pygmdata

Dave Borncamp

Jul 23, 2021

CONTENTS:

1 Installing pygmdata **3**
1.1 Requirements 3
1.2 Installing pygmdata 3
1.3 Verify Installation 3

2 PyGMData **5**
2.1 `pygmdata` 5

3 Indices and tables **13**

Index **15**

This documentation is autogenerated! This is the documentation page for the PyGMData library, the easy way to interact with Grey Matter Data via Python. Grey Matter Data is part of Decipher's technology stack. Documentation for Grey Matter Data can be found *here <https://docs.greymatter.io/v/1.3-beta/usage/platform-services/data>*

In order to use this library, Grey Matter Data must be running. Information on how to set up Grey Matter Data can be found *here <https://docs.greymatter.io/v/1.3-beta/reference/setup/platform-services/data>*

CHAPTER

ONE

INSTALLING PYGMDATA

1.1 Requirements

This library only supports Python 3.6+. Usage with any other version is not supported and if it works, it is purely by accident.

1.2 Installing pygmdata

Installing the pygmdata library is extremely easy. The package is currently in PyPi test and can be installed with

```
pip install -i https://test.pypi.org/simple/ pygmdata
```

1.3 Verify Installation

The installation can be verified by making sure that your instance of Grey Matter Data is running. Then running Python and hitting the `self` endpoint with the `get_self()` method:

```
>> from pygmdata import Data
>> d = pygmdata.Data("http://localhost:8181", USER_DN='CN=dave.borncamp,OU=Engineering,
↪O=Untrusted Example,L=Baltimore,ST=MD,C=US')
>> d.get_self()
'{"label":"CN=dave.borncamp,OU=Engineering,O=Untrusted Example,L=Baltimore,ST=MD,C=US",
↪"exp":1608285907,"iss":"greymatter.io","values":{"email":["dave.borncamp@greymatter.io
↪"],"org":["greymatter.io"]}}'
```

After that, you're all installed

CHAPTER TWO

PYGMDATA

The actual Grey Matter Data interface class for Python. This works by keeping an internal representation of the Data file structure in a flat store which has the resource and the OID. It then calls out to the Data API using REST and the OID for a target resource.

It supports using TLS connections to a mesh. In this case, a *USER_DN* is not expected to be in the headers as it will be over-written by the edge node and the DN from the cert will be used instead.

2.1 pygmdata

`class pygmdata.pygmdata.Data`(*base_url*, ***kwargs*)

Class to interact with GM-Data.

Parameters

- **base_url** – URL that Data lives at. All interactions will append to this URL to interact with Data (ex base_url + "/self")
- **kwargs** – Extra arguments to be supplied (case insensitive):
  - **USER_DN - Your USER_DN to be used for interacting with Data.** This will be added to the header of every request.
  - logfile - File to save the log to. If not specified
  - **log_level - Level of verbosity to log. Defaults to warning.** Can be integer or string.
  - **security - The default security policy to use. This can be** overidden when writing files. If not specified it will use:

    ```
    {"label": "DECIPHER//GMDATA",
     "foreground": "#FFFFFF",
     "background": "green"}
    ```

  - cert - Certificate to use in pem format.
  - key - keyfile to use in pem format.
  - trust - CA trust to use to make TLS connections.
  - **repopulate - A hack to get around changes that may have happened** in Data between file uploads and hierarchy updates

`append_data`(*data*, *data_filename*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)

Append the given filename with the given data in memory

Parameters

- `data` – Data to append to a file. Remember to add line endings if needed.
- `data_filename` – Target filename to update
- `object_policy` – optional - Object Policy to use. Will update an existing object with this value or will make a new object with this policy. If not supplied for either, it will make a best effort to come up with a good response
- `original_object_policy` – optional - Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.

Returns True on success

`append_file`(*local_filename*, *data_filename*, *object_policy=None*)

Append an uploaded file with another file on disk

Parameters

- `local_filename` – Filename on disk that will be appended to the data_filename
- `data_filename` – Filename to append the new file to
- `object_policy` – Object Policy to use. Will update an existing object with this value or will make a new object with this policy. If not supplied for either, it will make a best effort to come up with a good response

Returns True on success

`create_meta`(*data_filename*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)

Create the meta data for an object to be uploaded

Will determine if the action is to create or update the object and create all of the necessary metadata needed for making/updating it.

Parameters

- `data_filename` – The filename that will be used in Data
- `object_policy` – Object Policy to use. Will update an existing object with this value or will make a new object with this policy. If not supplied for either, it will make a best effort to come up with a good response
- `original_object_policy` – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.
- `kwargs` – extra keywords to be set: - security - The security tag of the given file. If not supplied it will keep what is already there or it will use the field from the parent if creating a new file. - mimetype - Mimetype to be used as a header value to be uploaded. If not supplied it will make it’s best guess at the value. - Anything that is in the allowable events. If updating a file these values will be used to superceed what is already in the props for that object. If creating a new file, *action*, *name*, *parentoid*, *isFile*, *originalobjectpolicy* (if supplied, see above for object_policy and original_object_policy), and *mimetype* (also see above) are all overwritten.

