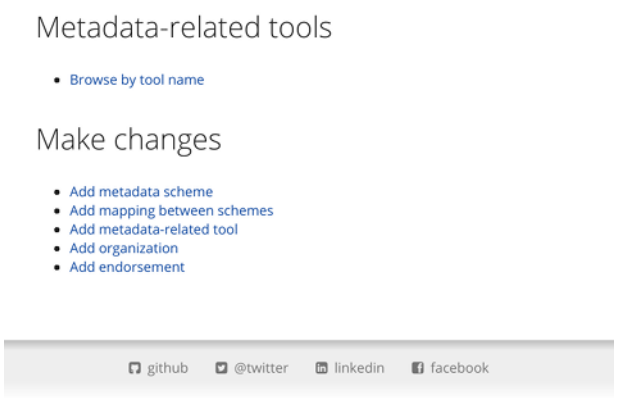

Figure 2.5: Front page of the Catalog after signing in

From here you can add new records to the database. If you have several records to add, it is most efficient do so in the following order, due to the directionality of the relationships between records:

1. organizations;
2. metadata schemes;
3. metadata-related tools, mappings and endorsements.

Editing existing records

To modify an existing record for a metadata scheme or related tool, navigate to the page corresponding to the record. At the end of the page, follow the link that says "Edit this record" (see Figure 2.6).

To modify the record for an organization, mapping or endorsement, first find a Catalog page (for a scheme or tool) where the entity appears. At the end of the set of links associated with it, you will find an "Edit record" link. The changes you make will (of course) be reflected everywhere that entity appears in the Catalog.

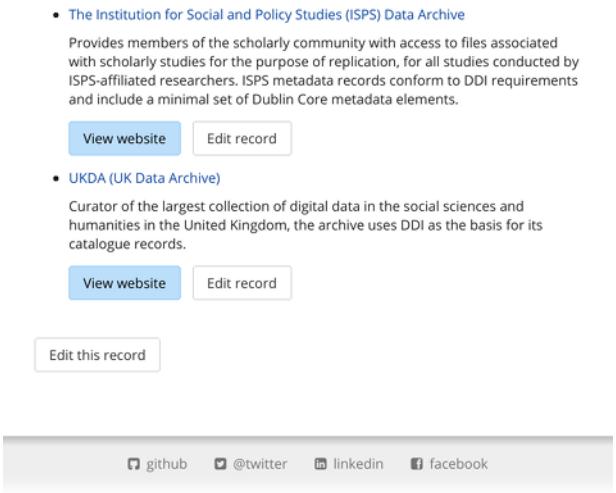

Figure 2.6: Links added to a Catalog page to allow editing

Using the editing forms

The forms for editing records are largely self-explanatory. Fill in as many fields as you can following the inline instructions.

If you need to link to a metadata scheme or organization that is not already in the Catalog, use the "Add scheme" or "Add organization" button provided to open up a new editing window and add it. You will need to reload your first editing page to see the new record.

The records for metadata schemes and tools allow you to provide more specific information for particular versions. Alongside each known version you will see a link "Add version-specific details." This will open up a new window which will be very like the first, but the versions section will be missing. You should only add information to this form that is unique to this particular version; this will prevent needless repetition on the parent page.

Some sets of fields can be repeated. You will see that one blank row is provided at a time. If you need to add several sets of information, save your changes after adding each one, and a new row will be added after each save. For this reason, the page does not automatically close or take you back to the finished Catalog page after saving, but instead allows you to continue editing.

Once you have saved all your changes, you can either close the window (if it was spawned during the editing process) or select "Cancel" to return to the normal Catalog page.

As part of the maintenance phase for the Catalog, we have plans to make this process smoother using AJAX (Asynchronous JavaScript and XML).

Signing out

You can sign back out of the Catalog at any time using the link in the top navigation bar.

