# Spec White Paper

Metadata Templates

## Abstract

This document outlines the addition to iRODS of *metadata templates*. In particular, we present the rationale behind and some use cases driving the development of metadata templates, we discuss the design of metadata template functionality, and we present a draft JSON schema and a reference implementation of metadata templates using the iRODS Java client library, Jargon.

## Use Cases and Design Considerations

Metadata templates represent a significant addition of functionality to iRODS's handling of metadata. In base iRODS, metadata is stored in the catalog as unadorned attribute-value-unit (AVU) triples of strings. This is flexible for many use cases, but does not afford the grid administrators much control of any stored metadata. By extending iRODS metadata, specifically by associating the AVU triples stored in the catalog with metadata templates which can provide additional information *about* these metadata elements (or meta-metadata), we can now support a variety of interesting functions.

iRODS metadata templates allow administrators and curators to:

- View and interact with AVUs in a *user-friendly* interface
- *Require* specified metadata elements to be associated with iRODS objects
- *Validate* metadata elements
- *Standardize* the metadata associated with types or collections of iRODS objects
- *Uniformly apply* metadata to many iRODS objects simultaneously
- Provide *rendering guidance* regarding objects or metadata to iRODS GUIs

Metadata templates, therefore, benefit many classes of iRODS users. A domain scientist or a research group can create a template that stores common metadata AVUs (such as creator name, creator contact information, funding source, etc.), enabling these common metadata to be quickly, easily, and uniformly applied to many iRODS objects. An iRODS administrator or curator can create a template that requires and validates specific metadata, ensuring that iRODS objects in a particular location or of a particular type have valid and consistently-applied

metadata. A front-end developer can use information from metadata templates to inform how to organize and render metadata or iRODS objects themselves.

# Draft JSON Schema

The following draft JSON schema describes the data format of an iRODS metadata template. (Note that while this schema provides default values for some template and element properties, the JSON schema specification does not actually require that defaults are enforced, since doing so would require modifying the data instance. In the Jargon implementation of metadata templates discussed in the next section, these default values are enforced in the Java code. Any other implementation of metadata templates would also require special consideration for handling default values.)

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
    "title":"Metadata template",
    "description":"JSON schema defining an iRODS metadata template",
    "type":"object",
    "properties":{
        "name":{
            "description":"The name of the template, e.g. 'dublinCore'",
            "type":"string"
        },
        "type":{
            "description":"The type of the template, i.e. FORM_BASED",
            "type":"string",
            "enum":[
                "FORM_BASED"
            ],
            "default":"FORM_BASED"
        },
        "source":{
            "description":"Where the data to populate the template will come from, i.e.
'USER'",
            "type":"string",
            "enum":[
                "USER"
            ],
            "default":"USER"
        },
        "destination":{
            "description":"How the filled-out metadata will be stored, i.e. 'IRODS'",
            "type":"string",
            "enum":[
                "IRODS"
            ],
            "default":"IRODS"
        },
        "description":{
```

```
            "description":"Helpful text about the specific metadata template",
            "type":"string"
        },
        "author":{
            "description":"The creator of the template, e.g. 'John Smith'",
            "type":"string"
        },
        "version":{
            "description":"The version of the template, e.g. '1.0'",
            "type":"string"
        },
        "required":{
            "description":"Whether this template is required to be applied",
            "type":"boolean",
            "default":false
        },
        "elements":{
            "description":"A list of MetadataElements that make up the template",
            "type":"array",
            "items":{
                "type":"object",
                "properties":{
                    "name":{
                        "description":"The name of the metadata element (should be unique per
template)",
                        "type":"string"
                    },
                    "i18nName":{
                        "description":"The internationalized name of the metadata element
(should be unique per template)",
                        "type":"string"
                    },
                    "description":{
                        "description":"Helpful text about the metadata element",
                        "type":"string"
                    },
                    "i18nDescription":{
                        "description":"The internationalized help text",
                        "type":"string"
                    },
                    "type":{
                        "description":"The data type for this element",
                        "type":"string",
                        "enum":[
                            "RAW_STRING",
                            "RAW_TEXT",
                            "RAW_URL",
                            "RAW_INT",
                            "RAW_FLOAT",
                            "RAW_BOOLEAN",
                            "RAW_DATE",
                            "RAW_TIME",
                            "RAW_DATETIME",
                            "REF_IRODS_QUERY",
                            "REF_IRODS_CATALOG",
                            "REF_URL",
                            "LIST_STRING",
```

