CerebralCortex Documentation

Release 3.0.0

Nasir Ali

Aug 08, 2019

Contents:

Cerebral Cortex is the big data cloud companion of mCerebrum designed to support population-scale data analysis, visualization, model development, and intervention design for mobile sensor data.

You can find more information about MD2K software on our software website or the MD2K organization on our MD2K website.

CerebralCortex Kernel is part of our CerebralCortex cloud platform. CerebralCortex Kernel is mainly responsible to store/retrieve mobile sensor data along with it's metadata.

Note:

We have renamed following repositories.

- CerebralCortex-Platform -> CerebralCortex
- CerebralCortex - > CerebralCortex-Kernel

CHAPTER 1

Examples

- How to use CerebralCortex-Kernel API

CHAPTER 2

Documentation

CHAPTER 3

Installation

CerebralCortex-Kernel is a part of CerebralCortex cloud platform. To test the complete cloud platform, please visit CerebralCortex.

CerebralCortex-Kernel requires minimum Python3.6. To install CerebralCortex-Kernel as an API:

```
pip3 install cerebralcortex-kernel
```

- Note: please use appropriate pip (e.g., pip, pip3, pip3.6 etc.) installed on your machine

3.1 Dependencies

- `Python3.6`
- Note: Python3.7 is not compatible with some of the requirements
- Make sure pip version matches Python version

CHAPTER 4

FAQ

1 - Do I need whole CerebralCortex cloud platform to use CerebralCortex-Kernal?

No! If you want to use CerebralCortex-Kernel independently then you would need: * Backend storage (FileSystem/HDFS and MySQL) with some data. Here is some sample data to play with. * Setup the configurations * Use the examples to start exploring data

2 - How can I change NoSQL storage backend?

CerebralCortex-Kernel follows component based structure. This makes it easier to add/remove features. * Add a new class in Data manager-Raw. * New class must have read/write methods. Here is a sample skeleton class with mandatory methods required in the new class. * Create an object of new class in Data-Raw with appropriate parameters. * Add appropriate configurations in cerebralcortex.yml in (NoSQL Storage)[https://github.com/MD2Korg/CerebralCortex-Kernel/blob/master/conf/cerebralcortex.yml#L8] section.

3 - How can I replace MySQL with another SQL storage system?

- Add a new class in Data manager-SQL.
- New class must implement all of the methods available in (stream_handler.py)[https://github.com/MD2Korg/CerebralCortex-Kernel/blob/master/cerebralcortex/core/data_manager/sql/stream_handler.py] class.
- Create an object of new class in Data-SQL with appropriate parameters.
- Add appropriate configurations in cerebralcortex.yml in (Relational Storage)[https://github.com/MD2Korg/CerebralCortex-Kernel/blob/master/conf/cerebralcortex.yml#L31] section.

4 - Where are all the backend storage related classes/methods?

In Data manager-Raw. You can add/change any backend storage.

CHAPTER 5

Contributing

Please read our Contributing Guidelines for details on the process for submitting pull requests to us.

We use the Python PEP 8 Style Guide.

Our Code of Conduct is the Contributor Covenant.

Bug reports can be submitted through JIRA.

Our discussion forum can be found here.

CHAPTER 6

Versioning

We use Semantic Versioning for versioning the software which is based on the following guidelines.

MAJOR.MINOR.PATCH (example: 3.0.12)

1. MAJOR version when incompatible API changes are made,
2. MINOR version when functionality is added in a backwards-compatible manner, and
3. PATCH version when backwards-compatible bug fixes are introduced.

For the versions available, see this repository's tags.

CHAPTER 7

Contributors

Link to the list of contributors who participated in this project.

CHAPTER 8

License

This project is licensed under the BSD 2-Clause - see the license file for details.

CHAPTER 9

Acknowledgments

- National Institutes of Health - Big Data to Knowledge Initiative
- Grants: R01MD010362, 1UG1DA04030901, 1U54EB020404, 1R01CA190329, 1R01DE02524, R00MD010468, 3UH2DA041713, 10555SC
- National Science Foundation
- Grants: 1640813, 1722646
- Intelligence Advanced Research Projects Activity
- Contract: 2017-17042800006

9.1 CerebralCortex Core

9.1.1 Subpackages

cerebralcortex.core package

Subpackages

cerebralcortex.core.config_manager package

Submodules

cerebralcortex.core.config_manager.config module

class Configuration(*config_dir: str*, *config_file_name: str = 'cerebralcortex.yml'*)

Bases: *cerebralcortex.core.config_manager.config_handler.ConfigHandler*

cerebralcortex.core.config_manager.config_handler module

class ConfigHandler
Bases: object

load_file(*filepath: str*)
Helper method to load a yaml file :param config_dir_path:

Parameters **filepath** (*str*) – path to a yml configuration file

Module contents

cerebralcortex.core.data_manager package

Subpackages

cerebralcortex.core.data_manager.object package

Submodules

cerebralcortex.core.data_manager.object.data module

class ObjectData(*CC*)
Bases: *cerebralcortex.core.data_manager.object.storage_filesystem.FileSystemStorage*

cerebralcortex.core.data_manager.object.storage_filesystem module

class FileSystemStorage
Bases: object

create_bucket(*bucket_name: str*) → bool
creates a bucket aka folder in object storage system.

Parameters **bucket_name** (*str*) – name of the bucket

Returns True if bucket was successfully created. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises ValueError – Bucket name cannot be empty/None.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.create_bucket("live_data_folder")
>>> True
```

get_bucket_objects(*bucket_name: str*) → dict
returns a list of all objects stored in the specified Minio bucket

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns {bucket-objects: [{"object_name":"", "metadata": {}}...], in case of an error {"error": str}

Return type dict

get_buckets() → dict
returns all available buckets in an object storage

Returns {bucket-name: str, [{"key":"value"}]}, in case of an error {"error": str}

Return type dict

get_object(*bucket_name: str*, *object_name: str*) → dict
Returns stored object (HttpResponse) :param bucket_name: :param object_name: :return: object (HttpResponse), in case of an error {"error": str}

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object that needs to be downloaded

Returns object that needs to be downloaded. If file does not exists then it returns an error {"error": "File does not exist."}

Return type file-object

Raises

- ValueError – Missing bucket_name and object_name params.
- Exception – {"error": "error-message"}

get_object_stats(*bucket_name: str*, *object_name: str*) → dict
Returns properties (e.g., object type, last modified etc.) of an object stored in a specified bucket

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object

Returns information of an object (e.g., creation_date, object_size etc.). In case of an error {"error": str}

Return type dict

Raises

- ValueError – Missing bucket_name and object_name params.
- Exception – {"error": "error-message"}

is_bucket(*bucket_name: str*) → bool
checks whether a bucket exist :param bucket_name: name of the bucket aka folder :type bucket_name: str

Returns True if bucket exist or False otherwise. In case an error {"error": str}

Return type bool

Raises ValueError – bucket_name cannot be None or empty.

is_object(*bucket_name: str*, *object_name: str*) → dict
checks whether an object exist in a bucket :param bucket_name: name of the bucket aka folder :type bucket_name: str :param object_name: name of the object :type object_name: str

Returns True if object exist or False otherwise. In case an error {"error": str}

Return type bool

Raises `Exeception` – if bucket_name and object_name are empty or None

upload_object (*bucket_name: str*, *object_name: str*, *object_filepath: str*) → bool
Upload an object in a bucket aka folder of object storage system.

Parameters

- **bucket_name** (*str*) – name of the bucket
- **object_name** (*str*) – name of the object to be uploaded
- **object_filepath** (*str*) – it shall contain full path of a file with file name (e.g., /home/nasir/obj.zip)

Returns True if object successfully uploaded. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises `ValueError` – Bucket name cannot be empty/None.

cerebralcortex.core.data_manager.object.storage_minio module

class MinioHandler
Bases: `object`

Todo: For now, Minio is disabled as CC config doesn't provide an option to use mutliple object-storage

create_bucket (*bucket_name: str*) → bool
creates a bucket aka folder in object storage system.

Parameters **bucket_name** (*str*) – name of the bucket

Returns True if bucket was successfully created. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises `ValueError` – Bucket name cannot be empty/None.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.create_bucket("live_data_folder")
>>> True
```

get_bucket_objects (*bucket_name: str*) → dict
returns a list of all objects stored in the specified Minio bucket