2.3 Using the public API

Display one record

To retrieve a record in JSON, send a GET request to a URL formed from the internal MSC ID: replace the initial `msc:` with the URL of the Catalog followed by either

- `/msc/`, in which case you have to specify JSON format in the headers, e.g.,

```
curl -H 'Accept: application/json'
  ↪  https://rdamsc.bath.ac.uk/msc/m13
```

- `/api/`, in which case you get JSON automatically, e.g.,

```
curl https://rdamsc.bath.ac.uk/api/m13
```

The response you will get will be a JSON object that closely resembles the data structure used within the Catalog database itself. So, for example, a request for `msc:g22` might yield:

```
{
  "description": "A Web-based service for searching for and
    ↪  visualizing chemical structures. It uses data from the
    ↪  Protein Data Bank that has been transformed to RDF.",
  "identifiers": [
    {
      "id": "msc:g22",
      "scheme": "RDA-MSCWG"
    }
  ],
  "locations": [
    {
      "type": "website",
      "url": "http://xpdb.nist.gov/chemblast/pdb.pl"
    }
  ],
  "name": "Chem-BLAST",
  "slug": "chem-blast"
}
```

Unlike the HTML equivalent, it will not pull in information from related records in the database, so to get the full picture you will need to make multiple requests. We welcome feedback on whether to provide a method for retrieving a composite record in one go.

Search for records

To perform a search of the database, and receive a list of internal MSC IDs in return, use something like the following:

```
curl -X POST -F 'title=ABCD' https://rdamsc.bath.ac.uk/query/schemes
curl -X POST -F 'name=University'
  ↪ https://rdamsc.bath.ac.uk/query/organizations
curl -X POST -F 'supported_scheme=msc:m13'
  ↪ https://rdamsc.bath.ac.uk/query/tools
curl -X POST -F 'input_scheme=msc:m15' -F 'output_scheme=msc:m11'
  ↪ https://rdamsc.bath.ac.uk/query/mappings
curl -X POST -F 'endorsed_scheme=msc:m46'
  ↪ https://rdamsc.bath.ac.uk/query/endorsements
```

Unless otherwise stated, if you search using more than one field, the result set will be broadened. In other words, it will be as if you searched using each field individually then combined the results.

Supported queries when retrieving a list of **scheme** IDs:

- `title`: searches within the title using regular expression syntax.
- `keyword`: searches for an exact match for the given term, plus narrower and broader terms, within the list of keywords. To search for more than one keyword at once, separate the keywords with pipes, e.g., `keyword=Astronomy|Biology`.
- `keyword_id`: works similarly to `keyword`, but accepts one or more URIs from the UNESCO Vocabulary, separated by pipes. (You may notice a slight difference in behavior from `keyword` since several URIs map to the same keyword.)
- `identifier`: searches for an exact match within the list of identifiers. The primary use of this is to search for schemes by external identifier, though it can also be used to test if an internal ID is in use.
- `funder`: searches, using Python's regular expression[1] syntax, within the names of organizations listed as funders of the scheme.
- `funder_id`: searches for an exact match within the list of identifiers of organizations listed as funders of the scheme.
- `dataType`: searches for an exact match within the list of data types.

Supported queries when retrieving a list of **organization** IDs:

- `name`: searches within the name using regular expression syntax.
- `identifier`: works in the same way as for schemes.
- `type`: searches for organizations of the given type, drawn from the controlled vocabulary, i.e., `standards body`, `archive`, `professional group` or `coordination group`.

Supported queries when retrieving a list of **tool** IDs:

- `title`: searches within the title using regular expression syntax.
- `identifier`: works in the same way as for schemes.

[1] https://docs.python.org/3/library/re.html

- `type`: searches for tools of the given type, drawn from the controlled vocabulary, i.e., `terminal (`*<platform>*`)`, `graphical (`*<platform>*`)`, `web service`, or `web application`, where *<platform>* is one of `Windows`, `Mac OS X`, `Linux`, `BSD`, `cross-platform` or an equivalent term for another platform.
- `supported_scheme`: searches for tools that support the given scheme, expressed as an internal identifier.