```
                            "LIST_INT",
                            "LIST_FLOAT",
                        ],
                        "default":"RAW_STRING"
                    },
                    "source":{
                        "description":"Where the data to populate the element will come from,
i.e. 'USER'",
                        "type":"string",
                        "enum":[
                            "USER"
                        ],
                        "default":"USER"
                    },
                    "defaultValue":{
                        "description":"The default value of this element",
                        "type":"string"
                    },
                    "validationOptions":{
                        "description":"Options for validation, used in combination with
validationStyle",
                        "type":"array",
                        "items":{
                            "type":"string"
                        }
                    },
                    "validationStyle":{
                        "description":"Validation style for this element, used in combination
with validationOptions",
                        "type":"string",
                        "enum":[
                            "DEFAULT",
                            "IS",
                            "IN_LIST",
                            "IN_RANGE",
                            "IN_RANGE_EXCLUSIVE",
                            "REGEX",
                            "FOLLOW_REF",
                            "DO_NOT_VALIDATE"
                        ],
                        "default":"DEFAULT"
                    },
                    "required":{
                        "description":"Whether this element is required to be populated",
                        "type":"boolean",
                        "default":false
                    }
                },
                "required":[
                    "name",
                    "type",
                    "source",
                    "validationStyle",
                    "required"
                ]
            }
        }
```

