Table of Contents

[Sample 4.0 REST API Code Documentations 2](#_Toc135048649)

[Getting Started: 2](#_Toc135048650)

[Deliverables: 2](#_Toc135048651)

[General Steps: 2](#_Toc135048652)

[Sample Deliverables BreakDown: 5](#_Toc135048653)

[Configuring the pull requests 5](#_Toc135048654)

[SampleData.cfg: This is the main place to configure your data pull request. 5](#_Toc135048655)

[HOW TO 6](#_Toc135048656)

[Main Data Flow 6](#_Toc135048657)

[How to use some of the parameters: 6](#_Toc135048658)

[POST requests 7](#_Toc135048659)

[OUTPUT: 7](#_Toc135048660)

[Appendix2 9](#_Toc135048661)

# Sample 4.0 REST API Code Documentations

## Getting Started:

You will need your WellData log in information as well as the Application ID previously provided.

|  |  |
| --- | --- |
| Login Information | |
| AppID | Retrieve from WellData Customer Support |
| username | Retrieve from WellData Customer Support |
| password | Retrieve from WellData Customer Support |
| Swagger (UI Interface) | <https://data.welldata.net/swagger/index.html> |

## Deliverables:

**Main.py** 🡪 Main place to configure and pull data

**Sample\_WD\_API.py** 🡪 Definitions of methods to access API endpoints.

**SampleHelper.py** 🡪 All other helper code to make program run.

**Sample Data.cfg** 🡪 configuration file. All the necessary configs go here including your username and password.

These will be the general steps to getting started. After the general steps, we have a breakdown tutorial of the sample code and how it operates.

### General Steps:

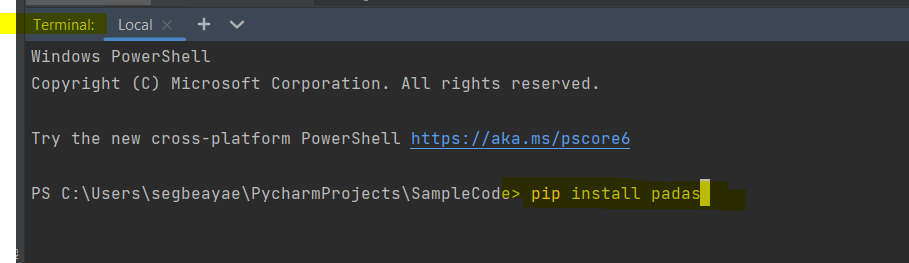
1. Once downloaded and opened to your favorite IDE, please pip install all necessary packages and imports.
   1. Run command:

* Top of each file has the required pip install needed :

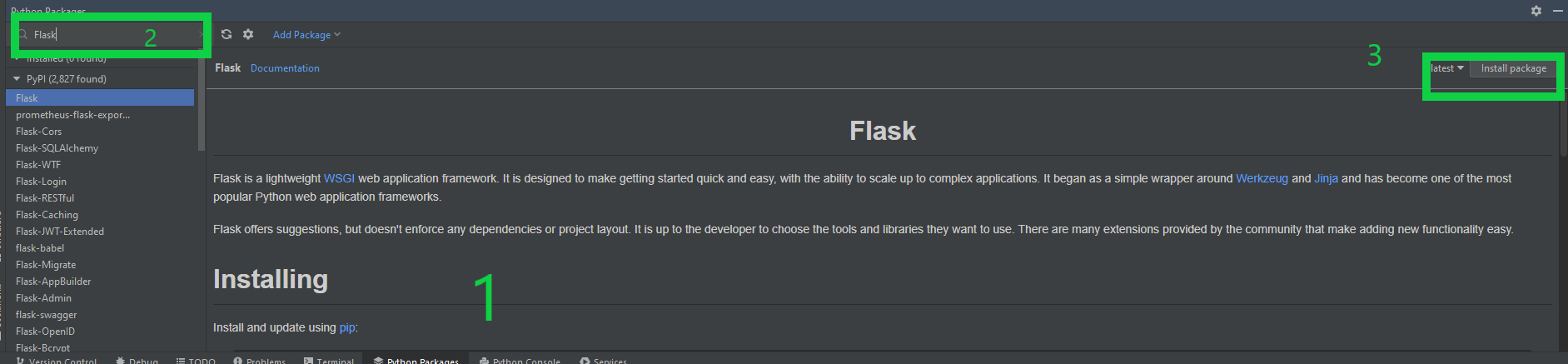
Let’s get started with pip installing these:

