TERBINE PYTHON LIBRARY

Last Revision: April 7, 2020
Version: 1.0.1

TABLE OF CONTENTS

Contents

OVERVIEW

This guide documents all of the methods available in the Python Library to be used to interface with the Terbine Application Programming Interface (API). This can be used in conjunction with the Terbine API Overview document, where detailed information to relevant services are found. Knowledge of Python, Dictionaries, and JSON is assumed. The following reference will provide a description of each function. A Terbine Python Library Tutorial and Jupyter Notebook and Pandas Example containing more in-depth examples for usage can be found following this reference.

GETTING STARTED

Before you begin, you will need to install the library to use in your Python code. There are two ways to install the library, by downloading manually from the Terbine site or by using the *pip* command. The library will also require the Python **Requests** HTTP library (If not already installed, the Requests library can also be downloaded by adding "requests" following the Terbine Python Library in the *pip* command).

.

To install the Python Library using *pip*, run the following in your command line:

```
pip
```

```
$ pip install terbine-py-library
```

INITIALIZE THE LIBRARY

Once installed, the Python Library must be imported and initialized. Import the library and initialize it with the base uniform resource identifier (URI) of the API to interface with. To create the Terbine Library instance, you will need to pass the base URI, "https://api.terbine.io", as a parameter to the new PyLibrary constructor. The library's functions will then be available to use within your code.

```
Initialize
```

```
import terbinePyLibrary as terbine

baseURI = "https://api.terbine.io"
terbine = terbine.PyLibrary(baseURI)
```


AUTHORIZATION

LOGIN

The Login function logs a user in to Terbine's System and provides the user's **token** (`loginResult['token']`) to be used in future functions requiring authentication, as well as the **orgID** (`loginResult['orgid']`) for use in workspace functions.

Parameters

- Email (string): Email address associated with user's Terbine.io account
- Password (string): Password associated with user's Terbine.io account

Returns

- JSON login response

Example

```
loginResult = terbine.login(username, password)
```

Sample Result

```
{
      "lastlogin":"2020-01-29T11:40:45Z",
      "username":"example@terbine.com",
      "displayname":"User Display Name",
      "token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJqZW5hbEB0ZXJiaW5lLmNvbSIsIm
      V4cCI6MTU4MDM1NjE1Nn0.JwFZQZv0j1UTnK-bDPsvhqwKFUTcxe--7x9mNhRDE0U",
      "userid":"aab864d2-74fc-44c1-b0d3-632829719929",
      "orgid":"6d94ff99-4b41-42d0-b3a4-d14d89da0838",
      "orgname":"Company Name",
      "orgType":3,
      "supervisor":false,
      "tokenlifeseconds":36300,
      "numberNotices":0,
// Truncated
```


SEARCH

SEARCH SERVICE

The Search function sends a query to search Terbine's available metadata instances and allows for customization of displayed results. The response provides a list of these instances that match the parameters. A metadata instance's **guid** (`(searchResult[index])['id']`) which is used in the Add to Workspace and List functions can be retrieved from this response.

Parameters:

- Query (string): Query word or unique ID to search
- PageNumber (integer): The result's page number to display
- PageSize (integer): The number of results to display on the page
- Sort (string): Sort by type, "DATEADDED"/"ALPHA"
- Order (string): Order of items, "DESC"/"ASC"

Returns:

- List of search results in JSON format

```
Example
```

```
searchResult = terbine.search(query, pageNumber, pageSize,  sort, order)
```

```
Sample Result
```

```
[
      {
            "Metadata":null,
            "Type":2,
            "Source":"ingestion",
            "Id":"9450a6e6-e0d5-43bf-9bb3-9ec59e9b87dc",
            "orgId":"6d94ff88-4b41-41d0-b3a4-d14d89da0939",
            "ownerOrgId":null,
            "alphaSort":"Nitric Oxide / Liberty Avenue, Lisbon, Portugal /
            2015",
            "title":"Nitric Oxide / Liberty Avenue, Lisbon, Portugal /
            2015",
            "description":"Readings of nitric oxide were taken by an air
            monitoring station located along the roadside of Liberty Avenue
// Truncated
```


WORKSPACE

LIST WORKSPACE

The List Workspace function displays a list of metadata instances currently added to a user's workspace. This response also provides each instance's **workspaceID** (`(workspaceListResult[index])['id']`) which is required for use in the Lock/Flag/Delete functions.

Parameters:

- Token (string): Token retrieved from Login response
- OrgID (string): Organization ID retrieved from Login response
- PageNumber (integer): The result's page number to display
- PageSize (integer): The number of results to display on the page
- Order (string): Order of items, "DESC"/"ASC"
- OrderBy(string): Order by type, "description"/"title" etc.