Returns Metadata dictionary

`delete_file`(*data_filename*, *oid=None*)

Delete a file from GM Data

Parameters

- `data_filename` – Path to the object to be deleted.
- `oid` – Object ID of the thing to properties of

Returns True on delete success, False on Failure

`download_file`(*file*, *local_filename*, *chunk_size=8192*)
Downloads a file onto the local file system.

Streams a file in chunks of 8192 to write the given file onto the filesystem. Streaming with chunks of this size can save lots of memory when downloading large files.

Parameters

- `file` – File within GM-Data to download
- `local_filename` – Filename to be written onto the local filesystem
- `chunk_size` – Size of chunks to be used. Defaults to 8192

Returns Written filename on success

`find_file`(*filename*)
Find a given file within the file hierarchy

Try to find a file within the file hierarchy, if it is not immediately found, repopulate the hierarchy and try again. If it is still not found, return None

Parameters `filename` – Filename to be found within GM Data

Returns The GM Data oid if found or None if not

`get_byte_steam`(*file*)
Get a file as a data stream into memory

Parameters `file` – File name within GM-Data to download

Returns bytestream of file contents

`get_config`()
Hit the */config* endpoint to probe how Data is setup

Returns json from the config endpoint

`get_derived`(*data_filename*, *oid=None*)
Get the derived files from a given filename

Parameters

- `data_filename` – Path to the object to be deleted.
- `oid` – Object ID of the thing to properties of):

Returns json of derived listing

`get_list`(*path*, *oid=None*)
Get the contents of a given path.

This gives the most recent tstamp by oid. The result is sorted by oid, and should only have one historical object per oid with highest tstamp.

Note: Listings of files will return None

Parameters

- `path` – Directory path that the object is nestled in.
- `oid` – Object ID of the thing to list

Returns json of listing if it exists, None if not

get_part(*data_filename*, *object_policy=None*, *original_object_policy=None*)
Get the file part append for a multi part file

Parameters

- **data_filename** – Filename in GM Data to append to
- **object_policy** – optional - Object Policy to use
- **original_object_policy** – optional - Original Object Policy to be used

Returns File part like 'aab'

get_props(*path=None*, *oid=None*)
Get the properties of a given Data object.

This essentially returns the metadata of a given object in Data.

Parameters

- **path** – Directory path that the object is nestled in.
- **oid** – Object ID of the thing to properties of, if this is supplied it ignores the path

Returns

json of properties if it exists, None if not.

```
{'tstamp': '1668e4d701ac18a4',
 'userpolicy': {'label': 'CN=dave.borncamp,OU=Engineering,O=Untrusted␣
↪Example,L=Baltimore,ST=MD,C=US'},
 'jwthash':
↪'368734e0d26fe381726932a727a04c9f4db9cca995e2341151d2c664e636b8f3',
 'schemaversion': 10,
 'name': 'dave.borncamp@greymatter.io',
 'action': 'C',
 'oid': '1668e4d701979c80',
 'parentoid': '1668e15c6792db54',
 'expiration': '7fffffffffffffff',
 'checkedtstamp': '1668e15c679cd280',
 'objectpolicy': {'requirements': {'f': 'if',
   'a': [{'f': 'contains',
   'a': [{'v': 'email'}, {'v': 'dave.borncamp@greymatter.io'}]},
    {'f': 'yield-all'},
    {'f': 'yield', 'a': [{'v': 'R'}, {'v': 'X'}]}]}},
 'derived': {},
 'security': {'label': 'DECIPHER//GMDATA',
  'foreground': '#FFFFFF',
  'background': 'green'},
 'originalobjectpolicy': '(if (contains email "dave.
↪borncamp@greymatter.io")(yield-all)(yield R X))',
 'policy': {'policy': ['R', 'X']},
 'cluster': 'default'}
```

get_self()
Hit GM Data's self endpoint.

Returns

Description of the user's credential token in the format of

```
{"label":USER_DN,"exp":1608262398,"iss":"greymatter.io",
"values":{"email":["dave.borncamp@greymatter.io"],
"org":["greymatter.io"]}}
```

get_self_identify(*object_policy=None*, *original_object_policy=None*)
Identify self and make user directory

Parameters

- **object_policy** – Object policy to use to create home directory. Make sure that "U" permissions are given, If not supplied it will try to use the policy of the root directory which may not grant the user "U" permissions.
- **original_object_policy** – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.

Returns True if successful

make_directory_tree(*path*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)
Recursively create directories in GM Data.

Parameters

- **path** – Path to be created in GM Data
- **object_policy** – A LISP statement of the Object Policy to be used for all folders that will be created in
- **kwargs** – extra keywords to be set: - security - The security tag of the given file. If not supplied it will keep what is already there or it will use the field from the parent if creating a new file.
- **original_object_policy** – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.

Returns oid on success

parse_events(***kwargs*)
Parse the given keyword arguments into a dictionary

Looks through the keywords to match up what is in allowable events, if an event is found add it to the returned meta

Returns metadata dictionary associated with the events in kwargs

populate_hierarchy(*path*, *refresh=True*)
Populate the internal hierarchy structure.

