TERBINE API OVERVIEW

Posted: June 4, 2018
Version 0.9.8 (draft)

TABLE OF CONTENTS

CONTENTS

Contents

OVERVIEW

This document is a high level overview of the TERBINE Application Programming Interface (API). This document is not intended to be an exhaustive list of available interfaces or services, but rather a smooth introduction to any client who intends to use the API to interface with the TERBINE marketplace services.

Referenced Documents

This API Overview is built on information contained within the IoT Metadata Specification. In addition, there is example code and a complimentary document title TERBINE API Tutorial that has working examples of using the TERBINE API.

API CATEGORIES

The TERBINE API can be broken down into the following categories.

- Organization / User Administration
- Organization / User Profile and Preferences
- Search
- Metadata / Schema Administration
- Technical (Upload/Download)

SECURITY

API security will be implemented using a combination of a security strategy Java Web Token (JWT) for authentication. When a user successfully logs in they receive a TOKEN back from the server with a limited lifetime. This token is exclusive to that user, expires after a period of time and must be passed in to all subsequent calls.

Role based security based on pre-defined TERBINE based roles will provide authorization to functionality and resources.

In addition wire based security will be implemented using SSL/TLS.

As needed, disk based encryption will provide “data at rest” based security.
All Personally Identified Information (PII) will be stored in an encrypted format and salted as needed. TERBINE policy is to avoid storage of any PII related data.

TERBINE also will support machine to machine (M2M) interaction and identify for this will be accomplished with a combination of provisioned API Key Identifier and Public Key.

The majority of the API will be secured excluding basic browsing, login and system overview services.

SERVICE MODEL

The TERBINE API will be implemented using a REST model built on HTTP(s).
Within the REST model the system is divided into a group of resources (users, metadata, etc.) and each is uniquely identified. Generally, but not exclusively, resources use a Universally Unique Identifier (UUID) which is a 128 bit value. An example of a UUID is **`fb9967f1-eadf-465f-b11e-cbeda2c62bdf`**. These will be allocated and managed by the TERBINE backend.

HTTP VERBS

REST built on HTTP(s) uses standard HTTP verbs for managing resources.

- GET – read a resource by id or group of resources by name.
- PUT – update a resource
- POST – create a new resource
- DELETE – delete a resource
- PATCH – this will have limited use and is used when a portion of a resource will be updated
-

EXAMPLES

GET

- *GET* `https://api.terbine.io/user/`fb9967f1-eadf-465f-b11e-cbeda2c62bdf
- *GET* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf/securityanswers`
- *GET* `https://api.terbine.io/metadata/sample`

POST

- *POST* `https://api.terbine.io/user`
- *POST* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf/securityanswers`

PUT

- *PUT* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf`
- *PUT* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf /metadata/eb9967f2-eadf-465f-b11e-cbeda2c62bdf`

DELETE

- *DELETE* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf`
- *DELETE* `https://api.terbine.io/user/fb9967f1-eadf-465f-b11e-cbeda2c62bdf /metadata/eb9967f2-eadf-465f-b11e-cbeda2c62bdf`

RETURN AND ERROR CODES

HTTP Verb	CRUD Operation	Entire Collection (e.g. /users)	Specific Item (e.g. /users/{id})
GET	read	200 (OK), list of users. Use pagination, sorting and filtering to navigate big lists.	200 (OK), single user, 404 (Not Found), if ID not found or invalid.
POST	create	201 (Created) Request Body with mandatory information for all resources. Response with new resources, ids and associated data.	201 (Created) Request Body with mandatory resource information. Response with new resource, id and associated data.
PUT	update	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid for all resources in body	200 (OK) or 204 (No Content). 404 (Not Found), if ID not found or invalid for specific resource
DELETE	delete	404 (Not Found), unless implemented delete the whole collection—not normally desirable.	200 (OK). 404 (Not Found), if ID not found or invalid for resource

As a general rule success of any operation can be checked by looking for a return code in the range of 200 – 299.

Any return code in the 300 – 399 range will denote a moved resource and will be used sparingly and be documented thoroughly at the specific API level.

Any return code in the 400 range denotes a client error. Most common examples of these are:

- 400 – Bad request. The correct method was used, but there was missing information in the body or malformed entity being passed.
- 401 – Unauthorized. User needs to be authenticated and use a correct API Key / Session token.
- 403 – User is Not Authorized. User not allowed to carry out this operation.

- 404 – Not Found. Resource with this ID not found.
- 405 – Method Not Allowed. A resource was called with a form not allowed (e.g. correct form for a GET but not a POST).

Server errors will be denoted by HTTP status codes in the 500 range. Most common are:

- 500 – Internal Server Error – unexpected error at server level
- 503 – Service Unavailable – API or service not available.