Supported queries when retrieving a list of **mapping** IDs:

- `identifier`: works in the same way as for schemes.
- `input_scheme`: searches for mappings from the given scheme, expressed as an internal identifier. Contrary to how other fields work, the search will be narrowed if you also give an `output_scheme`.
- `output_scheme`: searches for mappings to the given scheme, expressed as an internal identifier. Contrary to how other fields work, the search will be narrowed if you also give an `input_scheme`.

Supported queries when retrieving a list of **endorsement** IDs:

- `identifier`: works in the same way as for schemes.
- `endorsed_scheme`: searches for endorsements of the given scheme, expressed as an internal identifier.

The response will be a JSON object, consisting of the key `ids` with an array as its value:

```
{
  "ids": [
    "msc:m1",
    "msc:m2"
  ]
}
```

List records

To get a list of all the records of a particular type, send a GET request to one of the following URLs:

- `https://rdamsc.bath.ac.uk/api/m` for metadata schemes
- `https://rdamsc.bath.ac.uk/api/g` for organizations (groups)
- `https://rdamsc.bath.ac.uk/api/t` for tools
- `https://rdamsc.bath.ac.uk/api/c` for mappings (crosswalks)
- `https://rdamsc.bath.ac.uk/api/e` for endorsements

The response will be a JSON object, consisting of a key representing the type of record (e.g., `metadata-schemes`) with an array of objects as its value. Each object has two keys:

- `id`: the MSC ID of the record.

- `slug`: a less opaque identifying string for the record, derived from the title of a metadata scheme, the name of an organization, etc. This string is used to generate file names when dumping the database to individual YAML files.

Example:


```
{
  "tools": [
    {
      "id": 1,
      "slug": "agrimetamaker"
    },
    {
      "id": 2,
      "slug": "anz-mest-metadata-entry-and-search-tool"
    },
    {
      "id": 3,
      "slug": "avedas-converis"
    }
  ]
}
```


List subject terms in use

To get a list of all the subject terms currently in use, send a GET request to `https://rdamsc.bath.ac.uk/api/subject-index`:

```
curl https://rdamsc.bath.ac.uk/api/subject-index
```

The result will be a list of JSON objects, each of which contains two or three keys:

- `name` provides the subject keyword itself.
- `url` provides the URL, relative to the base URL of the Catalog, of the Web page listing matching metadata schemes.
- `children`, if applicable, provides a list of child keywords, similarly represented as a list of JSON objects.

Example:


```
[
  {
    "name": "Multidisciplinary",
    "url": "/subject/Multidisciplinary"
  },
  {
    "name": "Culture",
    "url": "/subject/Culture",
    "children": [
```

```
      {
        "name": "Cultural policy and planning",
        "url": "/subject/Cultural+policy+and+planning",
        "children": [
          {
            "name": "Cultural heritage",
            "url": "/subject/Cultural+heritage"
          }
        ]
      },
      {
        "name": "Population",
        "url": "/subject/Population"
      }
    ]
  }
]
```

2.4 Using the restricted API

In order to use the restricted API, you will need to have your organization or application registered in the user database. API accounts must have a name, email address and password.

These accounts must be set up by an administrator, and cannot be added through the Web interface.

You should only use the restricted API over HTTPS, otherwise your username and password may be at risk of interception.

Change password

To change the password, use something like the following:

```
curl -u userid:password -X POST -H "Content-Type: application/json"
  ↪  -d '{"new_password": "your_new_password"}'
  ↪  https://rdamsc.bath.ac.uk/api/reset-password
```

If successful , the response will be a JSON object like this:

```
{
  "username": "your_username",
  "password_reset": "true"
}
```

Get token

To receive an authorization token, use something like the following:

```
curl -u userid:password -X GET https://rdamsc.bath.ac.uk/api/token
```

If authentication is successful, the response will be a JSON object, consisting of the key `token` and a long string as the value:

```
{
  "token": "the_actual_token_itself"
}
```

The token will be valid for 600 seconds.