Every GM Data data object has an Object ID, including directories and files. This serves as a way to keep track of individual listings that can be easily accessed through an API call.

This function recursively searches the Data directory tree starting at the given oid and calls *list* on it.

Parameters

- **path** – Directory path that the object is nestled in. This will be prepended to the object's name and used as a key in the internal hierarchy dictionary. Always starts out as / then builds to */world* and so forth until the entire listing in Data is mapped.
- **refresh** – Delete the old hierarchy and start from scratch

post_write(*data*, *headers=None*)
Send a request to the */write* endpoint

Parameters

- `data` – The data to be uploaded, this needs to be a MultipartEncoder data type.
- `headers` – Any custom headers to be submitted. If this is not supplied it will use whatever is in self.headers. If it is supplied, it will only use those headers

Returns OID of object on write success, False on Failure

`set_log_level`(*level*)
Set the log level for the log

Parameters `level` – Level of verbosity to log. Defaults to warning. Can be integer or string.

Returns None

`static start_logger`(*name='pygmdata'*, *logfile=None*)
Start logging what is going on

Parameters

- `name` – Name of the logger to use. Defaults to "pygmdata"
- `logfile` – Name of output logfile. Default to not saving.

Returns logfile written to disk

`stream_file`(*file*)
Get a file loaded into memory.

Look at the Content-Type header and parse the returned variable accordingly:

- *image/** return a PIL image
- *application/json* return a dictionary in json format
- *text/plain* return decoded text of object
- Anything else return a buffer of the content

Parameters `file` – File name within GM-Data to download

Returns Object

`stream_upload`(*data_buf*, *data_filename*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)
Upload a file buffer from memory as a given filename

Parameters

- `data_buf` – Buffer of data to upload to a file.
- `data_filename` – Target filename to upload to
- `object_policy` – Object Policy to use. Will update an existing object with this value or will make a new object with this policy. If not supplied for either, it will make a best effort to come up with a good response.
- `original_object_policy` – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.

Returns True on success

`stream_upload_string`(*s*, *data_filename*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)
Upload a string into file from memory

Parameters

- `s` – Data to upload to a file.
- `data_filename` – Target filename to upload to
- `object_policy` – Object Policy to use. Will update an existing object with this value or will make a new object with this policy. If not supplied for either, it will make a best effort to come up with a good response
- `original_object_policy` – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.

Returns True on success

`upload_file`(*local_filename*, *data_filename*, *object_policy=None*, *original_object_policy=None*, ***kwargs*)
Upload a file from the local filesystem to GM-Data.

This will upload a file from the local file system to GM-Data. If the file already exists, it will update the file. If it does not exist, it will create a new file in the given directory in GM-Data.

Parameters

- `local_filename` – Filename to upload on the local filesystem
- `data_filename` – Filename of the destination in GM Data
- `object_policy` – Object Policy permissions for the file to have. If not supplied and updating a file, it will keep what is already in Data. If creating a new file and not supplied, it will likely fail as a file will be uploaded that cannot be accessed by anyone.
- `original_object_policy` – Field to be put into the originalobjectpolicy field. This can be lisp or OPA/Rego depending on the version of GM Data that is in use.
- `kwargs` – extra keywords to be set: - security - The security tag of the given file. If not supplied it will keep what is already there or it will use the field from the parent if creating a new file.

Returns False if request doesn't succeed or cannot be built True if it succeeds

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

append_data() (*pygmdata.pygmdata.Data method*), 5
append_file() (*pygmdata.pygmdata.Data method*), 6

C

create_meta() (*pygmdata.pygmdata.Data method*), 6

D

Data (*class in pygmdata.pygmdata*), 5
delete_file() (*pygmdata.pygmdata.Data method*), 6
download_file() (*pygmdata.pygmdata.Data method*), 7

F

find_file() (*pygmdata.pygmdata.Data method*), 7

G

get_byte_steam() (*pygmdata.pygmdata.Data method*), 7
get_config() (*pygmdata.pygmdata.Data method*), 7
get_derived() (*pygmdata.pygmdata.Data method*), 7
get_list() (*pygmdata.pygmdata.Data method*), 7
get_part() (*pygmdata.pygmdata.Data method*), 7
get_props() (*pygmdata.pygmdata.Data method*), 8
get_self() (*pygmdata.pygmdata.Data method*), 8
get_self_identify() (*pygmdata.pygmdata.Data method*), 9

M

make_directory_tree() (*pygmdata.pygmdata.Data method*), 9

P

parse_events() (*pygmdata.pygmdata.Data method*), 9
populate_hierarchy() (*pygmdata.pygmdata.Data method*), 9
post_write() (*pygmdata.pygmdata.Data method*), 9

S

set_log_level() (*pygmdata.pygmdata.Data method*), 10
start_logger() (*pygmdata.pygmdata.Data static method*), 10
stream_file() (*pygmdata.pygmdata.Data method*), 10
stream_upload() (*pygmdata.pygmdata.Data method*), 10
stream_upload_string() (*pygmdata.pygmdata.Data method*), 10

U

upload_file() (*pygmdata.pygmdata.Data method*), 11