Returns:

- List of current workspace items in JSON format

```
Example
```

```
workspaceListResult = terbine.listWorkspace(token, orgID, pageNumber,
pageSize, order, orderBy)
```

```
Sample Result
```

```
[
      {
            "typeId":1,
            "metadataType":"Archival",
            "id":"e0c4e29c-5780-4db0-91fe-feca64bf0158",
            "orgId":"c8dbd0ae-15a4-45ae-b595-6b75aea54de4",
            "name":"Water Levels / Oslo, Norway / 2014",
            "description":"Readings of water levels are taken by a float
            gauge located in Oslo, Norway. This float gauge is located
            within the Inner
// Truncated
```

ADD TO WORKSPACE

The Add to Workspace function adds a chosen metadata instance to the user's workspace where it can be later flagged, locked in for download, or deleted.

Parameters:

- Token (string): Token retrieved from Login response
- Guid (string): Unique Identifier of a Metadata Instance retrieved from Search response or Terbine.io

- OrgID (string): Organization ID retrieved from Login response

Returns:

- Added to Workspace response in JSON format, status of 1 if successful

```
Example
```

```
addWorkspaceResult = terbine.addToWorkspace(token, guid, orgID)
```

```
Sample Result
```

```
{
      "typeId":null,
      "metadataType":null,
      "Id":"b81a3d5c-a5fd-4b62-a459-2b3390baa282",
      "orgId":"c8dbd0ae-15a4-45ae-b595-6b75aea54de4",
      "name":"Nitric Oxide / Lyndon B Johnson Freeway, Dallas, Texas, United
      States / 2015",
      "description":"Workspace entry added for Nitric Oxide / Lyndon B
      Johnson Freeway, Dallas, Texas, United States / 2015",
      "metadataId":"3c48c005-c3e6-4bf2-a968-5f35c69e27ad",
      "datasetId":null,
      "uid":"de4fe9f8-d68a-4592-9fb3-ba2ff8e8f5f7",
      "type":1,
      "status":1,
//Truncated
```

LOCK IN WORKSPACE

The Lock in Workspace function locks a metadata instance found in the workspace and activates the dataset/stream for use in the user's dashboard.

Parameters:

- Token (string): Token retrieved from Login response
- OrgID (string): Organization ID retrieved from Login response
- WorkspaceID (string): Unique ID of item in workspace

Returns:

- Result of locking the workspace item in JSON format, status of 3 if successful

```
Example
```

```
lockResult = terbine.lockToWorkspace(token, orgID, workspaceID)
```

```
Sample Result
```

```
{
      "typeId":null,
      "metadataType":null,
```



```
      "id":"e0c4e29c-5780-4db0-91fe-feca64bf0158",
      "orgId":"c8dbd0ae-15a4-45ae-b595-6b75aea54de4",
      "name":null,
      "description":null,
      "metadataId":"08a12f66-27b5-47f5-99d6-32088ac92116",
      "datasetId":null,
      "uid":"de4fe9f8-d68a-4592-9fb3-ba2ff8e8f5f7",
      "type":1,
      "status":3,
// Truncated
```


FLAG IN WORKSPACE

The Flag in Workspace function flags a metadata instance found in the workspace to distinguish it from other added workspace items.

Parameters:

- Token (string): Token retrieved from Login response
- OrgID (string): Organization ID retrieved from Login response
- WorkspaceID (string): Unique ID of item in workspace

Returns:

- Result of flagging the workspace item in JSON format, status of 2 if successful

```
Example
```

```
flagResult = terbine.flagWorkspace(token, orgID, workspaceID)
```

```
Sample Result
```


```
{
      "typeId":null,
      "metadataType":null,
      "id":"e0c4e29c-5780-4db0-91fe-feca64bf0158",
      "orgId":"c8dbd0ae-15a4-45ae-b595-6b75aea54de4",
      "name":null,
      "description":null,
      "metadataId":"08a12f66-27b5-47f5-99d6-32088ac92116",
      "datasetId":null,
      "uid":"de4fe9f8-d68a-4592-9fb3-ba2ff8e8f5f7",
      "type":1,
      "Status":2,
// Truncated
```


DELETE FROM WORKSPACE

The Delete from Workspace function removes a metadata instance located in the workspace.

Parameters:

- Token (string): Token retrieved from Login response
- OrgID (string): Organization ID retrieved from Login response
- WorkspaceID (string): Unique ID of item in workspace

Returns:

- Result of deleting the workspace item in JSON format, code of OK if successful

Example

```
deleteWorkspaceResult = terbine.deleteFromWorkspace(token, orgID,
workspaceID)
```

Sample Result

```
{
      "code":"OK",
      "message":"Workspace with id e0c4e29c-5780-4db0-91fe-feca64bf0158
      deleted",
      "messageDate":"2020-01-31T20:11:55Z"
}
```


METADATA

METADATA BY ID

The Metadata function returns the metadata field values for a certain metadata instance. The response also provides the **datasetID** (`((metadataResult['dataset'])[index])['id']`) which is required for listing and downloading multiple files associated with the metadata instance.