Create a new record

To create a new record, send a POST request to one of the following URLs:

- `https://rdamsc.bath.ac.uk/api/m` for a metadata scheme
- `https://rdamsc.bath.ac.uk/api/g` for an organization
- `https://rdamsc.bath.ac.uk/api/t` for a tool
- `https://rdamsc.bath.ac.uk/api/c` for a mapping
- `https://rdamsc.bath.ac.uk/api/e` for an endorsement

The body of the request should be a JSON object representing the complete record; see the database documentation[2] for details. Say for example you wanted to register the following minimal record:

```
{
  "name": "Test group",
  "description": "This is a test.",
  "types": [
    "coordination group"
  ]
}
```

You would do so with a request such as this:

```
curl -u token:anything -X POST -H 'Content-Type: application/json' -d
  ↪  '{"name": "Test group", "description": "This is a test.",
  ↪  "types": [ "coordination group" ] }'
  ↪  https://rdamsc.bath.ac.uk/api/g
```

The response will be a JSON object, consisting of three keys:

- `success` indicates whether the record was created successfully.
- `conformance` indicates if the record was judged to be invalid, valid, useful, or complete, the definitions of which are given in the database documentation.[3]
- If the record was invalid, `errors` contains the reasons why the record was rejected, otherwise `id` contains the MSC ID of the new record.

[2] https://github.com/rd-alliance/metadata-catalog-dev/blob/master/db/README.md

[3] https://github.com/rd-alliance/metadata-catalog-dev/blob/master/db/README.md

Example of a successful response:

```
{
  "success": true,
  "conformance": "valid",
  "id": "msc:g99"
}
```

Example of an unsuccessful response:

```
{
  "success": false,
  "conformance": "invalid",
  "errors": {
    "locations": [
      {
        "type": [ "This field is required." ]
      }
    ]
  }
}
```

Modify an existing record

To modify a record, send a PUT request to `https://rdamsc.bath.ac.uk/api/` followed by the MSC ID, e.g., `https://rdamsc.bath.ac.uk/api/m1`.

The body of the request should be a JSON object representing the complete record; see the database documentation[4] for details. Example:

```
curl -u token:anything -X PUT -H 'Content-Type: application/json' -d
  ↪ '{"name": "Test group", "description": "This is a test.",
  ↪ "types": [ "coordination group" ] }'
  ↪ https://rdamsc.bath.ac.uk/api/g99
```

The response is the same as for "Create a new record."

Delete an existing record

To delete a record, send a DELETE request to `https://rdamsc.bath.ac.uk/api/` followed by the MSC ID, e.g., `https://rdamsc.bath.ac.uk/api/m1`. Example:

```
curl -u token:anything -X DELETE https://rdamsc.bath.ac.uk/api/g99
```

The response will be a JSON object, consisting of two keys:

- `success` indicates whether the record was deleted successfully.
- `id` contains the MSC ID of the deleted record.

[4] https://github.com/rd-alliance/metadata-catalog-dev/blob/master/db/README.md

```
{
  "success": true,
  "id": "msc:g99"
}
```

The ID will remain in the database as a null record, to prevent it being reused.

3

Developer Guide to the Catalog

If you would like to contribute to the development of the Catalog, it is recommended that you set up an environment where you will be able to run the code locally to test your changes.

3.1 Pre-requisite software

The Metadata Standards Catalog is written in Python 3,[1] so as a first step this will need to be installed on your machine. You will also need quite a few non-standard packages, but all of them are easily available via the `pip` utility:

- For the actual rendering of the pages you will need Flask,[2] Flask-WTF[3] (and hence WTForms[4]), and Flask-Login.[5]
- For Open ID v2.x login support, you will need Flask-OpenID.[6]
- For Open ID Connect (OAuth) support, you will need RAuth[7] (and hence Requests[8]), and Google's oauth2client.[9]
- For API authentication, you will need Flask-HTTPAuth[10] and PassLib.[11]
- For database capability, you will need TinyDB[12] v3.6.0+, tinyrecord,[13] and RDFLib.[14]