Optionally, specific information may be passed back detailing what is missing or why the error occurred.

HEADER INFORMATION

All REST calls will require multiple header values be set.

Within TERBINE we have multiple common header values that may or may not be required dependent on the value. The majority of the services are protected by requiring an **Authorization** header.

There is also a possibility to set a header value from client as **X-TRACKING-ID.** This id will be used for all logging on the server for that request. The use of this would be if there is any client logging, the client would create a new id for each request made to the server and in this way logging entries from client through to server and back in response can be linked by this id. The tracking id is very useful for user error reporting and troubleshooting.

If the client does not pass a tracking id, one will be created on the server and passed back in the response header.

DATA EXCHANGE FORMAT

All data is interchanged in the API via JSON. JSON is a simple text-based message format that is often used with RESTful Web services.

JSON EXAMPLE

```
{
 "id": "98931ef4-c423-42f5-869c-88eae5b39dfa",
 "orgId": "6d94ff88-4b41-41d0-b3a4-d14d89da0939",
```

```
 "identifier": {
   "extId": "A634321"
 },
 "meta": {
   "name": "Medical Test Metadata",
   "description": "Medical Test Metadata Description",
   "version": "v1"
 },
 "dataset": {
   "extId": "SRC_EXT_ID1",
   "type": 30,
   "sensorInfo": {
     "id": null,
     "type": 44,
     "make": "make 123",
     "model": "model 123",
     "comment": "Specific sensor comment"
   },
   "schemaInfo": {
     "format": 6,
     "type": 81,
     "property": ";",
     "body": null
   },
   "comment": "Comment on sample dataset information"
 },
 "containers": [
   {
     "type": 130,
     "parentId": null,
     "extId": "CONT_EXT_ID",
     "legalType": null,
     "latitude": "40.078521",
     "longitude": "-74.078974",
     "altitude": "N/A",
     "address": "zip:08742",
     "startDate": "2015-12-03T04:17:36Z",
     "endDate": null,
     "name": "Container name",
     "description": "Container long description"
   }
 ],
 "deliveries": [
   {
     "type": 22,
     "size": 32032223232,
     "hash": null,
     "deliveryDate": "2015-12-03T04:17:36Z"
   }
 ],
 "owners": [
   {
     "startDate": "2015-12-03T04:17:36Z",
     "endDate": null,
     "id": "67b181ee-eff0-4066-9629-badb071e9aff",
     "type": 60,
     "comment": "Ownership comment here"
   }
 ],
```

```
 "categories": [
    {
      "name": "Utilities",
      "id": 3
    },
    {
      "name": "Information",
      "id": 9
    }
 ],
 "tags": [
    {
      "name": "Voltage",
      "id": 0
    },
    {
      "name": "Sensor",
      "id": 0
    }
 ],
 "legal": [
    {
      "type": 102,
      "comment": "Legal Information for COPYRIGHT",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    },
    {
      "type": 101,
      "comment": "Legal Information for GOVERNMENT",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    }
 ],
 "regulatory": [
    {
      "type": 141,
      "comment": "Regulatory Information for SAFEHARBOR",
      "startDate": "2015-12-03T04:17:36Z",
      "endDate": null
    }
 ],
 "relations": [
    {
      "type": 112,
      "referenceId": "574c1a6a-34a6-4575-94cb-26f0c4c376bb",
      "comment": "Relationship Sample",
      "startDate": null,
      "endDate": null
    }
 ],
 "gradings": [
    {
      "type": 120,
      "grade": "Gold",
      "comment": "Sample grading information"
    }
 ]
}
```

EXPLANATION OF COMMENTS

1. JSON in the TERBINE API is not wrapped (type indicator at the root of the data).
2. Identifiers are case insensitive in the API. So orgId and orgid both work.
3. UUID – an example of a UUID value is `6d94ff88-4b41-41d0-b3a4-d14d89da0939`, these will be widely used within TERBINE as unique identifiers for resources.
4. Standard date format – this is an example of a standard date format, ISO 8601 format and transmitted in UTC. It is up to client to display these in any localized format. Date only format will be YYYY-MM-DD.
5. Embedded (Children) data – this is sub entity of the owning item **metadata** called **schema**. It has a child relationship to the parent. Dependent on context it may also live as a top level resource.
6. Null Values – this is how non-existent values are transmitted. If passed in a PUT or POST they will overwrite existing value with the NULL value.
7. Embedded Quotes – this shows a string with embedded quotes and therefore the \ character is preceding them.
8. Array of Embedded Objects – When there is a 0 to N (or 1 to N) the embedded objects are placed in an array as shown here.
9. Array of Single Data Items – When there is a list of single items they are embedded in an array as shown here.