Parameters:

- Token (string): Token retrieved from Login response
- Guid (string): Unique Identifier of a Metadata Instance retrieved from Search response or Terbine.io

Returns:

- Values of all metadata fields for a metadata instance in JSON format

```
Example
```

```
metadataResult = terbine.metadata(token, guid)
```

```
Sample Result
```


```
{
      "id":"60ae666d-380f-45e2-b651-8a84dbb71c06",
      "orgId":"6d94ff88-4b41-41d0-b3a4-d14d89da0939",
      "orgName":"TERBINE",
      "identifier":{
            "createUpdateInfo":{
                  "createUser":"aab864c1-74fc-44c1-b0d3-531828717729",
                  "createDate":"2020-01-30T09:51:41Z",
                  "updateUser":"496fa3e3-d7e4-4ee9-aa96-feb213994c7f",
                  "updateDate":"2020-01-30T09:55:28Z"
            },
            "id":null,
            "extId":null,
            "urn":null,
            "uri":null,
            "tid":null
      },
      "meta":{
            "createUpdateInfo":{
                  "createUser":"aab864c1-74fc-44c1-b0d3-531828717729",
                  "createDate":"2020-01-30T09:51:41Z",
                  "updateUser":"496fa3e3-d7e4-4ee9-aa96-feb213994c7f",
                  "updateDate":"2020-01-30T09:55:28Z"
            },
      "name":"Electricity Consumption / Aube, Grand-Est, France / 2014",
      "description":"Measurements of electricity consumption data by
// Truncated
```


DATASET

LIST FILES FOR DATASET

The List function returns the list of all files within a dataset. This response also provides the **contentID** (`((listResult[index])['id'])`) of each file which is required in the Download Single File function, as well as the original **file name** (`((listResult[index])['originalName'])`).

Parameters:

- Token (string): Token retrieved from Login response
- DatasetID (string): Dataset ID retrieved from Metadata response

Returns:

- List of files within a dataset in JSON format

```
Example
```

```
listResult = terbine.list(token, datasetID)
```

```
Sample Result
```

```
[
      {
      "id":"56783dd8-c02e-48a9-b5fd-ed35a6264c4e",
      "metadataId":"60ae666d-380f-45e2-b651-8a84dbb71c06",
      "datasetId":"abe07a75-bfab-40af-9666-d591fb18fcf2",
      "externalId":"1ee8381a70764511b0516a3129f53db3",
      "organizationId":"6d94ff88-4b41-41d0-b3a4-d14d89da0939",
      "type":1,
      "status":1,
      "fileStoreLocation":"6d94ff88-4b41-41d0-b3a4-d14d89da0939/60ae666d
      -380f-45e2-b651-8a84dbb71c06/GHxSqS1R3F8633PWLsgdisXN.terbine.xlsx",
      "fileExt":"xlsx",
      "size":998946,
// Truncated
```

DOWNLOAD SINGLE FILE

The Download Single File function provides an object containing the raw data of a single file within a dataset.

Parameters:

- Token (string): Token retrieved from Login response
- ContentID (string): Content ID retrieved from List response

Returns:

- Object containing raw data of downloaded single file (can be extracted with .text or .content)


```
Example
singleResult = terbine.downloadSingle(token, contentID)
```

```
Sample Result
<Response [200]>  # Successful
```

DOWNLOAD MULTIPLE FILES

The Download Multiple File function provides an object containing the raw data of multiple files within a zip file.

Parameters:

- Token (string): Token retrieved from Login response
- DatasetID (string): Dataset ID retrieved from Metadata response
- Max (integer): maximum number of files to download

Returns:

- Object containing raw data of downloaded zip file containing multiple files (can be extracted with .text or .content)

```
Example
multipleResult = terbine.downloadMultiple(token, datasetID, max)
```

```
Sample Result
<Response [200]>  # Successful
```


TERBINE PYTHON LIBRARY TUTORIAL

OVERVIEW

This document provides more in-depth examples of usage of the methods used in the Terbine Python Library. For example usage, simple user input prompts are used and functions that require prior calls include sequencing notes in the example's comments.

GETTING STARTED

To begin, the Python Library must be downloaded manually from the Terbine site or downloaded through *pip*. In this example we have downloaded the Python Library using the pip command:

```
pip
```

```
$ pip install terbine-py-library
```

INITIALIZE THE LIBRARY

In our script, we start by importing the Python Library and naming it as terbine. We then initialize a new library instance using the Base URI "https://api.terbine.io" and assigning it to the "terbine" variable.

```

```

```
import terbinePyLibrary as terbine

# Create Library Instance
terbine = terbine.PyLibrary("https://api.terbine.io")
```


AUTHORIZATION

LOGIN

The Login function is necessary for many of the API calls found in the Python Library, as they require authentication of the user via a **token** (`loginResult['token']`). Logging in also provides the user's **orgID** (`loginResult['orgid']`) which is used to access their workspace. The Login function must be called prior to these functions.

After downloading and initializing the Python Library, we begin by adding the login function to our script. In this example, email and password variables are retrieved from the user's input and used as parameters. The Login function is called and assigned to a loginResult variable. The response received can then be used to assign the token and orgID to be used in other functions requiring authentication.