[1] https://www.python.org/
[2] http://flask.palletsprojects.com/
[3] https://flask-wtf.readthedocs.io/
[4] https://wtforms.readthedocs.io/
[5] https://flask-login.readthedocs.io/
[6] https://pythonhosted.org/Flask-OpenID/
[7] https://rauth.readthedocs.io/
[8] http://docs.python-requests.org/
[9] https://developers.google.com/api-client-library/python/guide/aaa_oauth
[10] https://flask-httpauth.readthedocs.io/
[11] https://passlib.readthedocs.io/
[12] http://tinydb.readthedocs.io/
[13] https://github.com/eugene-eeo/tinyrecord
[14] http://rdflib.readthedocs.io/

- For version control of the databases, you will need Dulwich.[15]
- To allow requests from JavaScript, you will need Flask-CORS.[16]
- For compiling the database from and backing it up to YAML files, you will need PyYAML[17]; this is not used by the Web application itself, only by the database control script.

For example, on a Debian-based Linux machine you could run commands like these:

```
sudo apt install python3-pip
sudo -H pip3 install flask Flask-WTF flask-login Flask-OpenID rauth
  ↪  oauth2client flask-httpauth passlib tinydb tinyrecord rdflib
  ↪  dulwich flask-cors pyyaml
```

The Catalog was written for Flask 0.10 but can be used with later versions.

You will also need Git[18] to handle the version control.

3.2 Installing a local instance

Navigate to the folder where you want to download the code. Use Git to checkout a copy of the repository (or your own GitHub fork of it), and enter it:

```
git clone https://github.com/rd-alliance/metadata-catalog-dev.git
cd metadata-catalog-dev
```

Now you need to set up a database for the Catalog to use by running the `dbctl.py` script. You can do this by running one of the following commands, depending on your operating system and Python installation:

```
./dbctl.py compile
python dbctl.py compile
python3 dbctl.py compile
```

For brevity, the remaining instructions here will assume the first form for similar commands. Do remember to convert them as necessary to the form that works.

What the above command does is take the files in the `db` folder and use them to generate a database file, `instance/data/db.json`. It will also set up the `instance/data` folder as its own Git repository, so that changes can be tracked.

One other thing you will probably want to do is switch the debugging feature back on (it is switched off in the default configuration). You can do this by saving a new file, `instance/keys.cfg`, with the following content:

```
DEBUG = True
```

[15] https://www.dulwich.io/

[16] http://flask-cors.readthedocs.io/

[17] https://pyyaml.org/

[18] https://git-scm.com/

You can now start the Catalog itself:

```
./serve.py
```

The feedback on the command line will tell you the URL to use in your browser. The Catalog will automatically reload when you make changes to the code.

3.3 Version control

As mentioned above, version control is handled by Git.[19] The `master` branch represents the live code; when the branch is updated, the live instance is notified and pulls in the changes.

It is therefore recommended to make your changes in a different branch or a forked repository, so you can fully test your code before pushing (or making a pull request) to the `master` branch.

3.4 Folder structure

The top level folder contains the following files:

- Documentation files in Markdown plain text format: `README.md` provides general information and instructions for using the API; `INSTALLATION.md` and `ADMINISTRATION.md` contain implementer guides for installing and administering the Catalog, respectively.
- The main Python 3 scripts, `serve.py` and `dbctl.py`.
- The UNESCO Thesaurus in RDF/Turtle plain text format, both in its official form, `unesco-thesaurus.ttl`, and the simplified version used by the Catalog, `simple-unesco-thesaurus.ttl`.
- Tools for migrating data from the Metadata Standards Directory to the Catalog, namely `migrate.py` and vocabulary mapping files `jacs2*.yml`.

The remaining files are arranged into folders as follows:

- `config` contains a Python 3 module with the default configuration options. Sample configurations are given for production and development use. It is recommended that you override at least the `SECRET_KEY` variable in your `instance/keys.cfg` file.
- `db` contains the Catalog database in YAML form. This folder was initially populated by migrating the data files from the Metadata Standards Directory. It will be refreshed periodically with dumps from the live Catalog database.
- `github_webhook` contains a local copy of python-github-webhook,[20] a Python 3 module for interacting with GitHub webhooks.
- `static` contains the style sheets, Web fonts, JavaScript files and application icon served verbatim by Flask.

