FIMS documentation Documentation

Release 1.0

John Deck, RJ Ewing

Jun 16, 2021

User Information

All source code mentioned in this documentation is open source and freely available and can be found in appropriate repositories living under the Biocode, LLC GitHub Organization.

CHAPTER 1

Introduction

GEOME is used for data validation, expedition planning, and data management for field-based surveys enabling tracking physical objects including organisms, soil cores, water samples, and sub-samples. If you would like to start your own GEOME instance, you can either download and install the relevant modules (all freely available) or contact the owner of the GEOME installation code site to see if you can be added as a project to this installation.

CHAPTER 2

User Guide to Creating Local Identifiers

A crucial part of the GEOME is converting local identifiers that you construct and use in your own research, and turning these into globally unique, resolvable identifiers. Globally unique identifiers are created by appending your local identifier onto a unique root that is generated for every resource within every expedition. Examples of locally unique identifiers are "Grinnell1213", "MooreaEvent2", or "MBIO56_1".

Each identifier that is minted will be resolvable via HTTP using California Digital Library's Name-to-thing resolver. Since the name-to-thing resolver is sensitive to certain characters, we have limited the characters that are suitable for use as local identifiers. Allowable characters are validated on data load so if you choose an invalid character you will get an error message. The following are the allowed local identifier characters:

- A-Z
- a-z
- 0-9
- – (plus)
- = (equals)
- : (colon)
- . (period)
- _ (underscore)
- ((open parantheses)
-) (close parantheses)
- ~ (tilde)
- * (asterisk)

The following are valid identifiers: "MVZ:Herp:1234", "Grinnell (1234)"

The following would be invalid identifiers: "MVZ-Herp-1234", "Grinnell/Alexander 1234"

Once data is made loaded and made public, you can search for your newly minted globally unique and resolvable identifiers in the Query page, and they will be listed under the "BCID" column. If the identifier is shown as

“ark:/21547/R2MBIO564” you can substitute “http://n2t.net/ark:” for the “ark:” to make a a resolvable identifier as **‘https://n2t.net/ark:/21547/CXs2MBIO564‘_**, where MBIO564 is the locally uinque identifier.

CHAPTER 3

GEOME Queries

GEOME provides a custom sql-like query syntax to help you find the data you need. The following documentation supplements the Swagger Applicaiton Programming Interface.

By default, the query terms are executed against all columns in the project. To execute a query against a specific column, you can construct the query in the form `columnName:query`.

The *full text search* query `japan` would return all results where a column contains the word `japan`. Where as the *full text search* query `column1:japan` would return all results where **column1** contains the word `japan`.

All queries can be constructed using the sql operators *AND*, *OR*, and *NOT* as well as groupings within *()*;

The query `_expeditions_:myExpedition and not japan` would return all results in the *expedition* `myExpedition` which do not contain the word `japan`.

Below you will find more information about the supported queries.

3.1 Supported Queries

The following queries are supported:

- *full text search*
- *comparison*
- *project*
- **'expedition'_**
- **'exists'_**
- *like*
- *phrase*
- *range*
- *select*

3.1.1 full text search

This the default query, and will perform a search on the *tokenized* version of the uploaded data.

`col1:value` - will perform a fts on `col1` for `value` `col1:val*` - will perform a fts on `col1` for words starting with `val` `value` - will preform a fts on all columns for `value`

3.1.2 comparison

This query is used to compare 2 values. The following operators are supported:

NOTE: for correct comparison results when using <, <=, >, >=, the Attribute dataType should be one of (Integer, Float, Date, Datetime, Time). This can be set via the project configuration. Talk to your project administrator about this.

= - equals <> - not equals > - greater then >= - greater then or equal to < - less then <= - less then or equal to

3.1.3 project query

This query is will filter the results based on the project(s) that they belong to.

The query `_projects_:1` would return everything uploaded under project `1` The query `_projects_:[1, 2]` would return everything uploaded under project `1` or `2`

3.1.4 expedition query

This query is will filter the results based on the expedition(s) that they belong to. Note: as expeditions are only unique within a project, you most likely want to specify a project query as well.