Login

```
# Retrieve Input
username = input("Username: ")
password = input("Password: ")

# Login
loginResult = terbine.login(username, password)
print(loginResult)

# Assign Authentication Variables
token = loginResult['token']
orgID = loginResult['orgid']
```


SEARCH

SEARCH SERVICE

The Search function sends a query to search Terbine's available metadata instances and allows for customization of displayed results using parameters. The parameters allowed include the page number (page displayed), the page size (items per page), sort (DATEADDED to order by date or ALPHA to order by name) and order (DESC for descending or ASC for ascending). From the result, a user can retrieve a metadata instance's **guid** (`(searchResult[index])['id']`) which is used in the Add to Workspace and Metadata functions.

To add a Search Service in our program, we first retrieve query text and parameters from the user's input. The searchResult variable calls the search function and retrieves the response, an array of the results whose display is customized by the included parameters. The result can then be used to retrieve a **guid**, which will be shown in the Add to Workspace and Metadata examples.

```
Search
```

```
# Retrieve Input
query = input("Query: ")
pageNum = input("Page Number: ")
pageSize = input("Page Size: ")
sort = input("Sort: ")
order = input("Order: ")

# Search
searchResult = terbine.search(query, pageNum, pageSize, sort, order)
print(searchResult)
```


WORKSPACE

LIST WORKSPACE

The List Workspace function displays a list of metadata instances currently added to a user's workspace. This call requires prior authentication by obtaining the token and orgID from the Login function. Like the Search function, parameters to customize the display of the list include page number (page displayed), page size (items per page), order (DESC for descending or ASC for ascending), and orderBy (by description, title, etc.) The response will provide each of the listed instances **workspaceIDs** (`(workspaceListResult[index])['id']`), which is required for use in the Lock/Flag/Delete functions.

To access the workspace items, we begin by retrieving the parameters from the user's input. The List Workspace function is then called and the workspaceListResult variable is assigned the response, an array of the list items. A workspaceID variable can then be assigned using this response, which will be shown in the Lock/Flag/Delete examples.

```
List Workspace
```

```
# Login Function is called prior to retrieve token and orgID

# Retrieve Input
pageNum = input("Page Number: ")
pageSize = input("Page Size: ")
order = input("Order: ")
orderBy = input("Order By: ")

# List Workspace
workspaceListResult = terbine.listWorkspace(token, orgID, pageNum, pageSize,
order, orderBy)
print(workspaceListResult)
```

ADD TO WORKSPACE

The Add to Workspace function adds a chosen metadata instance to the user's workspace where it can be later flagged, locked in for download, or deleted. Authentication is required to access the workspace, so the Login function is called prior to retrieve the token and orgID. The guid parameter can then be retrieved from the Search result.

In order to add an item, the searchResult index of the metadata instance to be added is retrieved from the user's input. It is then used to retrieve the guid from the id of the item at that index. The Add to Workspace function is called by the addResult variable and retrieves the response. The result will have a status of "1" when added.

```
Add to Workspace
```



```
# Login and Search Functions are called prior to retrieve token, orgID
# and searchResult

# Retrieve Input
index = int(input("Index of item to add: "))

# Assign GUID
guid = (searchResult[index])['id']

# Add item to Workpace
addResult = terbine.addToWorkspace(token, guid, orgID)
print(addResult)
```

LOCK IN WORKSPACE

The Lock in Workspace function locks a metadata instance found in the workspace and activates the dataset/stream for use in the user's dashboard. Authentication is required to access the workspace, so the Login function is called prior to retrieve the token and orgID. Listing of the workspace is also required to retrieve the item's workspaceID.

To lock an item, we begin by retrieving the index of the chosen metadata instance found in the workspaceListResult from the user's input. This is then used to obtain the workspaceID from the item at that index. The Lock in Workspace function is called by the lockResult variable and retrieves the response. The result will have a status of "3" when locked.

```
Lock in Workspace
```

```
# Login and List Workspace Functions are called prior to retrieve token,
# orgID and workspaceListResult

# Retrieve Input
index = int(input("Index of item to lock: "))

# Assign GUID
workspaceID = (workspaceListResult[index])['id']

# Lock item to Workspace
lockResult = terbine.lockToWorkspace(token, orgID, workspaceID)
print(lockResult)
```

FLAG IN WORKSPACE

The Flag in Workspace function flags a metadata instance found in the workspace to distinguish it from other added workspace items. Authentication is required to access the workspace, so the Login function is called prior to retrieve the token and orgID. Listing of the workspace is also required to retrieve the item to flag's workspaceID.

In this example, the index of the metadata instance to be flagged from the workspaceListResult is retrieved from the user's input. It is then used to retrieve the workspaceID from the id of the item at that index. The Flag in Workspace function is called by the flagResult variable and retrieves the response. The result will have a status of "2" when flagged.

```
Flag in Workspace
```

```
# Login and List Workspace Functions are called prior to retrieve token,
# orgID and workspaceListResult

# Retrieve Input
index = int(input("Index of item to flag: "))

# Assign GUID
workspaceID = (workspaceListResult[index])['id']

# Flag item in Workspace
flagResult = terbine.flagWorkspace(token, orgID, workspaceID)
print(flagResult)
```

DELETE FROM WORKSPACE

The Delete from Workspace function removes a metadata instance located in the workspace. Authentication is required to access the workspace, so the Login function is called prior to retrieve the token and orgID. Listing of the workspace is also required to retrieve the item to delete's workspaceID.