```
    },
    "required":[
        "name",
        "type",
        "source",
        "destination",
        "required"
    ]
}
```

## Discussion of the JSON Schema

In addition to the properties discussed above, Metadata Templates and Metadata Elements also have associated universally unique identifiers (UUIDs). These are used to find templates and to associated individual AVUs with Metadata Templates in the `getAndMergeTemplateListForPath` function, discussed in greater detail below. The unit string of an AVU that is governed by a Metadata Template will contain, for example, `fromTemplate:01234567-01234-01234-01234-0123456789ab`.

Several of the properties on the Metadata Template object require explanation. The `type` property is an enum that indicates what type of metadata template this is. Right now, only `FORM_BASED` is a valid value, indicating that a list of metadata elements will displayed for the user in, for example, an HTML form, and that these elements will be listed in this metadata template as the `elements` property. We have also discussed the possibility of automatically generating a template from, for example, a schema.org schema (perhaps `SCHEMA_REF`).

The `source` property is an enum that indicates where the data that will be used to populate a metadata template comes from. Right now, only `USER` is a valid value, indicating that metadata values will be populated manually by a user. We have given some thought to allowing metadata to be generated or harvested from an iRODS rule (perhaps `RULE`) or a combination of the two (`MIXED`), but these represent future work.

The `destination` property is an enum that indicates how the actual metadata AVUs will be stored on disk. Right now, only `IRODS` is a valid value, indicating that the metadata will be stored as AVU triples in the iRODS metadata catalog. This property was included with the intention of enabling users who wish to use a separate metadata management database to do so while still taking advantage of the functionality of metadata templates. For example, CyVerse (formerly the iPlant Collaborative) uses an external PostgreSQL database to manage metadata, so perhaps `POSTGRES` would be added to the enum, for example.

The `required` property is a boolean flag that indicates whether or not this template *must* be applied to all collections in which it is visible. If `required` is true, this template must be populated.

Several properties of the Metadata Element object also require additional explanation. The `type` property is an enum that indicates what type element this is, and is used for both validation and display. At present, a Metadata Element can be of the following types: `RAW_STRING`, `RAW_TEXT`, `RAW_URL`, `RAW_INT`, `RAW_FLOAT`, `RAW_BOOLEAN`, `RAW_DATE`, `RAW_TIME`, `RAW_DATETIME`, `REF_IRODS_QUERY`, `REF_IRODS_CATALOG`, `REF_URL`, `LIST_STRING`, `LIST_INT`, `LIST_FLOAT`. The `RAW_` types are those whose values will be stored directly in the metadata catalog. `RAW_STRING`, `RAW_TEXT`, and `RAW_URL` are all treated as strings; the different types are used to suggest to the UI how the form will be rendered (`RAW_TEXT` might be rendered as a multi-line text input box, for example.) The `REF_` types indicate that the metadata value of interest might reside elsewhere, as a webpage, or an image file stored in iRODS storage, or a property of a data object stored in the catalog. `REF_IRODS_QUERY` is used to access properties of a data object, collection, or user (similar to what might be returned from a GenQuery), and `REF_IRODS_CATALOG` is used to specify the logical path to a file stored in iRODS. The `LIST_` types are used to store multiple metadata values with the same attribute name on an object, for example, if an object has multiple authors.

The `source` property is an enum that indicates where the data that will be used to populate a metadata element comes from. This is simply a more fine-grained application of the `source` property described above regarding Metadata Template objects. Right now, only `USER` is a valid value, indicating that metadata will be populated manually by a user.

The `validationOptions` and `validationStyle` properties are used in combination to define what validation, if any, will be performed on the value of this metadata element. `validationStyle` is an enum that can take on the following values: `DEFAULT`, `IS`, `IN_LIST`, `IN_RANGE`, `IN_RANGE_EXCLUSIVE`, `REGEX`, `FOLLOW_REF`, `DO_NOT_VALIDATE`. `DEFAULT` validation checks that the value is non-empty and of the type specified in `type`, and does not make use of `validationOptions`. `IS` validation checks that the value is the same as the one(s) in `validationOptions`. `IN_LIST` validation checks to see that the value is one of a specified list of possible values; the possible values are in the `validationOptions` array, and there can be an arbitrary number of them. `IN_RANGE` validation checks to see that the value is in between two values (inclusive); the two values are in the `validationOptions` array. (If there are more than two options, only the first two are used, and if there are fewer than two, no validation is performed.) `IN_RANGE_EXCLUSIVE` validation works the same, except values equal to either endpoint

are not valid. `REGEX` validation tests to see if the entered value matches the regular expression specified in the first element of `validationOptions`. `FOLLOW_REF` validation applies only to `REF_` types, and checks to see if the reference can possibly be valid (if the iRODS query is valid, if the iRODS object exists, or if the URL can be resolved).