[19] https://git-scm.com/

[20] https://bloomberg.github.io/python-github-webhook/

- `templates` contains the page templates used to generate the HTML pages. They are all written in the Jinja2 templating language.[21] The outer HTML page structure is in `base.html`. Macros for handling form errors are in `macros-forms.html`. The template for the index pages is in `contents.html`. All the others relate to specific pages.

3.5 Coding conventions

The Catalog is written in Python 3. The two main scripts are `serve.py`, which is the Web application, and `dbctl.py`, which is a command-line database control tool. There is another script, `migrate.py`, which was used to migrate data from the Metadata Standards Directory, but is no longer relevant.

The Python scripts are formatted according to PEP8 conventions. You can check conformance with these conventions with the `pycodestyle`[22] (formerly `pep8`) tool. Please run it with the default settings regarding which errors and warnings are enabled/disabled and the maximum line length.

3.6 Application: serve.py

To aid navigation through the file, sections and subsections are marked with underlined comments.

- **Dependencies:** The library import statements are grouped, with standard libraries first and non-standard ones second. Non-standard libraries should be annotated with a link to their documentation and a hint for how to install them (e.g., via `pip3`).
- **Customization:** This contains new and replacement classes required by the application:
  - `JSONStorageWithGit` is, as the name suggests, a version of TinyDB's `JSONStorage` that triggers a Git commit after each update, and uses the standard `json` library instead of `ujson`.
  - `User` is required by Flask-Login. `ApiUser` is the equivalent for API users.
  - `OAuthSignIn` and its subclasses `GoogleSignIn`, `LinkedinSignIn` and `TwitterSignIn` provide the functionality for signing in via OpenID Connect.
- **Basic setup:** This section contains the global variables used by the script. The first subsection contains the key objects (the app itself, the databases, authorization handler, webhook handler). The second implements key parts of the data model, such as table names and controlled vocabularies.
- **Utility functions:** These functions and classes do not handle requests directly but are used by several request-handling functions.

[21] http://jinja.palletsprojects.com/

[22] https://pycodestyle.readthedocs.io/

- **Non-interactive pages:** These functions handle requests for non-interactive pages.
  - Front page
  - Terms of use
  - Display (or return as JSON) a record for a metadata scheme or tool
  - Display a per-subject list of metadata schemes
  - Display a per-data-type list of metadata schemes
  - Display a per-funder/maintainer/user list of metadata schemes
  - Display the index of metadata schemes
  - Display the list of tools
  - Display the index of subject terms
- **Form utilities:** Most of these functions help out with the translation between the internal data representation of records and the data structure used by WTForms in the HTML forms. There is also a function for generating selection option lists from the database.
- **Interactive pages:** The HTML forms and API query equivalents are defined by classes that extend either `Form` (for read-only operations) or `FlaskForm` (for write operations).
  - Search/query forms for schemes (HTML and API) and organizations, tools, mappings and endorsements (API only). This section also contains some helper functions specific to these forms and their handlers.
  - User accounts. Forms and handler functions for signing in and out, and creating, editing and deleting profiles.
  - API authentication routes and functions.
  - Forms for editing the five different record types. The section is arranged with specialist fields and sub-forms representing repeated groups of fields first, following by the main forms themselves, followed by the route handler. There are also helper functions for ensuring consistency of data-type labels/URLs across the database, and removing duplicate error messages.
  - API route handling functions for create, update and delete operations.
  - API route handling functions for retrieving records, lists of records and the index of subject terms.
- **Automatic self-updating:** This contains the function that pulls changes from the `master` branch on GitHub whenever commits are pushed to the code repository.

The request-handling functions can be easily distinguished by the `@app.route()` decorator. The syntax for these route decorations can be found in the Werkzeug documentation.[23]

Regarding naming conventions, class names and WTForms validator functions use `CamelCase` while all other variables and functions use `snake_case`.

[23] http://werkzeug.palletsprojects.com/

3.7 Database control: dbctl.py

The structure of this file should be self-explanatory. Sections and subsections are marked with underlined comments.