To delete an item, we begin by retrieving the user's input of the index of the metadata instance to be deleted from the workspaceListResult. It is then used to retrieve the workspaceID from the id of the item at that index. The Delete in Workspace function is called by the deleteResult variable and retrieves the response. The result will show a code of "OK" if deleted successfully.

```
Delete from Workspace
```

```
# Login and List Workspace Functions are called prior to retrieve token,
# orgID and workspaceListResult

# Retrieve Input
index = int(input("Index of item to delete: "))

# Assign GUID
workspaceID = (workspaceListResult[index])['id']

# Lock item to Workpace
deleteResult = terbine.deleteFromWorkspace(token, orgID, workspaceID)
print(deleteResult)
```


METADATA

METADATA BY ID

The Metadata function returns the metadata field values for a certain metadata instance. Authentication is required and the Login Function must be called prior to retrieve the token. The response provides the **datasetID** (`((metadataResult['dataset'])[index])['id']`) within a dataset array. This is required for listing and downloading multiple files associated with the metadata instance.

To obtain the metadata of a dataset, first the index of the chosen item in the searchResult is retrieved from the user's input. It is then used to retrieve the guid from the id of the item at that index. The metadataResult variable calls the Metadata function and awaits the call to complete. The result is saved as a global variable for it to be used in future functions that require a datasetID.

```
Metadata
```

```
# Login and Search Service Functions are called prior to retrieve token and
# searchResult


# Retrieve Input
index = int(input("Index of item: "))

# Assign GUID
guid = (searchResult[index])['id']

# Get Metadata
metadataResult = terbine.metadata(token, guid)
print(metadataResult)
```


DATASET

LIST FILES FOR DATASET

The List function returns the list of all files within a dataset. Authentication is required and the Login Function must be called prior to retrieve the token. The datasetID retrieved from the Metadata function result is also used as a parameter. The List function response provides the **contentID** (`(listResult[index])['id']`) of each file which is required in the Download Single File function, as well as the original **file name** (`(listResult[index])['originalName']`).

After calling the Metadata function, we retrieve the index from the user's input of the chosen item in the metadata dataset array. It is then used to retrieve the guid from the id of the item at that index. The listResult variable calls the List function retrieves the result, a list of files within the dataset. This can then be used to obtain the contentID, which is shown in the Download Single File example.

List

```
# Login and Metadata Functions are called prior to retrieve token and
# metadataResult

# Retrieve Input
index = int(input("Index of item: "))

# Assign Dataset ID
datasetID = ((metadataResult['dataset'])[index])['id']

# List Files
listResult = terbine.list(token, datasetID)
print(listResult)
```

DOWNLOAD SINGLE FILE

The Download Single File function provides an object containing the raw data of a single file within a dataset. Authentication is required and the Login Function must be called prior to retrieve the token. The Metadata and List functions must also precede this function to retrieve the contentID parameter.

After calling the List function, we begin by retrieving the index of the chosen item in the listResult array from the user's input. It is then used to retrieve the contentID and fileName from the id of the item at that index. The downloadSingle variable calls the Download Single function and retrieves the response and extracts it using `.text`. This example also shows how to download the file locally to the user's system.


```
Download Single File
```

```
# Login and List Files for Dataset Functions are called prior to retrieve
# token and listResult

# Retrieve Input
index = int(input("Index of item: "))

# Assign GUID and file name
contentID = (listResult[index])['id']
fileName = (listResult[index])['originalName']

# Get File Content
singleResult = terbine.downloadSingle(token, contentID)
print(singleResult.text)

#Download Locally
open(fileName, 'wb').write(singleResult.content)
```

DOWNLOAD MULTIPLE FILES

The Download Multiple File function provides an object containing the raw data of multiple files within a zip folder. Authentication is required and the Login Function must be called prior to retrieve the token. The Metadata function must also precede this function to retrieve the datasetID parameter.

After calling the Metadata function, we first retrieve the index from the user's input of the chosen item in the metadataResult dataset array. It is then used to retrieve the datasetID from the id of the item at that index. The downloadMultiple variable calls the Download Multiple function and retrieves the object containing the zip file's raw data. This example shows how to download the object locally to the user's system.

```
Download Multiple Files
```

```
# Login and Metadata Functions are called prior to retrieve
# token and metadataResult

# Retrieve Input
index = int(input("Index of item: "))
max = input("Max: ")

# Assign Dataset ID and fileName
datasetID = ((metadataResult['dataset'])[index])['id']
fileName = 'example.zip'

# Get File Content
multipleResult = terbine.downloadMultiple(token, datasetID, max)

#Download Locally
open(fileName, 'wb').write(multipleResult.content)
```


JUPYTER NOTEBOOK AND PANDAS EXAMPLE

OVERVIEW

This document provides a more in-depth example of usage of the methods used in the Terbine Python Library within Jupyter Notebook, a web application used for creating and sharing documents that contain live code, equations, visualizations and narrative text. Pandas, an open source data analysis and manipulation tool, is also used to show the ease of manipulating and visualizing the data retrieved from Terbine's system. This example notebook will also be available for download on the Tools and Specs page.