# Reference Implementation Discussion

## Metadata Template File Locations

For the Jargon reference implementation, it is assumed that the metadata template files themselves reside in the iRODS Zone in the `.irods` collections that are also used for storing data related to other Jargon extensions, including starred files and virtual collections. This makes locating templates that can apply to a particular iRODS object a nicely constrained problem. Whenever one makes a call to the Metadata Template Service to request available templates, the service checks the `.irods` collections in the collection hierarchy above the object, as well as any special "public" collections (which must be specifically indicated to the Metadata Template service by calling `setPublicTemplateLocations`).

## Metadata Template Service

The Metadata Template Service has four major components:

1. Parser
2. Validator
3. Resolver
4. Exporter

In Jargon, these are implemented as separate classes.

### Metadata Template Parser

The Metadata Template Parser handles the creation of Metadata Template objects from JSON as well as the exporting of Metadata Template objects to JSON files.

The Metadata Template Parser should implement (at least) the following functions:

- `createMetadataTemplateFromJSON(String json ...)`
  Creates a MetadataTemplate object given a JSON definition of one.

- `createJSONFromMetadataTemplate(MetadataTemplate mt ...)`
  Creates a JSON definition from an existing MetadataTemplate object.

## Metadata Template Validator

The Metadata Template Validator handles the validation of Metadata Templates and Metadata Elements, as described in the previous section.

The Metadata Template Validator should implement (at least) the following functions:

- `validate(MetadataTemplate mt ...)`
  Validate a MetadataTemplate.
- `validate(MetadataElement me ...)`
  Validate a MetadataElement based on its `currentValue`, `validationOptions`, and `validationStyle`.

## Metadata Template Resolver

The Metadata Template Resolver handles the various file system calls relating to Metadata Template files (list, find, create, rename, update, delete), as well as the very important merge operation, which populates a Metadata Template object with the AVUs already associated with an iRODS object.

The Metadata Template Resolver should implement (at least) the following functions:

- `getPublicTemplateLocations(...)`
- `setPublicTemplateLocations(List<String> locations ...)`
  Specify which iRODS collections will be searched for public templates.

- `listTemplatesInDirectoryHierarchyAbovePath(String path ...)`
- `listPublicTemplates(...)`
- `listAllTemplates(String path ...)`
- `listRequiredTemplates(String path)`
  Retrieve lists of metadata templates.

- `findTemplateByName(String name, String path ...)`
- `findTemplateByNameInDirectoryHierarchy(String name, String path ...)`
- `findTemplateByNameInPublicTemplates(String name ...)`
- `findTemplateByFqName(String fqName ...)`
- `findTemplateByUUID(String uuid ...)`

Retrieve a single metadata template by name or unique identifier.

- `saveMetadataTemplateAsJSON(MetadataTemplate mt, String path ...)`

  Given a populated metadata template object, save it out to a JSON file

- `renameTemplateByFqName(String fqName, String newFqName ...)`
- `renameTemplateByUUID(String uuid, String newFqName, ...)`

  Change the name of a specified template

- `cloneTemplateByFqName(String fqName, String newTemplateName, String path ...)`
- `cloneTemplateByUUID(String uuid, String newTemplateName, String path ...)`

  Create a copy (with a new name) of a specified template

- `deleteTemplateByFqName(String fqName ...)`
- `deleteTemplateByUUID(String uuid ...)`

  Delete a specified template

- `getAndMergeTemplateListForPath(String objectPath ...)`

  Create the list of Metadata Templates that can apply to a given object, and populate their Metadata Elements from the AVUs on the object.

## `getAndMergeTemplateListForPath`

Since the merge function is so important to the functionality of Metadata Templates, it may benefit from a more extended discussion. Given the path to a collection or data object, `getAndMergeTemplateListForPath` generates a list of all Metadata Templates that could apply to that object (searching both in public template locations and in the directory hierarchy above the object), and populates those Metadata Templates with any metadata values found on the object. Any AVUs which do not match any of the available Metadata Templates are stored in a separate list of AVUs and are returned alongside the populated list of templates. So the caller of `getAndMergeTemplateListForPath` receives a List of Metadata Templates, populated with any AVUs that were found to match the templates, as well as a List of “orphaned” AVU objects that are not (yet) affiliated with any Metadata Template.

### Metadata Template Exporter

The Metadata Template Exporter handles the saving of populated Metadata Template files out to the permanent metadata store. For example, in Jargon/iRODS, the JargonMetadataExporter saves the populated Metadata Template out to a set of specially formatted AVUs.

- `saveTemplateToSystemMetadataOnObject(MetadataTemplate mt, String pathToObject)`
  Save the values in the Metadata Template with the object in the system metadata table
- `saveElementToSystemMetadataOnObject(MetadataElement me, String pathToObject)`
  Save the value in the Metadata Element with the object in the system metadata table

# Illustration of system in use