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns {bucket-objects: [{"object_name":"", "metadata": {}}...], in case of an error {"error": str}

Return type dict

get_buckets () → List
returns all available buckets in an object storage

Returns {bucket-name: str, [{"key":"value"}]}, in case of an error {"error": str}

Return type dict

get_object(*bucket_name: str*, *object_name: str*) → dict
Returns stored object (HttpResponse) :param bucket_name: :param object_name: :return: object (HttpResponse), in case of an error {"error": str}

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object that needs to be downloaded

Returns object that needs to be downloaded. If file does not exists then it returns an error {"error": "File does not exist."}

Return type file-object

Raises

- ValueError – Missing bucket_name and object_name params.
- Exception – {"error": "error-message"}

get_object_stats(*bucket_name: str*, *object_name: str*) → dict
Returns properties (e.g., object type, last modified etc.) of an object stored in a specified bucket

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object

Returns information of an object (e.g., creation_date, object_size etc.). In case of an error {"error": str}

Return type dict

Raises

- ValueError – Missing bucket_name and object_name params.
- Exception – {"error": "error-message"}

is_bucket(*bucket_name: str*) → bool
checks whether a bucket exist :param bucket_name: name of the bucket aka folder :type bucket_name: str

Returns True if bucket exist or False otherwise. In case an error {"error": str}

Return type bool

Raises ValueError – bucket_name cannot be None or empty.

is_object(*bucket_name: str*, *object_name: str*) → dict
checks whether an object exist in a bucket :param bucket_name: name of the bucket aka folder :type bucket_name: str :param object_name: name of the object :type object_name: str

Returns True if object exist or False otherwise. In case an error {"error": str}

Return type bool

Raises Excecption – if bucket_name and object_name are empty or None

upload_object(*bucket_name: str*, *object_name: str*, *object_filepath: object*) → bool
Upload an object in a bucket aka folder of object storage system.

Parameters

- **bucket_name** (*str*) – name of the bucket
- **object_name** (*str*) – name of the object to be uploaded
- **object_filepath** (*str*) – it shall contain full path of a file with file name (e.g., /home/nasir/obj.zip)

Returns True if object successfully uploaded. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises

- `ValueError` – Bucket name cannot be empty/None.
- `Exception` – if upload fails

upload_object_to_s3(*bucket_name: str*, *object_name: str*, *file_data: object*, *obj_length: int*) → bool

Upload an object in a bucket aka folder of object storage system.

Parameters

- **bucket_name** (*str*) – name of the bucket
- **object_name** (*str*) – name of the object to be uploaded
- **file_data** (*object*) – object of a file
- **obj_length** (*int*) – size of an object

Returns True if object successfully uploaded. On failure, throws an exception

Return type bool

Raises `Exception` – if upload fails

Module contents

cerebralcortex.core.data_manager.raw package

Submodules

cerebralcortex.core.data_manager.raw.data module

class RawData(*CC*)

Bases: *cerebralcortex.core.data_manager.raw.stream_handler.StreamHandler*, *cerebralcortex.core.data_manager.raw.storage_hdfs.HDFSStorage*, *cerebralcortex.core.data_manager.raw.storage_filesystem.FileSystemStorage*

cerebralcortex.core.data_manager.raw.storage_blueprint module

class BlueprintStorage(*obj*)

Bases: `object`

This is a sample reference class. If you want to add another storage layer then the class must have following methods in it. read_file() write_file()

read_file(*stream_name: str*, *version: str = 'all'*) → object
Get stream data from storage system. Data would be return as pyspark DataFrame object :param stream_name: name of a stream :type stream_name: str :param version: version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all") :type version: str

Returns pyspark DataFrame object

Return type object

Raises `Exception` – if stream name does not exist.

write_file(*stream_name: str*, *data: cerebralcortex.core.datatypes.datastream.DataStream*) → bool
Write pyspark DataFrame to a data storage system :param stream_name: name of the stream :type stream_name: str :param data: pyspark DataFrame object :type data: object

Returns True if data is stored successfully or throws an Exception.

Return type bool

Raises `Exception` – if DataFrame write operation fails

cerebralcortex.core.data_manager.raw.storage_filesystem module

class FileSystemStorage(*obj*)
Bases: `object`

read_file(*stream_name: str*, *version: str = 'all'*, *user_id: str = None*) → object
Get stream data from storage system. Data would be return as pyspark DataFrame object :param stream_name: name of a stream :type stream_name: str :param version: version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all") :type version: str :param user_id: id of a user :type user_id: str

---

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Returns pyspark DataFrame object

Return type object

Raises `Exception` – if stream name does not exist.

write_file(*stream_name: str*, *data: <property object at 0x7f31760ab4f8>*, *file_mode*) → bool
Write pyspark DataFrame to a file storage system

Parameters

- **stream_name** (*str*) – name of the stream
- **data** (*object*) – pyspark DataFrame object

Returns True if data is stored successfully or throws an Exception.

Return type bool

Raises `Exception` – if DataFrame write operation fails

write_pandas_dataframe(*stream_name*, *data*)

write_spark_dataframe(*stream_name*, *data*, *file_mode*)

cerebralcortex.core.data_manager.raw.storage_hdfs module

class HDFSStorage(*obj*)

Bases: object

read_file(*stream_name: str*, *version: str = 'all'*, *user_id: str = None*) → object

Get stream data from storage system. Data would be return as pyspark DataFrame object :param stream_name: name of a stream :type stream_name: str :param version: version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all") :type version: str :param user_id: id of a user :type user_id: str

---

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Returns pyspark DataFrame object

Return type object

Raises Exception – if stream name does not exist.

write_file(*stream_name: str*, *data: <property object at 0x7f31760ab4f8>*) → bool

Write pyspark DataFrame to HDFS

Parameters

- **stream_name** (*str*) – name of the stream
- **data** (*object*) – pyspark DataFrame object

Returns True if data is stored successfully or throws an Exception.

Return type bool

Raises Exception – if DataFrame write operation fails

write_pandas_dataframe(*stream_name*, *data*)

write_spark_dataframe(*stream_name*, *data*)

cerebralcortex.core.data_manager.raw.stream_handler module

class DataSet

Bases: enum.Enum

An enumeration.

COMPLETE = (1,)

ONLY_DATA = (2,)

ONLY_METADATA = 3

class StreamHandler

Bases: object

get_stream(*stream_name: str*, *version: str*, *user_id: str = None*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream

Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")
- **user_id** (*str*) – id of a user
- **data_type** (DataSet) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises ValueError – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

save_stream(*datastream*, *file_mode='append'*, *ingestInfluxDB=False*, *publishOnKafka=False*) → bool

Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (DataStream) – a DataStream object
- **ingestInfluxDB** (*bool*) – Setting this to True will ingest the raw data in InfluxDB as well that could be used to visualize data in Grafana

Returns True if stream is successfully stored or throws an exception

Return type bool

Todo: Add functionality to store data in influxdb.

Raises Exception – log or throws exception if stream is not stored

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

Module contents

cerebralcortex.core.data_manager.sql package

Submodules

cerebralcortex.core.data_manager.sql.cache_handler module

class CacheHandler
Bases: object

get_cache_value(*key: str*) → str
Retrieves value from the cache for the given key.

Parameters **key** – key in the cache

Returns The value in the cache

Return type str

Raises ValueError – if key is None or empty

set_cache_value(*key: str*, *value: str*) → bool
Creates a new cache entry in the cache. Values are overwritten for existing keys.

Parameters

- **key** – key in the cache
- **value** – value associated with the key

Returns True on successful insert or False otherwise.

Return type bool

Raises ValueError – if key is None or empty

cerebralcortex.core.data_manager.sql.data module

class SqlData(*CC*)
Bases: *cerebralcortex.core.data_manager.sql.stream_handler.StreamHandler*, *cerebralcortex.core.data_manager.sql.users_handler.UserHandler*, *cerebralcortex.core.data_manager.sql.kafka_offsets_handler.KafkaOffsetsHandler*, *cerebralcortex.core.data_manager.sql.cache_handler.CacheHandler*, *cerebralcortex.core.data_manager.sql.data_ingestion_handler.DataIngestionHandler*, cerebralcortex.core.data_manager.sql.metadata_handler.MetadataHandler

close(*conn*, *cursor*)
close connection of mysql.