CLIENT DEPENDENCIES

TERBINE API is built completely client agnostic. All API interfaces have neither client specific information nor variations of the definitions dependent on client type
Therefore a Web client, iOS or Android client should find the use of the TERBINE API seamless. For analytics purposes see below.

DEVICE CATEGORIES

TERBINE API allows passing of device info as part of API. This is done through the X-DEVICE-HDR header. Note we do not use the user-agent header for this purpose.

Following categories are supported:

- WEB
- ANDROID
- ANDROID_TABLET
- IOS_IPHONE
- IOS_IPAD

Note as of writing only WEB is supported.

AUTHORIZATION

LOGIN

POST

https://api.terbine.io/api/auth/login

Header

Content-Type: application/json

With post body with user id and password

```
{
 "username": "<user>",
 "password": "<password>"

}
```

Response


```
{
    "lastlogin": "2017-08-28T21:44:48Z",
    "username": "user name",
    "displayname": "user display name",
    "token":
    "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXJiaW5lYWRtaW4iLCJleHAiOjE1MDM5NjI1MzV9.z0KDmCubJmio7JsFGK8TE4yQyuzMGKCOLZMcrmJV_ME",
    "userid": "aab864d2-74fc-44c1-b0d3-632829719929",
    "orgid": "6d94ff99-4b41-42d0-b3a4-d14d89da0838",
    "orgname": "company name",
    "tokenlifeseconds": 4200,
    "numberNotices": 1,
    "tabs": [
        "ACCOUNT",
        "SUMMARY",
        "FINANCE",
```

```
        "LEGAL",
        "WORKSPACE",
        "METADATA",
        "TECHNICAL",
        "DATASETS",
        "POLICIES"
    ],
}
```

The field token should be taken from the login response and added to all subsequent calls as header such as:

```
Authorization:  bearer
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ0ZXJiaW5lYWRtaW4iLCJleHAiOj
E1MDM5NjI1MzV9.z0KDmCubJmio7JsFGK8TE4yQyuzMGKCOLZMcrmJV_ME
```

Clients should also note the **userid** and **orgid** fields in the response as these are often needed for specific services. These are system wide unique identifiers for the login user and the user's organization.

SEARCH

SEARCH SERVICE

/api/search/v2/metadata?pageNum=1&pageSize=5&sort=DATEADDED&order=DESC

```
{
 "text": "ambient",
 "filters": {
  "grade": [
   "bronze",
   "silver"
  ],
  "legal": [
   "109"
  ],
  "categories": [
   "1008"
  ]
 }
}
```

Response


```
[
  {
    "metadata": null,
    "type": 2,
    "source": "Organization",
    "id": "90189158-8362-4df5-b147-f97b10151dfc",
    "orgId": "893528a7-5d5c-45ee-87a9-e35bf0635e27",
    "ownerOrgId": null,
    "alphaSort": "Vehicle-sampled Ambient Air Temperature / United States
Nationwide",
    "title": "Vehicle-sampled Ambient Air Temperature / United States
Nationwide",
    "description": "Timestamped & geolocated samplings of outside air
temperature on vehicles equipped with the Voyomotive On Board Diagnostics (OBD-
II standard) capture / transceiver device. Fleet sampling of over 500 vehicles
in various geographies. \n",
    "dateIndexed": "2017-08-28T01:13:58Z",
    "dateAdded": "2017-07-01T18:52:13Z",
    "createDate": "2017-07-01T18:52:13Z",
    "categories": [
      "1008",
      "1009",
```

```
      "1021"
    ],
    "ownership": [
      "22"
    ],
    "dataset": [
      "30"
    ],
    "sensor": [
      "41"
    ],
    "format": [
      "502"
    ],
    "environment": [
      "60"
    ],
    "container": [
      "70"
    ],
    "schema": [
      "81"
    ],
    "legal": [
      "109"
    ],
    "relation": [
      "113"
    ],
    "grade": [
      "SILVER"
    ],
    "datagrade": [
      "121"
    ],
    "location": [
      "131"
    ],
    "locationSub": [
      "10005"
    ],
    "address": [
      " US"
    ],
    "locationOther": null,
    "coordinate": {
      "lon": "",
      "lat": "",
      "alt": null
    },
    "regulatory": [
      "149"
    ],
    "customListEntryId": null,
    "customListEntryName": null
  }
]
```

Please note that the search service does not require authentication. Unauthenticated calls will result in any metadata not covered by a policy.

Authenticated search will return all metadata owned by the user's organization, public metadata owned by other organizations and metadata from other organizations allowed by covered policy.

METADATA

DOMAIN TYPES

```
Request
GET
/api/domain/type
```

Response
JSON array with each type of domain that can be used within metadata.