The query `_expeditions_:myExpedition` would return everything uploaded under `myExpedition` The query `_expeditions_:[myExpedition1, myExpedition2]` would return everything uploaded under `myExpedition1` or `myExpedition2`

3.1.5 _exists_ query

This query returns results where a column has a value.

The query `_exists_:column1` would return all results where `column1` has a value. The query `_exists_:[column1, column2]` would return all results where `column1` or `column2` has a value.

3.1.6 like query

This query performs a sql `ILIKE` (case-insensitive `LIKE`) query.

```
col1:"%value" - col1 ILIKE '%value'
```

3.1.7 phrase query

This query performs a sql `ILIKE` (case-insensitive `LIKE`) query.

```
col1:"some value" - col1 ILIKE '%some value%'
```

3.1.8 range query

This is a shorthanded way to perform a comparison query.

NOTE: for correct comparison results, the Attribute dataType should be one of (Integer, Float, Date, Datetime, Time) This can be set via the project configuration. Talk to your project adminstrator about this.

`col1:[1 TO 10]` - `>= 1 AND <= 10` `col1:[1 TO 10}` - `>= 1 AND < 10` `col1:{1 TO 10}` - `> 1 AND < 10` `col1:{* TO 100]` - `<= 100`

3.1.9 select query

Used to select related parent/child data along with the queried entity. The provided value should be the conceptAlias of the Entity to select. The provided conceptAlias' do need to be related to the query entity, but do not need to be directly related. For example, if you are querying a parent entity, you can also select the grandChildren and the grandParents. Any combination of related entities can be selected.

NOTE: `_select` queries should not be preceded/followed by *and* or *or* keywords and can not be preceded by the *not* keyword.

`_select_:parentEntity` - selects both child and parent entity results for the query `_select_:[parentEntity, grandParentEntity]` - selects both child and parent entity results for the query

3.2 Tokenization

Text fields go through a tokenization process before they are indexed. This process attempts to breakdown text into words and numbers as well as converting words to their normalized form.

Tokenization Ex:

```
"many donkeys" -> ["many", "donkey"]
```

For more information, you can view the psql tokenization.

CHAPTER 4

Installation

This content is for people wishing to install GEOME on their own server.

4.1 Details

GEOME consists of a core set of Java classes and REST services. Developers have a choice of interacting with the REST services running _BCID, which has built in EZID minting capabilities, or running their own instance of _BCID and installing their own EZID instance requiring a purchase of an EZID account.

To run an instance of FIMS you will need the following components:

- A unix-based server * A java servlet container e.g. Tomcat, Glassfish, Jetty * Connection to a BCID service

4.2 Installation and Build – Migrating an existing installation

- Source code is available on this site via github
- Building is done via an Gradle build file (provided as part of the distribution)
- a properties file needs to be configured by copying biocode-fims.template to biocode-fims.props (in the root directory of the distribution)

Install the following software

- postgres
- jetty9
- java8
- bcid and geome-db repositories (from github)

Properties file

- update properties files in *src/main/environment/production*

gradle war deploy build/libs/geome-db.war (do the above for both bcid and geome-db)

generate openapi document using *gradle resolve*

CHAPTER 5

Configuration Files

GEOME has a network level configuratino file which defines network level rules and all available data properties and entities. Each project has its own configuration file as well which supplements the GEOME network configuration file. All configuration files are written in JSON This is where the projects specific configuration is specified. This includes resources, attributes, validation rules, and relations.

5.1 Attributes

5.1.1 DataType

Each attribute may specify a `dataType`. A dataType can be specified to provide additional validation, and in the case of date, datetime, and time, can be used for data formatting. This is especially helpful for standardizing the data to aid in querying and analysis.

The following `dataType` are supported:

- String (default if not specified)
- Integer
- Float
- Date
  - must specify `dataformat` as well
- Time
  - must specify `dataformat` as well
- Datetime
  - must specify `dataformat` as well

CHAPTER 6

Record

Fims validation and upload is based around the concept of a *Record*. A *Record* is a single instance of an *Entity*.

A *Record* is typically a k:v map of properties. The key should be the columnUri. It is the responsibility of the *DataReader* implementation to map any columnName -> columnUri when creating an instance of a *Record*.