Parameters

- **conn** (*object*) – MySQL connection object
- **cursor** (*object*) – MySQL cursor object

Raises Exception – if connection is closed

create_pool(*pool_name: str = 'CC_Pool'*, *pool_size: int = 1*)
Create a connection pool, after created, the request of connecting MySQL could get a connection from this pool instead of request to create a connection.

Parameters

- **pool_name** (*str*) – the name of pool, (default="CC_Pool")
- **pool_size** (*int*) – size of MySQL connections pool (default=1)

Returns MySQL connections pool

Return type object

execute(*sql*, *args=None*, *commit=False*, *executemany=False*) → List[dict]
Execute a sql, it could be with args and with out args. The usage is similar with execute() function in module pymysql.

Parameters

- **sql** (*str*) – sql clause
- **args** (*tuple*) – args need by sql clause
- **commit** (*bool*) – whether to commit
- **executemany** (*bool*) – execute batch

Returns returns a list of dicts if commit is set to False

Return type list[dict]

Raises Exception – if MySQL query fails

cerebralcortex.core.data_manager.sql.data_ingestion_handler module

class DataIngestionHandler
Bases: object

add_ingestion_log(*user_id: str = ''*, *stream_name: str = ''*, *file_path: str = ''*, *fault_type: str = ''*, *fault_description: str = ''*, *success: int = None*, *metadata=None*) → bool
Log errors and success of each record during data import process.

Parameters

- **user_id** (*str*) – id of a user
- **stream_name** (*str*) – name of a stream
- **file_path** (*str*) – filename with its path
- **fault_type** (*str*) – error type
- **fault_description** (*str*) – error details
- **success** (*int*) – 1 if data was successfully ingested, 0 otherwise
- **metadata** (*dict*) – (optional) metadata of a stream

Returns bool

Raises

- ValeError – if
- Exception – if sql query fails user_id, file_path, fault_type, or success parameters is missing

add_scanned_files(*user_id: str*, *stream_name: str*, *metadata: dict*, *files_list: list*) → bool
Add scanned files in ingestion log table that could be processed later on. This method is specific to MD2K data ingestion.

Parameters

- **user_id** (*str*) – id of a user
- **stream_name** (*str*) – name of a stream
- **metadata** (*dict*) – raw metadata
- **files_list** (*list*) – list of filenames with its path

Returns bool

Raises Exception – if sql query fails

get_files_list(*stream_name: str = None*, *user_id=None*, *success_type=None*) → list
Get a list of all the processed/un-processed files

Returns list of all processed files list

Return type list

get_ingestion_stats() → list
Get stats on ingested records

Returns {"fault_type": str, "total_faults": int, "success":int}

Return type dict

get_processed_files_list(*success_type=False*) → list
Get a list of all the processed/un-processed files

Returns list of all processed files list

Return type list

is_file_processed(*filename: str*) → bool
check if a file is processed and ingested

Returns True if file is already processed

Return type bool

update_ingestion_log(*file_path: str = ''*, *fault_type: str = ''*, *fault_description: str = ''*, *success: int = None*) → bool
update ingestion Logs of each record during data import process.

Parameters

- **file_path** (*str*) – filename with its path
- **fault_type** (*str*) – error type
- **fault_description** (*str*) – error details
- **success** (*int*) – 1 if data was successfully ingested, 0 otherwise

Returns bool

Raises

- ValeError – if
- Exception – if sql query fails user_id, file_path, fault_type, or success parameters is missing

update_ingestion_log_status(*stream_name*, *fault_type*, *fault_description*, *status_type*, *metadata={}*, *platform_metadata={}*)

update_ingestion_log_status_ignore(*stream_name*, *fault_type*, *fault_description*, *status_type*, *metadata=None*)

cerebralcortex.core.data_manager.sql.kafka_offsets_handler module

class KafkaOffsetsHandler

Bases: object

get_kafka_offsets(*topic: str*) → List[dict]

Get last stored kafka offsets

Parameters **topic** (*str*) – kafka topic name

Returns list of kafka offsets. This method will return empty list if topic does not exist and/or no offset is stored for the topic.

Return type list[dict]

Raises ValueError – Topic name cannot be empty/None

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_kafka_offsets("live-data")
>>> [{"id","topic", "topic_partition", "offset_start", "offset_until",
↪"offset_update_time"}]
```

store_or_update_Kafka_offset(*topic: str*, *topic_partition: str*, *offset_start: str*, *offset_until: str*) → bool

Store or Update kafka topic offsets. Offsets are used to track what messages have been processed.

Parameters

- **topic** (*str*) – name of the kafka topic
- **topic_partition** (*str*) – partition number
- **offset_start** (*str*) – starting of offset
- **offset_until** (*str*) – last processed offset

Raises

- ValueError – All params are required.
- Exception – Cannot add/update kafka offsets because ERROR-MESSAGE

Returns returns True if offsets are add/updated or throws an exception.

Return type bool

cerebralcortex.core.data_manager.sql.stream_handler module

class StreamHandler

Bases: object

get_stream_info_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns stream metadata and other info related to a stream

Return type dict

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> {"name": .....} # stream metadata and other information
```

get_stream_metadata(*stream_name: str*, *version: str = 'all'*) → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type list (*Metadata*)

Raises ValueError – stream_name cannot be None or empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [Metadata] # list of MetaData class objects
```

get_stream_metadata_hash(*stream_name: str*) → List[str]

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes

Return type list[str]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> ["00ab666c-afb8-476e-9872-6472b4e66b68", "15cc444c-dfb8-676e-3872-
↪8472b4e66b12"]
```

(continues on next page)

(continued from previous page)

get_stream_name(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions(*stream_name: str*) → list

Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises ValueError – if stream_name is empty or None

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

is_stream(*stream_name: str*) → bool

Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

list_streams() → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[*Metadata*]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.list_streams()
```

save_stream_metadata(*metadata_obj*) → dict

Update a record if stream already exists or insert a new record otherwise.

Parameters **metadata_obj** (Metadata) – stream metadata

Returns {"status": True/False,"verion":version}

Return type dict

Raises Exception – if fail to insert/update record in MySQL. Exceptions are logged in a log file

search_stream(*stream_name*)

Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

cerebralcortex.core.data_manager.sql.users_handler module

class UserHandler

Bases: object

create_user(*username: str*, *user_password: str*, *user_role: str*, *user_metadata: dict*, *user_settings: dict*) → bool

Create a user in SQL storage if it doesn't exist

Parameters

- **username** (*str*) – Only alphanumeric usernames are allowed with the max length of 25 chars.
- **user_password** (*str*) – no size limit on password
- **user_role** (*str*) – role of a user
- **user_metadata** (*dict*) – metadata of a user
- **user_settings** (*dict*) – user settings, mCerebrum configurations of a user

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- `ValueError` – if selected username is not available
- `Exception` – if sql query fails

delete_user(*username: str*)
Delete a user record in SQL table

Parameters **username** – username of a user that needs to be deleted

Returns if user is successfully removed

Return type bool

Raises

- `ValueError` – if username param is empty or None
- `Exception` – if sql query fails

encrypt_user_password(*user_password: str*) → str
Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises `ValueError` – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str*, *size: int = 8*) → str
Generate a random password

Parameters

- **string_type** – Accepted parameters are "varchar" and "char". (Default="varchar")
- **size** – password length (default=8)

Returns random password

Return type str

get_all_users(*study_name: str*) → List[dict]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study

Raises `ValueError` – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[dict]

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

get_user_id(*user_name: str*) → str
Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises ValueError – User name is a required field.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*, *username: str = None*) → dict
Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises ValueError – User ID/name cannot be empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf"........}
```

get_user_name(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises ValueError – User ID is a required field.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_name("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict
Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf".....}]
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool
Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises `ValueError` – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_user(*user_id: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*, *user_name: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'> = None*) → bool
Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises `ValueError` – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

login_user(*username: str*, *password: str*, *encrypted_password: bool = False*) → dict

Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypted_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises ValueError – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises ValueError – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

username_checks(*username: str*)

No space, special characters, dash etc. are allowed in username. Only alphanumeric usernames are allowed with the max length of 25 chars.