**Please install the following:**

* **pip install pandas**
* **pip install requests**
* **pip install tenacity**
* **pip install openpyxl**
* **pip install pydantic**
* **pip install sseclient**
* You can use terminal commands such as:



1. If you are using PyCharm as your IDE (any IDE): please go to the python package and download the packages above.



1. You will need to authenticate by getting a Token from the GET /API/v1/tokens/token endpoint using Basic Auth in the header.

**Note**: The token will expire after 2 hours, the token expiry is extended which each use. Your client should track and handle the expiry of your token.

1. You will need to identify your job (well) that you want to query data for. Use the GET /API/v1/jobs endpoint to grab your list of jobs (wells). Make note of the filter parameter jobStatus=ActiveJobs for those active wells.

The API offers access to a number of different types of data both TimeBased, DepthBased and Contextual. ***Attributes*** provides a listing of all of the Time and Depth series data available in the system.

1. After identifying the job you want, querying the Attributes endpoint /api/v1/jobs/{jobId}/attributes will return a listing of all the available data. The hasData field will tell you if data is available for the specific attribute.
2. To get the drilling summary data, use the POST api/v1/jobs/{jobId}/data/time/events endpoint.

**Note**: The body parameter is also required for this request. You will need outputAttributes, timeRange and a filter. {See [appendix](#_Appendices) 1}

## Sample Deliverables Breakdown:

**Main.py** : Main place to configure and pull data

Process:

1. Reads and sets up config.
2. Sets up URLs
3. Gets Token
4. Main Function calls:
   1. Gets Jobs
   2. Goes through list of jobs and gets active jobs
   3. Gets Attribute for each job that has Data
   4. Posts for time Based Data
   5. Post for CurrentTime Based Data
   6. Post for events
   7. Adds to List files
   8. Writes to excel spreadsheet
      1. \*\* See [appendix2](#_Appendix2) Below

**Sample\_WD\_API.py** 🡪 Definitions of methods to access API endpoints. Some of the useful methods to consider using are:

1. getToken – Gets a new token to implement the session
2. getApiCall – a generalize get method to retrieve most api calls
3. postApiCall- generalized post method to post and retrieve api calls
4. getJobs- useful method to get all jobs as well as individual jobs.

**SampleHelper.py** : Helper code to make program run.

1. SetConfigFile: creates and setup the config for the program based on the SampleData.cfg files
2. readConfig: reads the current configs from the SampleData.cfg file

**SampleData.cfg** 🡪 configuration file. All the necessary configs go here including your username and password.

Configuring the pull requests:

### SampleData.cfg: This is the main place to configure your data pull request.

Each parameter has a brief explanation:

The first section in SampleData.cfg contains your login information. Currently has it mapped to your username/password



Other parameters, gives you the expectations and the changes that may occur. The example shown sets the status of the Jobs you are looking to query for. In this case you’re querying for all active jobs.

## HOW TO

Main Data Flow:

Setup Credentials and Config File 🡪 Pull Jobs 🡪 getAttributes 🡪 Pull Historical/Current/Event Data 🡪 Write to Excel File.

### How to use some of the parameters:

Sample\_WD\_API.getJobs(URLs\_v1['getJobsId'], token, CFG, take=1000, jobId=w['id'], total=False, jobStatus="ActiveJobs" )

The above is a sample code to get the jobs by ID. These are some of the parameters to consider.