Each type of *Record* will have a *RecordValidator* implementation that is responsible for handling the validation of that *Record* type. The default *Record* type is a *GenericRecord*. A *GenericRecord* is the most common in the fims system will be. The validation for a *GenericRecord* is strictly controlled by the project configuration w/o any additional validation logic.

currently we have support for the following types:

- *GenericRecord*
- *FastaRecord*
- *FastqRecord*
- *PhotoRecord*

CHAPTER 7

RecordSet

A collection of *Record* instances.

CHAPTER 8

Dataset

A collection of *RecordSet* instances. If a *Dataset* has any *RecordSet's for a child 'Entity*, then the *Dataset* will contain the both the parent and child *RecordSet's. The 'DatasetBuilder* should be used to help construct a valid *Dataset* instance.

8.1 Data Readers

DataReader implementations contain the logic for reading and converting a specific file type into a RecordSet (TODO: more info about RecordSets). When a file is uploaded for validation, it is passed to the *DataReaderFactory* which will return the appropriate *DataReader* implementation for the provided file. A *DataReader* should return true when *handlesExtension* is called if that reader can handle the provided ext.

A current limitation of DataReaders is that if multiple *DataReader* implementations handle the same file ext, only 1 can be enabled at a given time. This restriction may be lifted in the future.

TODO more info about current DataReader implementations

8.2 Entity

Custom entities can be created and must subclass the *Entity* class. All subclasses must exist in the *biocode.fims.digester* package to be properly registered as a valid subtype for polymorphic serialization/deserialization via Jackson. An Entity subclass provides the ability to fix certain parts of a given entity, as well as provide additional validation logic (to be executed on ProjectConfig updates) to ensure the entity is well formed and not missing any pertinant information.

CHAPTER 9

REST Services

FIMS REST Services are available at: http://www.biscicol.org/apidocs/

9.1 Versioning

FIMS REST Services are now versioned. v1 is the default version. You may specify the version by including the header:

```
Api-Version:  {version}
```

or via the url:

```
http://biscicol.org/biocode-fims/rest/{version}/...
```

We currently support the following versions:

- v1
- v1.1

more info about the specific version resources to come...

CHAPTER 10

User Accounts

User accounts are not required to lookup/resolve BCIDs. However, they are required to work with projects, expeditions, or create new BCIDs. Here we describe how to obtain a user account for Biocode-

10.1 Account Creation

User accounts can be created by either by the Biocode-Fims instance owner or by project administrators. Project administrators can add any existing user in the Biocode-Fims system as an authorized expedition creator. Talk to your project administrator to be added to a particular project.

[https://github.com/biocodellc/biocode-fims-commons/wiki/OAuth2 Information about Open Authorization]

10.2 Project Administrators

Project administrators are set by the Biocode-Fims instance owner upon request. There is only one designated project administrator per project. The project administrator can add, create, and remove users, set the location of the validation XML file, and define the project abstract.

CHAPTER 11

curl Examples

Plenty of curl examples are available at our Swagger Application Programming Interface documentation at:

https://api.geome-db.org/apidocs/

CHAPTER 12

oauth2

All developers need to register their app. Please contact the system admin to register. You will be issued a client_id and client_secret. The client_secret should be kept private.

12.1 Authorization

Client app will make a GET request to /id/authenticationService/oauth/authorize. This request will contain the following query

- client_id (Required) - The client_id your app was issued during when registered.
- redirect_uri (Required) - The absolute URI you would like the response directed to.
- state (Optional) - Will be returned, unmodified, in the response.

The response will contain the following query parameters:

- code - The random 20 character string used to exchange for an access_token. This code expires in 10 mins and can only be used 1 time.
- state - Only if this parameter was included in the request.

12.2 Access Token

Client app will make a POST request to /id/authenticationService/oauth/access_token. This request will contain the following pa

- client_id (Required) - The client_id your app was issued during when registered.
- client_secret (Required) - The client_secret your app was issued during when registered.
- code (Required) - The authorization code received in the authorization request.
- redirect_uri (Required) - The absolute URI you would like the response directed to. Must be identical to the redirect_uri provided in the authorization request.