Parameters **username** (*str*) –

Returns True if provided username comply the standard or throw an exception

Return type bool

Raises `Exception` – if username doesn't follow standards

Module contents

cerebralcortex.core.data_manager.time_series package

Submodules

cerebralcortex.core.data_manager.time_series.data module

class TimeSeriesData(*CC*)
Bases: *cerebralcortex.core.data_manager.time_series.influxdb_handler.InfluxdbHandler*

cerebralcortex.core.data_manager.time_series.influxdb_handler module

class InfluxdbHandler
Bases: `object`

save_data_to_influxdb(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*)
Save data stream to influxdb only for visualization purposes.

Parameters **datastream** (`DataStream`) – a DataStream object

Returns True if data is ingested successfully or False otherwise

Return type bool

Todo: This needs to be updated with the new structure. Should metadata be stored or not?

Example

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_data_to_influxdb(ds)
```

Module contents

Module contents

cerebralcortex.core.datatypes package

Submodules

cerebralcortex.core.datatypes.datastream module

class DataStream(*data: object = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*)

Bases: `object`

collect()

Collect all the data to master node and return list of rows

Returns rows of all the dataframe

Return type List

compute(*udfName*, *timeInterval=None*)

compute_average(*windowDuration: int = None*, *colmnName: str = None*) → object

Window data and compute average of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – average will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_max(*windowDuration: int = None*, *colmnName: str = None*) → object

Window data and compute max of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – max will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_min(*windowDuration: int = None*, *colmnName: str = None*) → object

Window data and compute min of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – min value will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_sqrt(*windowDuration: int = None*, *colmnName: str = None*) → object

Window data and compute square root of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – square root will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_stddev(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute standard deviation of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – standard deviation will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_sum(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute sum of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – average will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_variance(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute variance of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – variance will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

create_windows(*window_length='hour'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

data
get stream data

Returns (DataFrame):

drop_column (**args*, ***kwargs*)
calls deafult dataframe drop

Parameters

- ***args** –
- ****kwargs** –

filter (*columnName*, *operator*, *value*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_user (*user_ids: List*)
filter data to get only selective users' data

Parameters **user_ids** (*List[str]*) – list of users' UUIDs

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_version (*version: List*)
filter data to get only selective users' data

Parameters **version** (*List[str]*) – list of stream versions

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Todo: Metadata version should be return with the data

---

get_metadata (*version: int = None*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
get stream metadata

Parameters **version** (*int*) – version of a stream

Returns single version of a stream

Return type *Metadata*

Raises Exception – if specified version is not available for the stream

groupby (**columnName*)
Group data by column name :param columnName: name of the column to group by with :type columnName: str

Returns:

join(*dataStream*, *propagation='forward'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

limit(**args*, ***kwargs*)
calls deafult dataframe limit

Parameters

- ***args** –
- ****kwargs** –

map_stream(*window_ds*)
Map/join a stream to a windowed stream

Parameters **window_ds** (*Datastream*) – windowed datastream object

Returns joined/mapped stream

Return type Datastream

metadata
return stream metadata

Returns

Return type *Metadata*

plot(*y_axis_column=None*)

plot_gps_cords(*zoom=5*)

plot_hist(*x_axis_column=None*)

plot_stress_bar(*x_axis_column='stresser_main'*)

plot_stress_comparison(*x_axis_column='stresser_main'*, *usr_id=None*, *compare_with='all'*)

plot_stress_gantt()

plot_stress_pie(*x_axis_column='stresser_main'*)

plot_stress_sankey(*cat_cols=['stresser_main', 'stresser_sub'], value_cols='density', title="Stressers' Sankey Diagram"*)

run_algorithm(*udfName*, *columnNames: List[str] = []*, *windowDuration: int = 60*, *slideDuration: int = None*, *groupByColumnName: List[str] = []*, *startTime=None*, *preserve_ts=False*)
Run an algorithm

Parameters

- **udfName** – Name of the algorithm

- **List[str]** (*groupByColumnName*) – column names on which windowing should be performed. Windowing will be performed on all columns if none is provided
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided
- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

schema()
Get data schema (e.g., column names and number of columns etc.)

Returns pyspark dataframe schema object

show(**args*, ***kwargs*)

sort(*columnNames: list = []*, *ascending=True*)
Sort data column in ASC or DESC order

Returns DataStream object

Return type object

summary()
print the summary of the data

to_pandas()
This method converts pyspark dataframe into pandas dataframe.

Notes

This method will collect all the data on master node to convert pyspark dataframe into pandas dataframe. After converting to pandas dataframe datastream objects helper methods will not be accessible.

Returns this will return a new datastream object with blank metadata

Return type Datastream (*Metadata*, pandas.DataFrame)

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("STREAM-NAME")
>>> new_ds = ds.to_pandas()
>>> new_ds.data.head()
```

where(**args*, ***kwargs*)
calls deafult dataframe where

Parameters

- ***args** –
- ****kwargs** –

window(*windowDuration: int = 60*, *groupByColumnName: List[str] = []*, *columnName: List[str] = []*, *slideDuration: int = None*, *startTime=None*, *preserve_ts=False*)

Window data into fixed length chunks. If no columnName is provided then the windowing will be performed on all the columns.

Parameters

- **windowDuration** (*int*) – duration of a window in seconds
- **List[str]** (*columnName*) – groupby column names, for example, groupby user, col1, col2
- **List[str]** – column names on which windowing should be performed. Windowing will be performed on all columns if none is provided
- **slideDuration** (*int*) – slide duration of a window
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided
- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Note: This windowing method will use collect_list to return values for each window. collect_list is not optimized.

---

get_window(*x*)

windowing_udf(*x*)

Module contents

class DataStream(*data: object = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*)

Bases: `object`

collect()

Collect all the data to master node and return list of rows

Returns rows of all the dataframe

Return type List

compute(*udfName*, *timeInterval=None*)

compute_average(*windowDuration: int = None*, *colmnName: str = None*) → object

Window data and compute average of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – average will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_max(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute max of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – max will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_min(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute min of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – min value will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_sqrt(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute square root of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – square root will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_stddev(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute standard deviation of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – standard deviation will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_sum(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute sum of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – average will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

compute_variance(*windowDuration: int = None*, *colmnName: str = None*) → object
Window data and compute variance of a windowed data of a single or all columns

Parameters

- **windowDuration** (*int*) – duration of a window in seconds. If it is not set then stats will be computed for the whole data in a column(s)
- **colmnName** (*str*) – variance will be computed for all the columns if columnName param is not provided (for all windows)

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

create_windows(*window_length='hour'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

data
get stream data

Returns (DataFrame):

drop_column(**args*, ***kwargs*)
calls deafult dataframe drop

Parameters

- ***args** –
- ****kwargs** –

filter(*columnName*, *operator*, *value*)
filter data

Parameters

- **columnName** (*str*) – name of the column

- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_user(*user_ids: List*)
filter data to get only selective users' data

Parameters **user_ids** (*List[str]*) – list of users' UUIDs

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

filter_version(*version: List*)
filter data to get only selective users' data

Parameters **version** (*List[str]*) – list of stream versions

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Todo: Metadata version should be return with the data

---

get_metadata(*version: int = None*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
get stream metadata

Parameters **version** (*int*) – version of a stream

Returns single version of a stream

Return type *Metadata*

Raises Exception – if specified version is not available for the stream

groupby(**columnName*)
Group data by column name :param columnName: name of the column to group by with :type columnName: str

Returns:

join(*dataStream*, *propagation='forward'*)
filter data

Parameters

- **columnName** (*str*) – name of the column
- **operator** (*str*) – basic operators (e.g., >, <, ==, !=)
- **value** (*Any*) – if the columnName is timestamp, please provide python datatime object