- **Dependencies:** The library import statements are grouped, with standard libraries first and non-standard ones second. Non-standard libraries should be annotated with a link to their documentation and a hint for how to install them (e.g., via `pip3`).
- **Utility definitions:** The first of these is for a class that overrides the default storage used by TinyDB. The issue is that TinyDB uses `ujson` by default if available for its speed, but `ujson` does not write JSON to disk in a consistent order. This making version control of those files highly inefficient, and the diff logs rather noisy. This class definition enforces use of the standard `json` library which can sort keys consistently.

  The second definition provides a handler for serializing date objects to JSON as ISO formatted strings.
- **Initializing:** This section contains the global variables used by the script, including the definitions of the command-line argument parser and sub-parsers.
- **Operations:** In this section, each of the functions of the script is defined in turn. The names of the main functions start with a capital letter, prefixed by `db`; after the definition the function is associated with the matching subparser. If the function relies on helper functions, these are defined immediately before; the naming convention for helper functions is `snake_case`. Helper functions should also have a docstring (a triple-quoted string at the very start explaining what the function does).
- **Processing:** This contains the `main` function that runs when the script is executed.

4

Impact statement

The MSC is the continuation of Metadata Standards Directory and DCC Disciplinary Metadata Catalogue, so it is anticipated that users and adopters of the earlier incarnations will migrate over to the new Catalog in time. The following are examples of impact specific to the Catalog:

- MSC has been adopted by RDA as the output from MSCWG and currently is being used actively. As discussed earlier in the report, it will be used as the data source, and possibly the interface, for a tool proposed by the Metadata Interest Group for interconverting between arbitrary standards.
- The API is used to incorporate content from the MSC into the DCC's DMPonline tool.[1]
- The MSC and its API are used by the FAIRsFAIR project.[2]
- The MSC has been used as a research resource by postgraduate students at the University of Victoria and Karlsruhe Institute of Technology.

[1] https://dmponline.dcc.ac.uk/

[2] https://www.fairsfair.eu/

5

Sustainability plan

The MSC has been designed deliberately to be easy to use and to require a minimum of systems maintenance. Full documentation has been provided on how to install development and production instances, in order to aid ongoing maintenance and allow easy transfer between hosts. This documentation has already been tested in the transfer of the live instance from the Digital Curation Centre's servers to those of the University of Bath.

The University of Bath has committed to hosting the MSC for the foreseeable future, for which the co-chairs of the group would like to express their gratitude.

The MSC database is backed up periodically to GitHub so as to guard against data loss, enable development instances to use a snapshot of the live data, and facilitate any future platform migrations.

The Metadata Standards Catalog Working Group will continue in maintenance mode with oversight from the Metadata Interest Group. The group will further develop the code base in the light of feature and security improvements in its software dependencies. Several possible improvements to the user experience, API, and data model have already been identified; the group aims to implement these changes in a new release of the MSC by the end of 2020.

The Metadata Interest Group is working to build a community of contributors to the MSC, similar to and seeded from the one the Metadata Standards Directory Working Group achieved for the precursor Directory. In this way, the records in the MSC will be kept up to date with a high degree of coverage.

6

Works cited

Ball, A., Greenberg, J, Jeffery, K., and Koskela, R., *RDA Metadata Standards Directory Working Group: Final Report* (Research Data Alliance, 2016), https://rd-alliance.org/group/metadata-standards-catalog-wg/outcomes/metadata-standards-directory-wg-recommedations.html

Tenopir, C., Allard, S., Douglass, K., Aydinoglu, A. U., Wu, L., Read, E., Manoff, M., and Frame, M., "Data Sharing by Scientists: Practices and Perceptions," *PLoS ONE* 6/6 (2011): e21101. doi: 10.1371/journal.pone.0021101

Willis, C., Greenberg, J., and White, H., "Analysis and Synthesis of Metadata Goals for Scientific Data," *Journal of the American Society for Information Science and Technology* 63/8 (2012): 1505–1520. doi: 10.1002/asi.22683