- state (Optional) - Will be returned, unmodified, in the response.
- grant_type (Optional) - If grant_type is "password", and a username and password is provided, the username and password will be used for authentication. If authentication is successful, an access_token and refresh_token will be returned
- password (Optional) - Required if grant_type is "password".
- username (Optional) - Required if grant_type is "password".

The JSON response will contain the following parameters:

- access_token - The random 20 character string used to access a user's profile.
- refresh_token - The random 20 character string used to obtain a new access_token. This expires after 24 hrs.
- token_type - currently we only issue bearer tokens.
- expires_in - the number of seconds the token is good for.
- state - Only if this parameter was included in the request.

12.3 Refresh Token

Client app will make a POST request to /id/authenticationService/oauth/refresh. This request will contain the following parame

- client_id (Required) - The client_id your app was issued during when registered.
- client_secret (Required) - The client_secret your app was issued during when registered.
- refresh_token (Required) - The refresh_token you were issued with you access token.

The server will validate the refresh token and if the refresh token is less then 24 hrs old, a new access token will be issued. The current refresh token will be expired and a new one will be issued.

The JSON response will contain the following parameters:

- access_token - The random 20 character string used to access a user's profile.
- refresh_token - The random 20 character string used to obtain a new access_token. This expires after 24 hrs.
- token_type - currently we only issue bearer tokens.
- expires_in - the number of seconds the token is good for.

12.4 API Access

In order to obtain a user's profile information, make a GET request to /id/userService/profile with the access_token as a query parameter.

If the token is still valid, you will receive a JSON response with the following user information:

- firstName
- lastName
- email
- institution

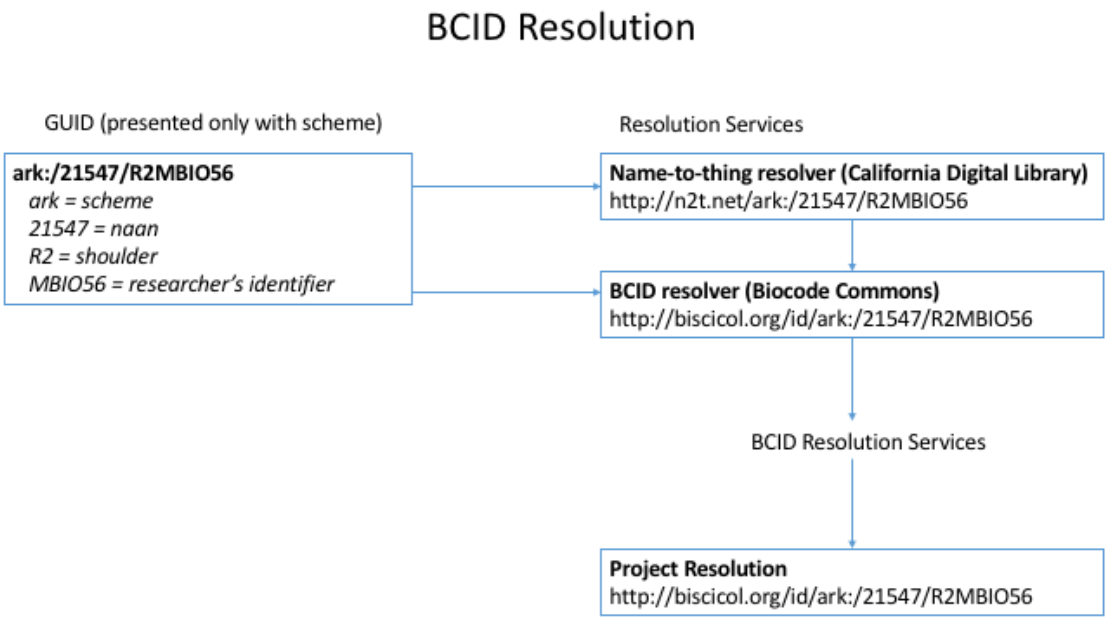
- userId
- username
- projectAdmin
- hasSetPassword

We also support access to any rest services on behalf of the user. Just append "?access_token=your_access_token" to the url in order to access the service.