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

limit(**args*, ***kwargs*)
calls deafult dataframe limit

Parameters

- ***args** –

- ****kwargs** –

map_stream(*window_ds*)
Map/join a stream to a windowed stream

Parameters **window_ds** (*Datastream*) – windowed datastream object

Returns joined/mapped stream

Return type Datastream

metadata
return stream metadata

Returns

Return type *Metadata*

plot(*y_axis_column=None*)

plot_gps_cords(*zoom=5*)

plot_hist(*x_axis_column=None*)

plot_stress_bar(*x_axis_column='stresser_main'*)

plot_stress_comparison(*x_axis_column='stresser_main'*, *usr_id=None*, *compare_with='all'*)

plot_stress_gantt()

plot_stress_pie(*x_axis_column='stresser_main'*)

plot_stress_sankey(*cat_cols=['stresser_main', 'stresser_sub'], value_cols='density', title="Stressers' Sankey Diagram"*)

run_algorithm(*udfName*, *columnNames: List[str] = []*, *windowDuration: int = 60*, *slideDuration: int = None*, *groupByColumnName: List[str] = []*, *startTime=None*, *preserve_ts=False*)
Run an algorithm

Parameters

- **udfName** – Name of the algorithm
- **List[str]** (*groupByColumnName*) – column names on which windowing should be performed. Windowing will be performed on all columns if none is provided
- **windowDuration** (*int*) – duration of a window in seconds
- **slideDuration** (*int*) – slide duration of a window
- **List[str]** – groupby column names, for example, groupby user, col1, col2
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15... provide startTime as 15 minutes. First time of data will be used as startTime if none is provided
- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

schema()
Get data schema (e.g., column names and number of columns etc.)

Returns pyspark dataframe schema object

show(**args*, ***kwargs*)

sort(*columnNames: list = []*, *ascending=True*)
Sort data column in ASC or DESC order

Returns DataStream object

Return type object

summary()
print the summary of the data

to_pandas()
This method converts pyspark dataframe into pandas dataframe.

Notes

This method will collect all the data on master node to convert pyspark dataframe into pandas dataframe. After converting to pandas dataframe datastream objects helper methods will not be accessible.

Returns this will return a new datastream object with blank metadata

Return type Datastream (*Metadata*, pandas.DataFrame)

Examples

```
>>> CC = CerebralCortex("/directory/path/of/configs/")
>>> ds = CC.get_stream("STREAM-NAME")
>>> new_ds = ds.to_pandas()
>>> new_ds.data.head()
```

where(**args*, ***kwargs*)
calls deafult dataframe where

Parameters

- ***args** –
- ****kwargs** –

window(*windowDuration: int = 60*, *groupByColumnName: List[str] = []*, *columnName: List[str] = []*, *slideDuration: int = None*, *startTime=None*, *preserve_ts=False*)
Window data into fixed length chunks. If no columnName is provided then the windowing will be performed on all the columns.

Parameters

- **windowDuration** (*int*) – duration of a window in seconds
- **List[str]** (*columnName*) – groupby column names, for example, groupby user, col1, col2
- **List[str]** – column names on which windowing should be performed. Windowing will be performed on all columns if none is provided
- **slideDuration** (*int*) – slide duration of a window
- **startTime** (*datetime*) – The startTime is the offset with respect to 1970-01-01 00:00:00 UTC with which to start window intervals. For example, in order to have hourly tumbling windows that start 15 minutes past the hour, e.g. 12:15-13:15, 13:15-14:15...

provide startTime as 15 minutes. First time of data will be used as startTime if none is provided

- **preserve_ts** (*bool*) – setting this to True will return timestamps of corresponding to each windowed value

Returns this will return a new datastream object with blank metadata

Return type *DataStream*

Note: This windowing method will use collect_list to return values for each window. collect_list is not optimized.

---

cerebralcortex.core.log_manager package

Submodules

cerebralcortex.core.log_manager.log_handler module

class LogHandler
Bases: object

log(*error_message=''*, *error_type=(1,)*)

class LogTypes
Bases: object

CRITICAL = (2,)

DEBUG = 6

ERROR = (3,)

EXCEPTION = (1,)

MISSING_DATA = (5,)

WARNING = (4,)

cerebralcortex.core.log_manager.logging module

class CCLogging(*CC*)
Bases: *cerebralcortex.core.log_manager.log_handler.LogHandler*

Module contents

cerebralcortex.core.messaging_manager package

Submodules

cerebralcortex.core.messaging_manager.kafka_handler module

class KafkaHandler
Bases: object

create_direct_kafka_stream(*kafka_topic: str*, *ssc*) → pyspark.streaming.kafka.KafkaDStream
Create a direct stream to kafka topic. Supports only one topic at a time

Parameters **kafka_topic** – kafka topic to create stream against

Raises Exception – if direct stream cannot be created.

Todo: Enable logging of errors

produce_message(*topic: str*, *msg: str*)
Publish a message on kafka message queue

Parameters

- **topic** (*str*) – name of the kafka topic
- **msg** (*dict*) – message that needs to published on kafka

Returns True if successful. In case of failure, it returns an Exception message.

Return type bool

Raises

- ValueError – topic and message parameters cannot be empty or None.
- Exception – Error publishing message. Topic: topic_name - error-message

subscribe_to_topic(*topic: str*) → dict
Subscribe to kafka topic as a consumer

Parameters **topic** (*str*) – name of the kafka topic

Yields *dict* – kafka message

Raises ValueError – Topic parameter is missing.

cerebralcortex.core.messaging_manager.messaging_queue module

class MessagingQueue(*CC: object*, *auto_offset_reset: str = 'largest'*)
Bases: *cerebralcortex.core.messaging_manager.kafka_handler.KafkaHandler*

Module contents

cerebralcortex.core.metadata_manager package

Subpackages

cerebralcortex.core.metadata_manager.stream package

Submodules

cerebralcortex.core.metadata_manager.stream.data_descriptor module

class DataDescriptor
Bases: object

from_json(*obj*)
Cast DataDescriptor class object into json

Parameters **obj** (DataDescriptor) – object of a data descriptor class

Returns

Return type self

set_attribute(*key*, *value*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_name(*value*)
Name of data descriptor

Parameters **value** (*str*) – name

Returns

Return type self

set_type(*value: str*)
Type of a data descriptor

Parameters **value** (*str*) – type

Returns

Return type self

cerebralcortex.core.metadata_manager.stream.metadata module

class Metadata
Bases: object

add_annotation(*annotation: str*)
Add annotation stream name

Parameters **annotation** (*str*) – name of annotation stream

Returns self

add_dataDescriptor(*dd: cerebralcortex.core.metadata_manager.stream.data_descriptor.DataDescriptor*)
Add data description of a stream

Parameters **dd** (DataDescriptor) – data descriptor

Returns self

add_input_stream(*input_stream: str*)
Add input streams that were used to derive a new stream

Parameters **input_stream** (*str*) – name of input stream

Returns self

add_module(*mod: cerebralcortex.core.metadata_manager.stream.module_info.ModuleMetadata*)
Add module metadata

Parameters **mod** (ModuleMetadata) – module metadata

Returns self

from_json_file(*metadata: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

from_json_sql(*metadata_json: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

get_hash() → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Returns hash id of metadata

Return type str

get_hash_by_json(*metadata: dict = None*) → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Parameters **metadata** – only pass this if this method is used on a dict object outside of Metadata class

Returns hash id of metadata

Return type str

is_valid() → bool
check whether all required fields are set

Returns True if fields are set or throws an exception in case of missing values

Return type bool

Exception: ValueError: if metadata fields are not set

set_description(*stream_description: str*)
Add stream description

Parameters **stream_description** (*str*) – textual description of a stream

Returns self

set_name(*value: str*)
set name of a stream

Parameters **value** (*str*) – name of a stream

Returns self

to_json() → dict
Convert MetaData object into a dict (json) object

Returns dict form of MetaData object

Return type dict

cerebralcortex.core.metadata_manager.stream.module_info module

class ModuleMetadata
Bases: object

from_json(*obj*)
Cast ModuleMetadata class object into json

Parameters **obj** (ModuleMetadata) – object of a ModuleMetadata class

Returns

Return type self

set_attribute(*key: str*, *value: str*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises ValueError – if key/value are missing

set_author(*key*, *value*)
set author key/value pair. For example, key=name, value=md2k

Parameters

- **key** (*str*) – author metadata key
- **value** (*str*) – author metadata value