GETTING STARTED

To begin, run Jupyter Notebook and open a new notebook (to download Jupyter Notebook, follow the install instructions located on their site). In the first cells, we import dependencies that will be needed in our project (**pandas** for data manipulation and visualization, **io** to create a useable file object from the data we retrieve, and **sys** to use command line commands).

We first install the **Requests** library using pip. Note that "`!{sys.executable} -m`" is used prior to the pip install command to allow for using this system command within the notebook.

```
In [1]: import pandas as pd
        import io
        import sys
```

```
In [2]: # Get Requests Library if not already installed
        !{sys.executable} -m pip install requests
```

Next, we install the Terbine Python Library using pip. The terbinePyLibrary is then imported as a terbine variable. We can now run these cells and the Terbine Python Library will be ready for use.

```
In [3]: # Install Terbine Python Library
        !{sys.executable} -m pip install terbine-py-library
```

```
In [4]: # Import the Terbine Python Library Modules
        import terbinePyLibrary as terbine
```


AUTHORIZATION

To login, we must first input our Terbine.io credentials. In this example, we input the username followed by the password password and save them to variables. Next, the Login function is called and saved to the loginResult variable. The response can be printed to verify the login was successful.

We then retrieve the **token** (`loginResult['token']`), which will be used in our subsequent function calls.

```
In [*]: # Retrieve User Credentials
        username = input("Username: ")
        password = input("Password: ")

        Username:

In [*]: # Login and Retrieve Token
        loginResult = terbine.login(username, password)
        token = loginResult['token']
        print(loginResult)
```

SEARCH

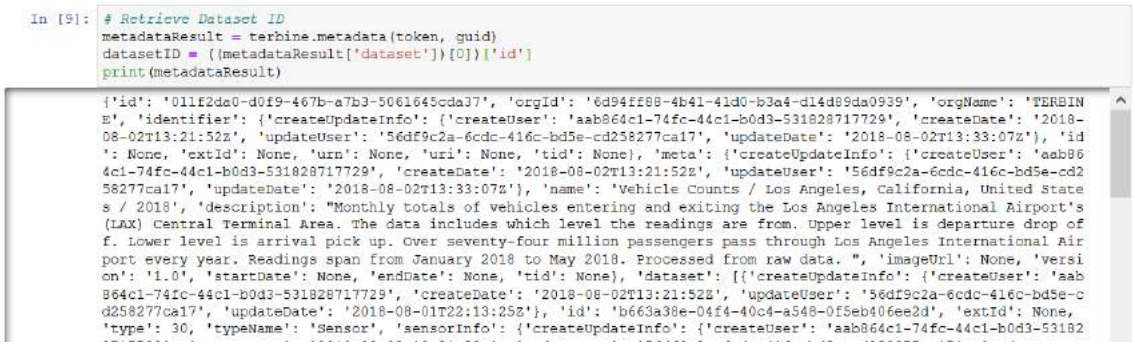
We must now choose the data we would like to visualize. We begin by searching the system for a dataset using the Search Function. In this example, we use “vehicle counts california 2018” as our query parameter. We decide to not adjust the optional pagination parameters, so no further parameters are added after the query.

We decide to use the first result from the matches, “Vehicle Counts / Los Angeles, California, United States / 2018”. We must now retrieve the **guid** (`(searchResult[0])['id']`), which will be used next for calling the Metadata function.

```
In [8]: # Search for Dataset and Retrieve GUID
        searchResult = terbine.search("vehicle counts california 2018")
        guid = (searchResult[0])['id']
        print(searchResult)

[{'metadata': None, 'type': 2, 'source': 'ingestion', 'id': '011f2da0-d0f9-467b-a7b3-5061645cda37', 'orgId': '6d94ff8
8-4b41-41d0-b3a4-d14d89da0939', 'ownerOrgId': None, 'alphaSort': 'Vehicle Counts / Los Angeles, California, United St
ates / 2018', 'title': 'Vehicle Counts / Los Angeles, California, United States / 2018', 'description': "Monthly tota
ls of vehicles entering and exiting the Los Angeles International Airport's (LAX) Central Terminal Area. The data inc
ludes which level the readings are from. Upper level is departure drop off. Lower level is arrival pick up. Over seve
nty-four million passengers pass through Los Angeles International Airport every year. Readings span from January 201
8 to May 2018. Processed from raw data. ", 'dateIndexed': '2020-03-12T13:02:15Z', 'dateAdded': '2018-08-02T13:21:52Z
```


METADATA

The Metadata function will be used to retrieve the datasetID needed to access the content. We pass the token and guid variables as parameters and call the Metadata function. The **datasetID** (`((metadataResult['dataset'])[0])['id']`) can then be retrieved from the result to be used to list the datasets available. Note, the metadataResult also provides other useful information such as the file type (csv in this example) and the schema body, which will later be referenced when we manipulate the data.



```
In [9]: # Retrieve Dataset ID
        metadataResult = terbine.metadata(token, guid)
        datasetID = ((metadataResult['dataset'])[0])['id']
        print(metadataResult)
```