DOMAIN VALUES

```
Request
GET
/api/domain/<id>

Parameter <id> is the one the ids returned from
/api/domain/type
Example (returns sensor type values)
/api/domain/4
```

Response
JSON with each vzlue for passed type

METADATA SERVICES

GET ALL METADATA FOR ORGANIZATION

```
Request
GET
/api/metadata

Request is paginated so takes pageNum and pageSize query
parameters.
```
Response
JSON array of metadata objects.

GET METADATA BY ID

```
Request
GET
/api/metadata/<id>

Example
/api/metadata/711225d5-3f21-4ddc-a332-62699d319681.
```

Response
JSON representation of a single metadata object.

INSERT METADATA

```
Request
POST
/api/metadata

Example
/api/metadata

Sample POST body
```

Response
JSON representation of new inserted metadata object with IDs created.

UPDATE METADATA

```
Request
PUT
/api/metadata

Example
/api/metadata

Sample PUT body
```

Response
JSON representation of newly updated metadata object.

DELETE METADATA

```
Request
DELETE
/api/metadata

Example
/api/metadata
```

Example:

```
 DELETE /api/metadata/711225d5-3f21-4ddc-a332-62699d319681
```

```
Response
HTTP 200 with message metadata was disabled.
```

CLONE METADATA

```
Request
POST
/api/metadata/<id>/clone

Example
/api/metadata
```

Example:

```
 POST /api/metadata/711225d5-3f21-4ddc-a332-
62699d319681/clone
```

```
Response
HTTP 200 with newly created (cloned) metadata
```

GENERAL NOTES

Pagination

Paginated requests allow passing these items as query parameters:

- pageNum = page to be returns, starting from 1
- pageSize = size of returned result set
- orderBy = column to sort by
- order = ASC or DESC

DATASET

OVERVIEW

Provides API description for Dataset and Dataset Content (files).

The first part of understanding dataset and content is explaining terminology.
A **dataset** is a description (avoiding use of word metadata) of the physical content that is uploaded for a given **metadata** configuration. The id field in the dataset tab is the **"dataset id"**. The **metadata** owns the dataset. A dataset can be **inbound** or **outbound**. These terms are always used in relation to the user, so inbound is purchased datasets and outbound is provided datasets.

A metadata configuration, per the data model, can have more than one dataset. A dataset then has multiple files. A dataset can be thought of as a container for a group of same structured files.

The actual physical data might be addressed with any of these terms (**content, file, attachment**). This is still a bit nebulous as we explore different use cases within the system (e.g. streaming data, external storage, etc).

For our case, when a user enters metadata it also adds a dataset as an outbound type.

SERVICE SUMMARY

GET LIST OF FILES FOR DATASET

Request
GET
/api/dataset/{datasetid}/content
Response
Array of content description with information about each file. Note the ID in this response is what goes into various following calls as "contented", highlighted below. (Items in green might be used for display)


```
[
  {
      "id": "e72ab85d-bc44-4b41-9c5b-0c6f0ae54efc",
      "metadataId": "d2fb8dd5-9151-40a0-a794-3a35bf893103",
      "datasetId": "b233d348-963a-45b4-a04b-c46becb846c0",
      "externalId": "be-292922432",
```

```
        "organizationId": "6d94ff88-4b41-41d0-b3a4-d14d89da0939",
        "type": 1,
        "status": 1,
        "fileStoreLocation": "6d94ff88-4b41-41d0-b3a4-d14d89da0939/d2fb8dd5-9151-
40a0-a794-3a35bf893103/UItzTNVeTy7AKZt4AerYv2Ru.terbine.json",
        "fileExt": "json",
        "size": 22742,
        "fileHash": "b6aa9e273e8de94948a582b78f33207b6b9890b4",
        "lastModifyTime": "2017-06-26T20:42:46Z",
        "dateUploaded": "2017-06-26T20:42:45Z",
      "dateStart": "2017-06-24T17:45:40Z",
        "dateEnd": null,
        "deleted": 0,
        "originalName": "voyo-2017-04-20-18-30-01.json"
    },
```


…. rest of response omitted.