Returns

Return type self

set_name(*value*)
name of the module

Parameters **value** (*str*) – name

Returns

Return type self

set_version (*value*)
version of the module

Parameters **value** (*str*) – version

Returns

Return type self

Module contents

class Metadata
Bases: object

add_annotation (*annotation: str*)
Add annotation stream name

Parameters **annotation** (*str*) – name of annotation stream

Returns self

add_dataDescriptor (*dd: cerebralcortex.core.metadata_manager.stream.data_descriptor.DataDescriptor*)
Add data description of a stream

Parameters **dd** (DataDescriptor) – data descriptor

Returns self

add_input_stream (*input_stream: str*)
Add input streams that were used to derive a new stream

Parameters **input_stream** (*str*) – name of input stream

Returns self

add_module (*mod: cerebralcortex.core.metadata_manager.stream.module_info.ModuleMetadata*)
Add module metadata

Parameters **mod** (ModuleMetadata) – module metadata

Returns self

from_json_file (*metadata: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

from_json_sql (*metadata_json: dict*) → List
Convert dict (json) objects into Metadata class objects

Parameters **dict** (*json_list*) – metadata dict

Returns metadata class object

Return type *Metadata*

get_hash () → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Returns hash id of metadata

Return type str

get_hash_by_json(*metadata: dict = None*) → str
Get the unique hash of metadata. Hash is generated based on "stream-name + data_descriptor + module-metadata"

Parameters **metadata** – only pass this if this method is used on a dict object outside of Metadata class

Returns hash id of metadata

Return type str

is_valid() → bool
check whether all required fields are set

Returns True if fields are set or throws an exception in case of missing values

Return type bool

Exception: ValueError: if metadata fields are not set

set_description(*stream_description: str*)
Add stream description

Parameters **stream_description** (*str*) – textual description of a stream

Returns self

set_name(*value: str*)
set name of a stream

Parameters **value** (*str*) – name of a stream

Returns self

to_json() → dict
Convert MetaData object into a dict (json) object

Returns dict form of MetaData object

Return type dict

class DataDescriptor
Bases: object

from_json(*obj*)
Cast DataDescriptor class object into json

Parameters **obj** (DataDescriptor) – object of a data descriptor class

Returns

Return type self

set_attribute(*key*, *value*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises `ValueError` – if key/value are missing

set_name(*value*)
Name of data descriptor

Parameters **value** (*str*) – name

Returns

Return type self

set_type(*value: str*)
Type of a data descriptor

Parameters **value** (*str*) – type

Returns

Return type self

class ModuleMetadata
Bases: `object`

from_json(*obj*)
Cast ModuleMetadata class object into json

Parameters **obj** (`ModuleMetadata`) – object of a ModuleMetadata class

Returns

Return type self

set_attribute(*key: str*, *value: str*)
Attributes field is option in metadata object. Arbitrary number or attributes could be attached to a DataDescriptor

Parameters

- **key** (*str*) – key of an attribute
- **value** (*str*) – value of an attribute

Returns

Return type self

Raises `ValueError` – if key/value are missing

set_author(*key*, *value*)
set author key/value pair. For example, key=name, value=md2k

Parameters

- **key** (*str*) – author metadata key
- **value** (*str*) – author metadata value

Returns

Return type self

set_name(*value*)
name of the module

Parameters **value** (*str*) – name

Returns

Return type self

set_version(*value*)
version of the module

Parameters **value** (*str*) – version

Returns

Return type self

cerebralcortex.core.metadata_manager.user package

Submodules

cerebralcortex.core.metadata_manager.user.user module

class User(*user_id: uuid.UUID*, *username: str*, *password: str*, *token: str = None*, *token_issued_at: datetime.datetime = None*, *token_expiry: datetime.datetime = None*, *user_role: datetime.datetime = None*, *user_metadata: dict = None*, *active: bool = 1*)
Bases: object

isactive
user status

Type Returns (int)

password
encrypted password

Type Returns

Type (str)

token
auth token

Type Returns

Type (str)

token_expiry
date and time when token will expire

Type Returns

Type (datetime)

token_issued_at
date and time when token was issues

Type Returns

Type (datetime)

user_id
user id

Type Returns

Type (str)

user_metadata
metadata of a user

Type Returns (dict)

user_role
role

Type Returns (str)

username
user name

Type Returns

Type (str)

Module contents

Module contents

cerebralcortex.core.util package

Submodules

cerebralcortex.core.util.datetime_helper_methods module

get_timezone(*tz_offset: float*, *common_only: bool = False*)
Returns a timezone for a given offset in milliseconds

Parameters

- **tz_offset** (*float*) – in milliseconds
- **common_only** (*bool*) –

Returns timezone of an offset

Return type str

cerebralcortex.core.util.spark_helper module

get_or_create_sc(*type='sparkContext'*, *name='CerebralCortex-Kernal'*, *enable_spark_ui=False*)
get or create spark context

Parameters

- **type** (*str*) – type (sparkContext, SparkSessionBuilder, sparkSession, sqlContext). (default="sparkContext")
- **name** (*str*) – spark app name (default="CerebralCortex-Kernal")

Returns:

Module contents

Module contents

9.1.2 Submodules

9.1.3 cerebralcortex.kernel module

class Kernel (*configs_dir_path: str = None, auto_offset_reset: str = 'largest', enable_spark: bool = True, enable_spark_ui=False*)

Bases: `object`

connect (*username: str, password: str, encrypted_password: bool = False*) → dict

Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypted_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises `ValueError` – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

create_bucket (*bucket_name: str*) → bool

creates a bucket aka folder in object storage system.

Parameters **bucket_name** (*str*) – name of the bucket

Returns True if bucket was successfully created. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises `ValueError` – Bucket name cannot be empty/None.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.create_bucket("live_data_folder")
>>> True
```

create_user (*username: str, user_password: str, user_role: str, user_metadata: dict, user_settings: dict*) → bool

Create a user in SQL storage if it doesn't exist

Parameters

- **username** (*str*) – Only alphanumeric usernames are allowed with the max length of 25 chars.
- **user_password** (*str*) – no size limit on password
- **user_role** (*str*) – role of a user
- **user_metadata** (*dict*) – metadata of a user
- **user_settings** (*dict*) – user settings, mCerebrum configurations of a user

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- `ValueError` – if selected username is not available
- `Exception` – if sql query fails

delete_user(*username: str*) → bool
Delete a user record in SQL table

Parameters **username** – username of a user that needs to be deleted

Returns if user is successfully removed

Return type bool

Raises

- `ValueError` – if username param is empty or None
- `Exception` – if sql query fails

encrypt_user_password(*user_password: str*) → str
Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises `ValueError` – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str = 'varchar'*, *size: int = 8*) → str
Generate a random password

Parameters

- **string_type** – Accepted parameters are “varchar” and “char”. (Default=”varchar”)
- **size** – password length (default=8)

Returns random password

Return type str

get_all_users(*study_name: str*) → List[dict]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study

Raises `ValueError` – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[dict]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

get_bucket_objects(*bucket_name: str*) → dict
returns a list of all objects stored in the specified Minio bucket

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns {bucket-objects: [{"object_name":"", "metadata": {}}...], in case of an error {"error": str}

Return type dict

get_buckets() → dict
returns all available buckets in an object storage

Returns {bucket-name: str, [{"key":"value"}]}, in case of an error {"error": str}

Return type dict

get_cache_value(*key: str*) → str
Retrieves value from the cache for the given key.

Parameters **key** – key in the cache

Returns The value in the cache

Return type str

Raises `ValueError` – if key is None or empty

get_kafka_offsets(*topic: str*) → dict
Get last stored kafka offsets

Parameters **topic** (*str*) – kafka topic name

Returns list of kafka offsets. This method will return empty list if topic does not exist and/or no offset is stored for the topic.

Return type list[dict]

Raises `ValueError` – Topic name cannot be empty/None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_kafka_offsets("live-data")
>>> [{"id","topic", "topic_partition", "offset_start", "offset_until",
↪"offset_update_time"}]
```

get_object(*bucket_name: str*, *object_name: str*) → dict
Returns stored object (HttpResponse)

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object that needs to be downloaded

Returns object that needs to be downloaded. If file does not exists then it returns an error {"error": "File does not exist."}

Return type file-object

Raises

- `ValueError` – Missing bucket_name and object_name params.
- `Exception` – {"error": "error-message"}

get_object_stats(*bucket_name: str*, *object_name: str*) → dict

Returns properties (e.g., object type, last modified etc.) of an object stored in a specified bucket