{'id': '011f2da0-d0f9-467b-a7b3-5061645cda37', 'orgId': '6d94ff88-4b41-41d0-b3a4-d14d89da0939', 'orgName': 'TERBIN
E', 'identifier': {'createUpdateInfo': {'createUser': 'aab864c1-74fc-44c1-b0d3-531828717729', 'createDate': '2018-
08-02T13:21:52Z', 'updateUser': '56df9c2a-6cdc-416c-bd5e-cd258277ca17', 'updateDate': '2018-08-02T13:33:07Z'}, 'id
': None, 'extId': None, 'urn': None, 'uri': None, 'tid': None}, 'meta': {'createUpdateInfo': {'createUser': 'aab86
4c1-74fc-44c1-b0d3-531828717729', 'createDate': '2018-08-02T13:21:52Z', 'updateUser': '56df9c2a-6cdc-416c-bd5e-cd2
58277ca17', 'updateDate': '2018-08-02T13:33:07Z'}, 'name': 'Vehicle Counts / Los Angeles, California, United State
s / 2018', 'description': "Monthly totals of vehicles entering and exiting the Los Angeles International Airport's
(LAX) Central Terminal Area. The data includes which level the readings are from. Upper level is departure drop of
f. Lower level is arrival pick up. Over seventy-four million passengers pass through Los Angeles International Air
port every year. Readings span from January 2018 to May 2018. Processed from raw data. ", 'imageUrl': None, 'versi
on': '1.0', 'startDate': None, 'endDate': None, 'tid': None}, 'dataset': [{'createUpdateInfo': {'createUser': 'aab
864c1-74fc-44c1-b0d3-531828717729', 'createDate': '2018-08-02T13:21:52Z', 'updateUser': '56df9c2a-6cdc-416c-bd5e-c
d258277ca17', 'updateDate': '2018-08-01T22:13:25Z'}, 'id': 'b663a38e-04f4-40c4-a548-0f5eb406ee2d', 'extId': None,
'type': 30, 'typeName': 'Sensor', 'sensorInfo': {'createUpdateInfo': {'createUser': 'aab864c1-74fc-44c1-b0d3-53182

DATASET

We can then list the available files for a dataset by passing the token and datasetID to the List function. After it is called, we can retrieve the **contentID** (`(listResult[0])['id']`) for a certain file in the list, which will be used to download it. This dataset only contains one file, so we choose its index to get its contentID.

```
In [10]: # Retrieve Content ID
         listResult = terbine.list(token, datasetID)
         contentID = (listResult[0])['id']
         print(listResult)
```

[{'id': '53ad161f-80c5-4efb-8b4b-9e199b60d219', 'metadataId': '011f2da0-d0f9-467b-a7b3-5061645cda37', 'datasetId': 'b
663a38e-04f4-40c4-a548-0f5eb406ee2d', 'externalId': '71b2898eaa3849b2ad539ad9423a9f83', 'organizationId': '6d94ff88-4
b41-41d0-b3a4-d14d89da0939', 'type': 1, 'status': 1, 'fileStoreLocation': '6d94ff88-4b41-41d0-b3a4-d14d89da0939/011f2
da0-d0f9-467b-a7b3-5061645cda37/GWJq6K5kGIt3kdoprP0JE5L0.terbine.csv', 'fileExt': 'csv', 'size': 788, 'fileHash': 'b9
e8a8a90f7af0f02cae149dbf91fa490749caa3', 'lastModifyTime': '2018-08-01T22:13:14Z', 'dateUploaded': '2018-08-01T22:13:
14Z', 'dateStart': '2018-08-01T22:13:14Z', 'dateEnd': None, 'deleted': 0, 'originalName': 'vehicle_counts_los_angeles
_california_united_states_2018_1.csv', 'numDownloads': 5}]

To download the file, we pass the token and contentID and call the Download Single File function. We print the result's text to preview the data to be used for the visualization.


```
In [11]: # Download Data
         singleResult = terbine.downloadSingle(token, contentID)
         print(singleResult.text)
```

```
01/01/2018 12:00:00 AM,Entry,Lower Level,1264344
01/01/2018 12:00:00 AM,Entry,Upper Level,1501903
01/01/2018 12:00:00 AM,Exit,Lower Level,1929900
01/01/2018 12:00:00 AM,Exit,Upper Level,994130
03/01/2018 12:00:00 AM,Entry,Lower Level,1342159
03/01/2018 12:00:00 AM,Entry,Upper Level,1559233
03/01/2018 12:00:00 AM,Exit,Lower Level,1934828
03/01/2018 12:00:00 AM,Exit,Upper Level,1039613
04/01/2018 12:00:00 AM,Entry,Lower Level,1329798
04/01/2018 12:00:00 AM,Entry,Upper Level,1517056
04/01/2018 12:00:00 AM,Exit,Lower Level,1917622
04/01/2018 12:00:00 AM,Exit,Upper Level,994651
05/01/2018 12:00:00 AM,Entry,Lower Level,1439131
05/01/2018 12:00:00 AM,Entry,Upper Level,1601307
05/01/2018 12:00:00 AM,Exit,Lower Level,2059938
05/01/2018 12:00:00 AM,Exit,Upper Level,1044608
```