UPLOAD SINGLE FILE

Request

```
POST
Content-Type – application/x-www-form-urlencoded
```

/api/dataset/{datasetid}/content/upload

Request Body

File info is sent as file

Below info is sent as form key Info

Example form part for “info” key:

```
{
  "fileName": "filetoupload.csv",
  "externalId": "be-232323",
  "dataStart": "2017-06-24T21:45:40Z",
  "dataEnd": "2017-06-26T21:45:40Z",
}
```

```
Note all except filename are optional, though dataStart is
recommended as is when data is valid.

Response
Returns a record with all information related to the file,
including its new unique ID.

{
```

```
      "id": "0d5d81c0-dc77-4d85-8bd7-09462df1dfb8",
    "metadataId": "bca81462-e586-4fa4-ab72-37d3306b93ee",
      "datasetId": "47bce565-36c6-46e4-956a-c7cd14e9520a",
    "externalId":
"766f796f2d323031372d30342d32302d31382d33302d30312e6a736f6e",
    "organizationId": "6d94ff88-4b41-41d0-b3a4-d14d89da0939",
      "type": 1,
      "status": 1,
    "fileStoreLocation": "6d94ff88-4b41-41d0-b3a4-
d14d89da0939/bca81462-e586-4fa4-ab72-
37d3306b93ee/bLp0GuBvFnMJJISiAq7cU3lu.terbine.json",
      "fileExt": "json",
      "size": 22742,
      "fileHash": "b6aa9e273e8de94948a582b78f33207b6b9890b4",
      "lastModifyTime": "2017-06-29T22:33:39Z",
    "dateUploaded": "2017-06-29T22:33:21Z",
      "dateStart": "2017-06-24T21:45:40Z",
      "dateEnd": null,
      "deleted": 0,
    "originalName": "voyo-2017-04-20-18-30-01.json"
}
```


DOWNLOAD SINGLE FILE

Request
GET
/api/dataset/content/{contentid}/download

Response

File to download streamed to client.

DOWNLOAD MULTIPLE FILES

This will download, in zip file, number of files specified by the called, if the query parameter "maximum" is not provided it defaults to 50. The files will be in descending order the most recent. Zip file will have current date and time as name.
(as we understand more, we will expand this with parameters such as date ranges and embedded values)

Request
GET
/api/dataset/{datasetid}/content/download?maximum=10

Response
Zip file containing files

Download Single File for Preview

This is similar to download, only a length parameter can be included to allow only downloading a certain number of bytes. If the length query parameter is not provided, it defaults to a value of 1024.

This will only work for text files currently, and not types such as PDF, XLS, XLSX etc.

Request
GET
/api/dataset/content/{contentid}/download/preview?length=1024

Response
Portion of File to download streamed to client. If file is smaller than requested length all of it will download.

Delete Single Item

Request
DELETE
/api/dataset/content/{contentid}

Response
Message if deletion was successful.

Delete All Items in Dataset

Request
DELETE
/api/dataset/{datasetid}/content

Response
Message if deletion was successful.

REVISIONS

Version	Date	Name	Description
Initiated	2015/09/06	Brian Enochson	For Review
0.9	2016/01/20	Brian Enochson	Initial Internal Release
0.9.1	2017/08/28	Brian Enochson	Updated for search
0.9.2	2017/09/21	Brian Enochson	
0.9.3	2017/10/08	Brian Enochson	
0.9.5	2017/11/18	Brian Enochson	Updated for Pre-Release
0.9.6	2017/11/25	Brian Enochson	Updates relating to domain and dataset API. References added to API Tutorial document. Section re-ordering.
0.9.7	2018/02/26	Brian Enochson	Modification to reflect latest API updates.
0.9.8	2018/06/04	Brian Enochson	Updated to match latest metadata specification.

TERBINE API TUTORIAL

Posted: June 4, 2018
Version 0.9.8 (draft)

TABLE OF CONTENTS

CONTENTS

Contents

OVERVIEW

This document is a tutorial of a real world type application for using the TERBINE Application Programming Interface (API). This is to be used in conjunction with the TERBINE API Overview document where detailed information to relevant services is found.

WORKFLOW

TERBINE has a defined workflow to how data is processed. This is based on how specific roles would process data, either inbound outbound, and the decision tree for supplying or obtaining data within a standard enterprise.

Before we begin there are several terms that need to be understood.

- Metadata
- Dataset
- Member
- Organization
- Member
- Workspace
- Technical Tab
- API
- Json Web Token (JWT)

PREREQUISITES

- Account Enabled – this can be created within the TERBINE Web App or by contacting TERBINE directly.
- Admin User Enabled and a working user and password that has been provided by TERBINE.
- Java 8, Maven, GIT Command Line installed and working.
- Knowledge of REST services, JSON, JWT and intermediate Java assumed.
- Knowledge of IOT Metadata Specification.

WHAT WE WILL COVER

- Domain data
- Create Metadata Configuration
- Searching Metadata
- Filtering Searched Metadata
- Getting information on available datasets
- Upload a dataset
- Download a dataset

EXAMPLE CODE

The code can be accessed in our public Github repo at https://github.com/Terbine/examplejava. This is a Java Maven based project.