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object

Returns information of an object (e.g., creation_date, object_size etc.). In case of an error {"error": str}

Return type dict

Raises

- `ValueError` – Missing bucket_name and object_name params.
- `Exception` – {"error": "error-message"}

get_stream(*stream_name: str*, *version: str = 'all'*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream

Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")
- **data_type** (`DataSet`) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises `ValueError` – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

get_stream_info_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns stream metadata and other info related to a stream

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> {"name": .....} # stream metadata and other information
```

get_stream_metadata(*stream_name: str*, *version: str = 'all'*) → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type list[*Metadata*]

Raises ValueError – stream_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [Metadata] # list of MetaData class objects
```

get_stream_metadata_hash(*stream_name: str*) → list

Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes

Return type list[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> ["00ab666c-afb8-476e-9872-6472b4e66b68", "15cc444c-dfb8-676e-3872-
↪8472b4e66b12"]
```

get_stream_name (*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions (*stream_name: str*) → list

Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises ValueError – if stream_name is empty or None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

get_user_id (*user_name: str*) → str

Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises ValueError – User name is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: str = None*, *username: str = None*) → dict
Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises ValueError – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf"........}
```

get_user_name(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises ValueError – User ID is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_name("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict
Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf".....}]
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool
Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises `ValueError` – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_bucket(*bucket_name: str*) → bool
checks whether a bucket exist

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns True if bucket exist or False otherwise. In case an error {"error": str}

Return type bool

Raises `ValueError` – bucket_name cannot be None or empty.

is_object(*bucket_name: str*, *object_name: str*) → bool
checks whether an object exist in a bucket

Parameters

- **bucket_name** (*str*) – name of the bucket aka folder
- **object_name** (*str*) – name of the object

Returns True if object exist or False otherwise. In case an error {"error": str}

Return type bool

Raises `Exeception` – if bucket_name and object_name are empty or None

is_stream(*stream_name: str*) → bool
Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

is_user (*user_id: str = None*, *user_name: str = None*) → bool
Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises ValueError – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

kafka_produce_message (*topic: str*, *msg: dict*)
Publish a message on kafka message queue

Parameters

- **topic** (*str*) – name of the kafka topic
- **msg** (*dict*) – message that needs to published on kafka

Returns True if successful. In case of failure, it returns an Exception message.

Return type bool

Raises

- ValueError – topic and message parameters cannot be empty or None.
- Exception – Error publishing message. Topic: topic_name - error-message

kafka_subscribe_to_topic (*topic: str*)
Subscribe to kafka topic as a consumer

Parameters **topic** (*str*) – name of the kafka topic

Yields *dict* – kafka message

Raises ValueError – Topic parameter is missing.

list_streams () → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]
Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[*Metadata*]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.list_streams()
```

save_data_to_influxdb(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*)
Save data stream to influxdb only for visualization purposes.

Parameters **datastream** (DataStream) – a DataStream object

Returns True if data is ingested successfully or False otherwise

Return type bool

Todo: This needs to be updated with the new structure. Should metadata be stored or not?

Example

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_data_to_influxdb(ds)
```

save_stream(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*, *ingestInfluxDB: bool = False*) → bool
Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (DataStream) – a DataStream object
- **ingestInfluxDB** (*bool*) – Setting this to True will ingest the raw data in InfluxDB as well that could be used to visualize data in Grafana

Returns True if stream is successfully stored or throws an exception

Return type bool

Raises Exception – log or throws exception if stream is not stored

Todo: Add functionality to store data in influxdb.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

search_stream(*stream_name*)
Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

set_cache_value(*key: str*, *value: str*) → bool

Creates a new cache entry in the cache. Values are overwritten for existing keys.

Parameters

- **key** – key in the cache
- **value** – value associated with the key

Returns True on successful insert or False otherwise.

Return type bool

Raises ValueError – if key is None or empty

store_or_update_Kafka_offset(*topic: str*, *topic_partition: str*, *offset_start: str*, *offset_until: str*) → bool

Store or Update kafka topic offsets. Offsets are used to track what messages have been processed.

Parameters

- **topic** (*str*) – name of the kafka topic
- **topic_partition** (*str*) – partition number
- **offset_start** (*str*) – starting of offset
- **offset_until** (*str*) – last processed offset

Raises

- ValueError – All params are required.
- Exception – Cannot add/update kafka offsets because ERROR-MESSAGE

Returns returns True if offsets are add/updated or throws an exception.

Return type bool

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Notes

This method is used by API-server to store newly created auth-token

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises `ValueError` – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

upload_object(*bucket_name: str*, *object_name: str*, *object_filepath: str*) → bool
Upload an object in a bucket aka folder of object storage system.

Parameters

- **bucket_name** (*str*) – name of the bucket
- **object_name** (*str*) – name of the object to be uploaded
- **object_filepath** (*str*) – it shall contain full path of a file with file name (e.g., /home/nasir/obj.zip)

Returns True if object successfully uploaded. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises `ValueError` – Bucket name cannot be empty/None.

9.1.4 Module contents

class Kernel(*configs_dir_path: str = None*, *auto_offset_reset: str = 'largest'*, *enable_spark: bool = True*, *enable_spark_ui=False*)
Bases: `object`

connect(*username: str*, *password: str*, *encrypted_password: bool = False*) → dict
Authenticate a user based on username and password and return an auth token

Parameters

- **username** (*str*) – username of a user
- **password** (*str*) – password of a user
- **encrypted_password** (*str*) – is password encrypted or not. mCerebrum sends encrypted passwords

Raises `ValueError` – User name and password cannot be empty/None.

Returns return eturn {"status":bool, "auth_token": str, "msg": str}

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.connect("nasir_ali",
↪"2ksdfhoi2r2ljndf823hlkf8234hohwef0234hlkjwer98u234", True)
>>> True
```

create_bucket(*bucket_name: str*) → bool
creates a bucket aka folder in object storage system.

Parameters **bucket_name** (*str*) – name of the bucket

Returns True if bucket was successfully created. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises `ValueError` – Bucket name cannot be empty/None.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.create_bucket("live_data_folder")
>>> True
```

create_user(*username: str*, *user_password: str*, *user_role: str*, *user_metadata: dict*, *user_settings: dict*) → bool

Create a user in SQL storage if it doesn't exist

Parameters

- **username** (*str*) – Only alphanumeric usernames are allowed with the max length of 25 chars.
- **user_password** (*str*) – no size limit on password
- **user_role** (*str*) – role of a user
- **user_metadata** (*dict*) – metadata of a user
- **user_settings** (*dict*) – user settings, mCerebrum configurations of a user

Returns True if user is successfully registered or throws any error in case of failure

Return type bool

Raises

- `ValueError` – if selected username is not available
- `Exception` – if sql query fails

delete_user(*username: str*) → bool

Delete a user record in SQL table

Parameters **username** – username of a user that needs to be deleted

Returns if user is successfully removed

Return type bool

Raises

- `ValueError` – if username param is empty or None
- `Exception` – if sql query fails

encrypt_user_password(*user_password: str*) → str

Encrypt password

Parameters **user_password** (*str*) – unencrypted password

Raises `ValueError` – password cannot be None or empty.

Returns encrypted password

Return type str

gen_random_pass(*string_type: str = 'varchar'*, *size: int = 8*) → str

Generate a random password

Parameters

- **string_type** – Accepted parameters are "varchar" and "char". (Default="varchar")
- **size** – password length (default=8)

Returns random password

Return type str

get_all_users(*study_name: str*) → List[dict]
Get a list of all users part of a study.

Parameters **study_name** (*str*) – name of a study

Raises ValueError – Study name is a requied field.

Returns Returns empty list if there is no user associated to the study_name and/or study_name does not exist.