MANIPULATE DATA

We must now read in the file using Pandas to create a DataFrame table. The file we chose is in CSV format, so we can use Panda's read_csv function to convert the data to the table. The first parameter will be the data/file (`singleResult.content`), which must then be decoded and converted using io's StringIO function. As seen in the singleResult's text, the data's first row did not include column names, so we can use the names list parameter to add them. We can find the column names in the schema body field found in the metadataResult. Calling this function will then create the table of the data.

```
In [12]: # Use Pandas to Convert Data to Dataframe object
         data = singleResult.content
         c = pd.read_csv(io.StringIO(data.decode('utf-8')), names=["ReportingMonth","Entry/Exit","Upper/Lower","Total"])
         print(c)
```

	ReportingMonth	Entry/Exit	Upper/Lower	Total
0	01/01/2018 12:00:00 AM	Entry	Lower Level	1264344
1	01/01/2018 12:00:00 AM	Entry	Upper Level	1501903
2	01/01/2018 12:00:00 AM	Exit	Lower Level	1929900
3	01/01/2018 12:00:00 AM	Exit	Upper Level	994130
4	03/01/2018 12:00:00 AM	Entry	Lower Level	1342159
5	03/01/2018 12:00:00 AM	Entry	Upper Level	1559233
6	03/01/2018 12:00:00 AM	Exit	Lower Level	1934828
7	03/01/2018 12:00:00 AM	Exit	Upper Level	1039613
8	04/01/2018 12:00:00 AM	Entry	Lower Level	1329798
9	04/01/2018 12:00:00 AM	Entry	Upper Level	1517056
10	04/01/2018 12:00:00 AM	Exit	Lower Level	1917622
11	04/01/2018 12:00:00 AM	Exit	Upper Level	994651
12	05/01/2018 12:00:00 AM	Entry	Lower Level	1439131
13	05/01/2018 12:00:00 AM	Entry	Upper Level	1601307
14	05/01/2018 12:00:00 AM	Exit	Lower Level	2059938
15	05/01/2018 12:00:00 AM	Exit	Upper Level	1044608

In this example, we see that we want to adjust the table to use the values in the "ReportingMonth" column as distinct indexes for each row. To do this we must first simplify the table by concatenating the "Entry/Exit" and "Upper/Lower" columns into a "Type/Location" column. We can then use Panda's pivot command with the index, columns, and values parameters to manipulate the table. This results in our desired table with the "ReportingMonth" values used as the index.


```
In [13]: # Use Pandas to Concatenate Columns and Pivot Table
         c["Type/Location"] = c["Entry/Exit"] + " " + c["Upper/Lower"]
         c = c.pivot(index="ReportingMonth",columns="Type/Location", values="Total")
         print(c)
```

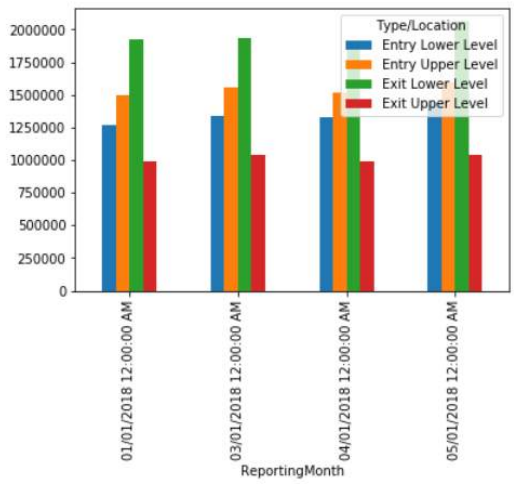
```
Type/Location           Entry Lower Level  Entry Upper Level  \
ReportingMonth
01/01/2018 12:00:00 AM            1264344            1501903
03/01/2018 12:00:00 AM            1342159            1559233
04/01/2018 12:00:00 AM            1329798            1517056
05/01/2018 12:00:00 AM            1439131            1601307

Type/Location           Exit Lower Level  Exit Upper Level
ReportingMonth
01/01/2018 12:00:00 AM           1929900            994130
03/01/2018 12:00:00 AM           1934828           1039613
04/01/2018 12:00:00 AM           1917622            994651
05/01/2018 12:00:00 AM           2059938           1044608
```

VISUALIZE

Finally, we are able to visualize the data after manipulating the table. We decide to use a bar graph to show the totals for each “Type/Location” by “ReportingMonth”. We simply use the Panda’s plot.bar function for this visualization. The visualization is now complete and shows the similarities of total vehicle counts for each location by the reported month!

```
In [15]: # Plot Data into Bar Graph
         c.plot.bar()
```

```
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x28c96d587b8>
```



REVISIONS

Version	Date	Name	Description
1.0.1	2020/04/07	Jenal Sanchez	Initial Release