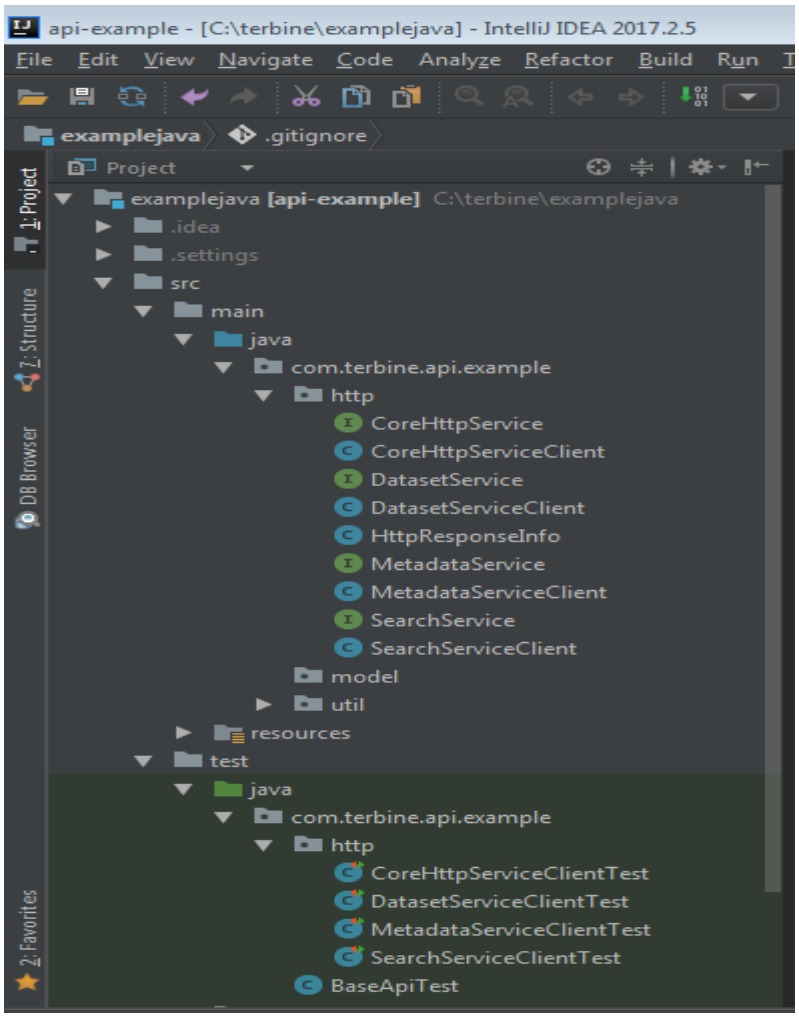
For instance go to where you want the source code to be placed and run the following command.

```
$ git clone https://github.com/Terbine/examplejava.git
```

To enable the tests the following configuration must be done.

Go to file src/test/resources/unit_test.properties and replace the placeholders with values provided by TERBINE.

Below is a screen shot of the sample project opened within IntelliJ Community Edition.



CODE TOUR

AUTHENTICATION SERVICES

For a login example go to the file CoreHttpServiceClient.java and CoreHttpServiceClientTest.java.

This assumes the user, password and baseUri have been set in unit_test.properties.

A JSON string representing the an **AuthenticatedUser** is returned. To call any services requiring an authenticated user the value in the **token** field that is returned must be sent as an `authorization: bearer <token>` HTTP Header.

So in the response the owning organization identifier (as UUID) is also returned. This is in the **AuthenticatedUser** json in response from login in the **orgid** field. This will be needed in some services when manipulating data.

In the file **CoreServiceHttpClientTest.java** the following tests refer to domain data.

- testLogin

DOMAIN DATA SERVICES

For domain data example go to the file CoreHttpServiceClient.java and CoreHttpServiceClientTest.java.

This assumes the user, password and baseUri have been set in unit_test.properties.

Domain data within TERBINE is any data that is used as reference to standard or industry specific codes and to classify data, via the metadata configuration. Normally these are some sort of numeric or alphanumeric code and an associated description. TERBINE uses these to identify schema types, sensor types, legal types etc.

These need to be placed into the respective id field when inserting or updating a new metadata configuration.

For an example of the use of these ID's within a metadata configuration refer to the `/src/test/resources/sample/insertmetadata.json` file.

In the file **CoreServiceHttpClientTest.java** the following tests refer to domain data.

- testGetDomainTypes
- testGetDomainById
- testGetDomainByString

- testGetDomainSic
- testGetDomainGic
- testGetCountries
- testGetStates

SEARCH SERVICES

For search examples go to the file SearchServiceClient.java and SearchServiceClientTest.java.

This assumes the user, password and baseUri have been set in unit_test.properties.

Whenever metadata is created or modified the contents of that is indexed and becomes instantly searchable. This allows ease of discovery for metadata of specific categories or types.

TERBINE provides services to search and filter metadata instances.

Search services are paginated meaning they allow specifying a page number, page size as well as sort field and sort order. Within TERBINE services that potentially return a large number of results are generally paginated.