**jobStatus:** ActiveJobs/AllJobs/EndedJobs

**jobId:** wellid usually in the form of 🡪 net\_###### where # represent digits

**URLs\_v1**: this is where you specify the type of query you want. In this case we are taking jobs by ID. You can also check the Sample\_WD\_API.py to see the other types of URLS to query

**Total:** True/False 🡪 if true, will return the total jobs found

**Take:** numbers of jobs to query for.

POST requests**:**

Sample\_WD\_API.postTimeBased(URLs\_v1['postTimeBased'], token, CFG, data=json\_postValue, jobId=w['id'])

For post requests, we have the same as above, the only difference is that we have to convert our Post request into json format.

The fields below show the conversion from raw to json value per the main file.

|  |  |
| --- | --- |
| **RAW** | **JSON** |
| postValue | json\_postValue |
| timeCurrValue | json\_timeCurr |
| outputEventsValue | json\_outputEvents |

**\*\*Please note, you will only have to update the raw value to query for a post request.**

OUTPUT: **Excel File Explained**

|  |  |
| --- | --- |
| EXCEL SHEET | PURPOSE |
| **JobsID** | Gives you the Well information by ID for the wells beginning and ending as well as the data processing times. |
| **Attributes** | Pulls in the attribute list pertaining to each well |
| **Time Based** | Stores the Time based information based on the post request |
| **CurrentTIme** | Stores the Current time based information based on the post request |
| **Events** | Stores the Events pertaining to the well. This is based on the post request queried for. |
| **JobList** | Aggregation of previous information |

Appendices

Appendix 1:

{

    "outputAttributes": [

        {

            "id": "HoleDepth",

            "mode": "First"

        },

        {

            "id": "HoleDepth",

            "mode": "Last"

        },

        {

            "id": "AvgRopFtHr",

            "mode": "Average"

        },

        {

            "id": "BitWeightQualified",

            "mode": "Average"

        },

        {

            "id": "TopDrvRpm",

            "mode": "Average"

        },

        {

            "id": "FlowIn",

            "mode": "Average"

        },

        {

            "id": "PumpPressure",

            "mode": "Average"

        },

        {

            "id": "DiffPress",

            "mode": "Average"

        },

        {

            "id": "TopDrvTorque",

            "mode": "Average"

        },

        {

            "id": "MudMotorRpm",

            "mode": "Average"

        },

        {

            "id": "BitRpm",

            "mode": "Average"

        }

    ],

    "timeRange": {

        "from": "2022-11-13T12:30:00-06:00"

    },

    "filter": {

        "attributeId": "DbaRigstate",

        "isIn": {

            "values": [

                10,

                11

            ]

        }

    }

}

Appendix 1 Code Breakdown:

The filter used is an In filter which will match any value from a list provided it is a convenience method

that is equivalent to running multiple EqualTo filters and combining the results. In this case the filter is

using the DbaRigstate attribute with the values 10 (Rotary Drilling) or 11 (Slide Drilling). This filter

configuration will return all on-bottom drilling events detected by DBA.

The time range can be configured by the user to achieve the desired outcome, for example if you are

looking to collect the last 24 hours of events you can set it using the time range, the time should be

submitted in the rig time zone not the users time zone. We recommend for batch requests to delay the

request until after the backup cycle for data 1-4 hours depending on configuration, this reduces the risk

of missing data due to streaming intermittency.

The output attributes is an option field in the request that allows the user to add context to the events.

The Attribute Id is provided with the aggregation type, optionally the units of measure can be set. If no

units are set in the request the data will return in the rigs default display units.

**Return Payload --**

Data returned will have the Attributes listed, the Filters that were applied and the Events. The events

will contain the eventRange (from and to date/time stamp) and the values for the items listed in the

outputAttributes The parameters within the event object is a list, if the values are at different times they

will return with the correct timestamp starting December 2022.

## Appendix2

Please note: current program executes and delivers an excel spreadsheet.