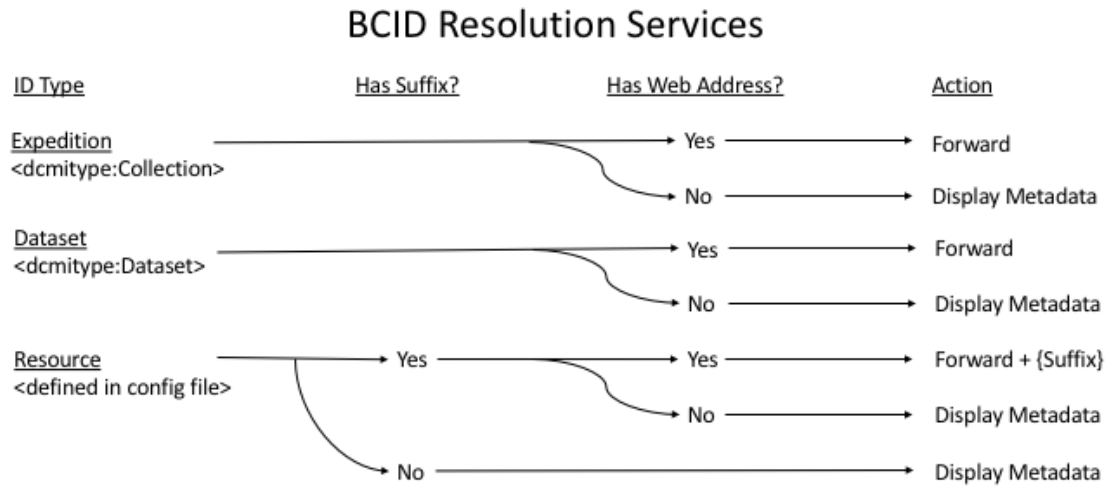
CHAPTER 13

Resolution System

The following illustration shows how BCIDs work with local identifiers, the world wide web, and EZID's name-to-thing resolution service. A field researcher uses their own numbering system (e.g. 'MBIO56'), and uploads their data to FIMS, which assigns it to a resource category (e.g. 'R2'). The FIMS system itself is registered under the ark: scheme, and has a name assigning authority number (NAAN) of 21547. Resolution requests coming through name-to-thing are re-directed to the BCID resolution service.



The following chart shows how BCID resolution works for expeditions, datasets, and resources in the FIMS system with actions falling under forwarding, or metadata display. Forwarding behaviour is determined by either the specification of a target webaddress in the database, or absent that, a specification in the project's configuration file.



Forward Logic:

```
If (bcid.webaddress != null) return bcid.webaddress;   // From database
else {
   If (ID Type = Expedition) return <metadataParam.expeditionForwardingAddress>{ark};
   else if (ID Type != Dataset) return metadataParam.conceptForwardingAddress/{ark}/{suffix};
   else return "Display Metadata Address"
}
```

Uses apache strSubstituor

CHAPTER 14

Types Of Identifiers

FIMS uses a centralized minting service to assign identifiers for three types of identifiers: expeditions, datasets, and resources. The three types of identifiers are described below.

Each FIMS system installation must use its own name assigning authority number and register with California Digital Library's EZID service to mint Archival Resource Keys (ARKs).

14.1 Expedition identifiers

- resourceType: http://purl.org/dc/dcmitype/Collection
- Mutable, representing the most current version of a particular spreadsheet
- **Metadata:**
  - expeditionCode
  - expeditionTitle
  - userId (who created this expedition)
  - ts (when loaded)
  - projectId (project this belongs to)
  - public (public or not)

14.2 Dataset identifiers

- resourceType: http://purl.org/dc/dcmitype/Dataset
- Immutable
- Belongs to a specific expedition
- **Metadata:**

  - webAddress (where this dataset can be found, in its native format, depending on installation)
  - userId (who uploaded this dataset)
  - doi (an optional doi, in addition to the created ARK)

14.3 Resource identifiers

- resourceType: defined in configuration file
- Belongs to an expedition. Multiple resources may be specified for each expedition.
- Implements suffix-passthrough feature to identify individual resources within each dataset. For example, a single "Material Sample" identifier is created for each expedition. If the expedition has 1000 rows representing physical samples, 1000 identifiers can be resolved by appending a locally unique suffix on to the Resource Identifier root.
- A resource identifier plus the locally unique primary key loaded for the most recent dataset in an expedition forms the globally unique identifier for a particular resource.