In the file **SearchServiceClientTest.java** the following tests refer to search services.

- testBasicSearch
- testBasicSearchWithFilter

METADATA SERVICES

For metadata examples go to the file MetadataServiceClient.java and MetadataServiceClientTest.java.

This assumes the user, password and baseUri have been set in unit_test.propertie
Metadata is the core of any interaction within TERBINE. It describes the data that is stored or referenced by TERBINE and allows searching and discovery of the data.

For our example the JSON for a new metadata configuration in the `/src/test/resources/sample/insertmetadata.json` file will be used.

To use this you will want to modify at least 3 field.

- **orgId** : this field would be replaced with the UUID returned from the login request.

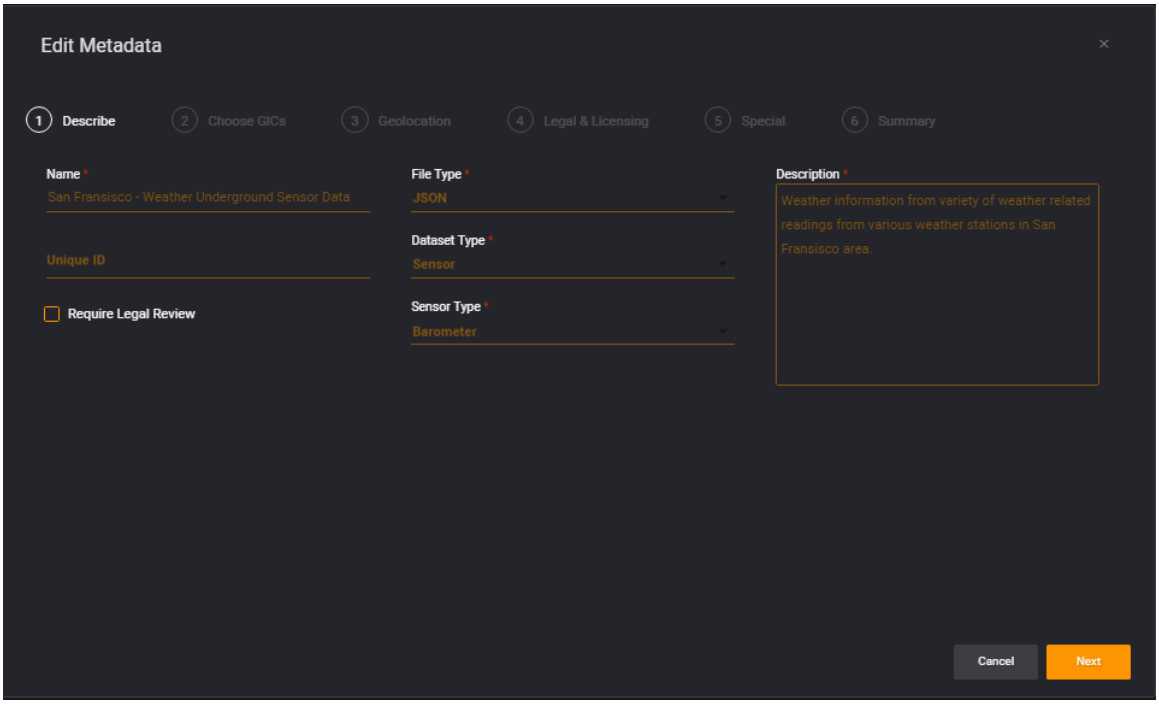
- **meta.name :** this is the display name for the metadata configuration.
- **meta.description** : this is the long description for the metadata configuration.

You will also want to review the various items, checking against the metadata specification and the results of the values returned by the domain data services.

The various domain fields, such as **legal.type**, **container.type**, **dataset.type**, **dataset.schemaInfo.type** or **dataset.sensorInfo.type,** are identifiers that are returned by the respective `api/domain/<id>` service.

For **gicsCode.id**, level and name the `api/domain/giccode` service contains the possible values.

These can also be added and then reviewed with the terbine.io UI in the Metadata Configuration edit wizard. Shown here:

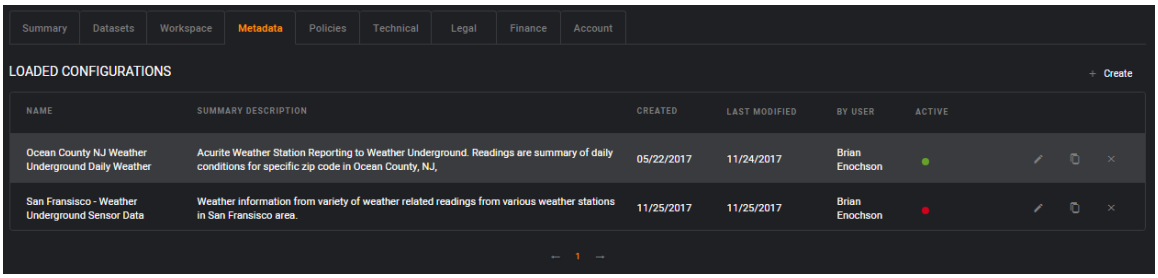


In the file **MetadataServiceClientTest.java** the following tests refer to metadata services.

- testInsertMetadata
- testUpdateMetadata
- testGetMetadata

After running the **testInsertMetadata** test within the modified insertmetadata.json file, you will want to note the UUID ID for the new metadata configuration and update **unit_test.properties** file with the metadata id that was created. This will be needed for following related to update and retrieve (get) metadata as well as in the **Dataset Services**.

After modifying insertmetadata.json as you like and running testInsertMetadata you will have added a new metadata configuration. At this time you can login to the system and view the metadata in your Dashboard.



Notice the second entry, which was added by the test configuration is available for data set loading, but not active (Red Dot in Active column). This means you can work with it, but it is not publicly available and will not yet be indexed.

If you want it to be searchable and viewed by the public you can activate it by clicking on the red activation flag and it will turn green.

DATASET SERVICES

For dataset examples go to the file DatasetServiceClient.java and DatasetServiceClientTest.java.

This assumes the user, password and baseUri have been set in unit_test.properties.

For every metadata configuration there will be zero or more datasets loaded. Datasets are the actual content that conforms to the information defined within the metadata configuration.

To understand this relationship we often have used the movie file analogy. If you have a movie file (in our case dataset) you are able to view the movie, but will not have information related to release year, actors, producers, soundtrack etc. If you go to imdb.com and look up the movie then you can find out this information. In the same way that IMDB provides information about the movie, Metadata Configuration provide information about your IoT data.

In the file **DatasetServiceClientTest.java** the following tests refer to dataset services.

- testUploadDataset
- testGetAllDatasetsForMetadata
- testDownloadDataset

It is assumed that metadata has been inserted for the test file and the metadata unique UUID has been added to **unit_test.properties**.

Two sample datasets are provided in the src/test/resources/sample directory, containing Weather Underground data collected from Weather Stations in San Francisco.

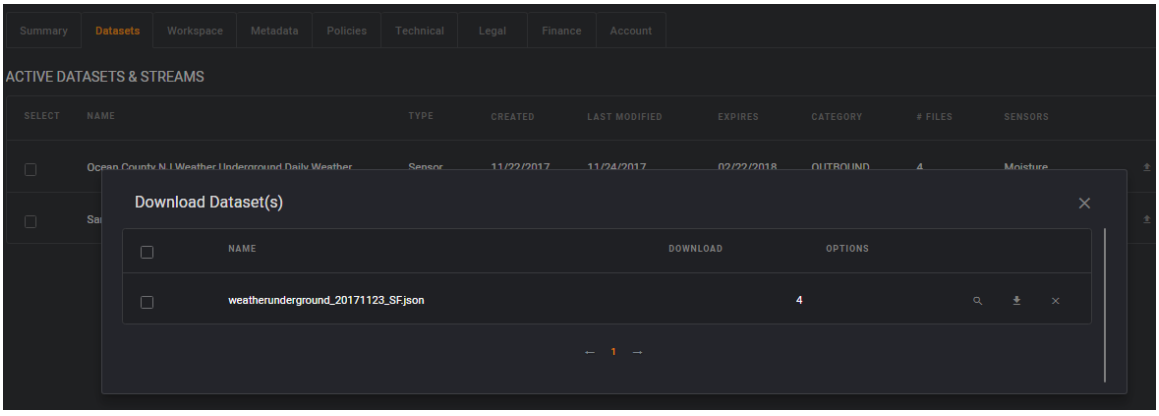
- weatherunderground_20171123_SF.json
- weatherunderground_20171124_SF.json

This is typical for IoT data containing location and time information and one or multiple readings related to that. In this case these are daily aggregated summaries for all weather stations.

Important when uploaded to try and set the data start date and data end date if possible for enhanced discovery by data ranges.

The examples also call the getDatasetIdForMetadata method which shows how to retrieve the data container UUID ID for a metadata confirmation under which all datasets are loaded.

Once it is uploaded you can retrieve dataset information, download specific items and preview them on the UI.



This shows for a metadata one of the datasets uploaded, accessed by clicking on the Dataset entry within the UI. The icons on the left allow you to Preview, Download and Delete a dataset.

REVISIONS

Version	Date	Name	Description
0.9.6	2017/11/20	Brian Enochson	For Review
0.9.7	2017/11/28	Brian Enochson	Added Screen Shots and Additional Examples
0.9.8	2018/06/04	Brian Enochson	Updated to match schema in latest metadata specification.