Return type list[dict]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [{"76cc444c-4fb8-776e-2872-9472b4e66b16": "nasir_ali"}] # [{user_id, user_
↪name}]
```

get_bucket_objects(*bucket_name: str*) → dict
returns a list of all objects stored in the specified Minio bucket

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns {bucket-objects: [{"object_name":"", "metadata": {}}...], in case of an error {"error": str}

Return type dict

get_buckets() → dict
returns all available buckets in an object storage

Returns {bucket-name: str, [{"key":"value"}]}, in case of an error {"error": str}

Return type dict

get_cache_value(*key: str*) → str
Retrieves value from the cache for the given key.

Parameters **key** – key in the cache

Returns The value in the cache

Return type str

Raises ValueError – if key is None or empty

get_kafka_offsets(*topic: str*) → dict
Get last stored kafka offsets

Parameters **topic** (*str*) – kafka topic name

Returns list of kafka offsets. This method will return empty list if topic does not exist and/or no offset is stored for the topic.

Return type list[dict]

Raises `ValueError` – Topic name cannot be empty/None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_kafka_offsets("live-data")
>>> [{"id","topic", "topic_partition", "offset_start", "offset_until",
↪"offset_update_time"}]
```

get_object(*bucket_name: str*, *object_name: str*) → dict
Returns stored object (HttpResponse)

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object that needs to be downloaded

Returns object that needs to be downloaded. If file does not exists then it returns an error {"error": "File does not exist."}

Return type file-object

Raises

- `ValueError` – Missing bucket_name and object_name params.
- `Exception` – {"error": "error-message"}

get_object_stats(*bucket_name: str*, *object_name: str*) → dict
Returns properties (e.g., object type, last modified etc.) of an object stored in a specified bucket

Parameters

- **bucket_name** (*str*) – name of a bucket aka folder
- **object_name** (*str*) – name of an object

Returns information of an object (e.g., creation_date, object_size etc.). In case of an error {"error": str}

Return type dict

Raises

- `ValueError` – Missing bucket_name and object_name params.
- `Exception` – {"error": "error-message"}

get_stream(*stream_name: str*, *version: str = 'all'*, *data_type=<DataSet.COMPLETE: (1,)>*) → cerebralcortex.core.datatypes.datastream.DataStream
Retrieve a data-stream with it's metadata.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")
- **data_type** (`DataSet`) – DataSet.COMPLETE returns both Data and Metadata. DataSet.ONLY_DATA returns only Data. DataSet.ONLY_METADATA returns only metadata of a stream. (Default=DataSet.COMPLETE)

Returns contains Data and/or metadata

Return type *DataStream*

Raises `ValueError` – if stream name is empty or None

Note: Please specify a version if you know the exact version of a stream. Getting all the stream data and then filtering versions won't be efficient.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = CC.get_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV-
↪-RIGHT_WRIST")
>>> ds.data # an object of a dataframe
>>> ds.metadata # an object of MetaData class
>>> ds.get_metadata(version=1) # get the specific version metadata of a stream
```

get_stream_info_by_hash(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str

metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns stream metadata and other info related to a stream

Return type dict

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> {"name": .....} # stream metadata and other information
```

get_stream_metadata(*stream_name: str*, *version: str = 'all'*) → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]

Get a list of metadata for all versions available for a stream.

Parameters

- **stream_name** (*str*) – name of a stream
- **version** (*str*) – version of a stream. Acceptable parameters are all, latest, or a specific version of a stream (e.g., 2.0) (Default="all")

Returns Returns an empty list if no metadata is available for a stream_name or a list of metadata otherwise.

Return type list[*Metadata*]

Raises `ValueError` – stream_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_all_users("mperf")
>>> [Metadata] # list of MetaData class objects
```

get_stream_metadata_hash(*stream_name: str*) → list
Get all the metadata_hash associated with a stream name.

Parameters **stream_name** (*str*) – name of a stream

Returns list of all the metadata hashes

Return type list[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_metadata_hash("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> ["00ab666c-afb8-476e-9872-6472b4e66b68", "15cc444c-dfb8-676e-3872-
↪8472b4e66b12"]
```

get_stream_name(*metadata_hash: <module 'uuid' from '/home/docs/.pyenv/versions/3.6.8/lib/python3.6/uuid.py'>*) → str
metadata_hash are unique to each stream version. This reverse look can return the stream name of a metadata_hash.

Parameters **metadata_hash** (*uuid*) – This could be an actual uuid object or a string form of uuid.

Returns name of a stream

Return type str

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_name("00ab666c-afb8-476e-9872-6472b4e66b68")
>>> ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--RIGHT_WRIST
```

get_stream_versions(*stream_name: str*) → list
Returns a list of versions available for a stream

Parameters **stream_name** (*str*) – name of a stream

Returns list of int

Return type list

Raises `ValueError` – if stream_name is empty or None

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_stream_versions("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_
↪HRV--RIGHT_WRIST")
>>> [1, 2, 4]
```

get_user_id(*user_name: str*) → str
Get the user id linked to user_name.

Parameters **user_name** (*str*) – username of a user

Returns user id associated to user_name

Return type str

Raises ValueError – User name is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_id("nasir_ali")
>>> '76cc444c-4fb8-776e-2872-9472b4e66b16'
```

get_user_metadata(*user_id: str = None*, *username: str = None*) → dict
Get user metadata by user_id or by username

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns user metadata

Return type dict

Todo: Return list of User class object

Raises ValueError – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_metadata(username="nasir_ali")
>>> {"study_name":"mperf".....}
```

get_user_name(*user_id: str*) → str
Get the user name linked to a user id.

Parameters **user_name** (*str*) – username of a user

Returns user_id associated to username

Return type bool

Raises ValueError – User ID is a required field.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_name("76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> 'nasir_ali'
```

get_user_settings(*username: str = None*, *auth_token: str = None*) → dict
Get user settings by auth-token or by username. These are user's mCerebrum settings

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – auth-token

Returns List of dictionaries of user metadata

Return type list[dict]

Todo: Return list of User class object

Raises `ValueError` – User ID/name cannot be empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.get_user_settings(username="nasir_ali")
>>> [{"mcerebrum":"some-conf".....}]
```

is_auth_token_valid(*username: str*, *auth_token: str*, *checktime: bool = False*) → bool
Validate whether a token is valid or expired based on the token expiry datetime stored in SQL

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – token generated by API-Server
- **checktime** (*bool*) – setting this to False will only check if the token is available in system. Setting this to true will check if the token is expired based on the token expiry date.

Raises `ValueError` – Auth token and auth-token expiry time cannot be null/empty.

Returns returns True if token is valid or False otherwise.

Return type bool

is_bucket(*bucket_name: str*) → bool
checks whether a bucket exist

Parameters **bucket_name** (*str*) – name of the bucket aka folder

Returns True if bucket exist or False otherwise. In case an error {"error": str}

Return type bool

Raises `ValueError` – bucket_name cannot be None or empty.

is_object(*bucket_name: str*, *object_name: str*) → bool
checks whether an object exist in a bucket

Parameters

- **bucket_name** (*str*) – name of the bucket aka folder
- **object_name** (*str*) – name of the object

Returns True if object exist or False otherwise. In case an error {"error": str}

Return type bool

Raises Exeception – if bucket_name and object_name are empty or None

is_stream(*stream_name: str*) → bool

Returns true if provided stream exists.

Parameters **stream_name** (*str*) – name of a stream

Returns True if stream_name exist False otherwise

Return type bool

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.is_stream("ACCELEROMETER--org.md2k.motionsense--MOTION_SENSE_HRV--
↪RIGHT_WRIST")
>>> True
```

is_user(*user_id: str = None*, *user_name: str = None*) → bool

Checks whether a user exists in the system. One of both parameters could be set to verify whether user exist.

Parameters

- **user_id** (*str*) – id (uuid) of a user
- **user_name** (*str*) – username of a user

Returns True if a user exists in the system or False otherwise.

Return type bool

Raises ValueError – Both user_id and user_name cannot be None or empty.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.is_user(user_id="76cc444c-4fb8-776e-2872-9472b4e66b16")
>>> True
```

kafka_produce_message(*topic: str*, *msg: dict*)

Publish a message on kafka message queue

Parameters

- **topic** (*str*) – name of the kafka topic
- **msg** (*dict*) – message that needs to published on kafka

Returns True if successful. In case of failure, it returns an Exception message.

Return type bool

Raises

- `ValueError` – topic and message parameters cannot be empty or None.
- `Exception` – Error publishing message. Topic: topic_name - error-message

kafka_subscribe_to_topic(*topic: str*)
Subscribe to kafka topic as a consumer

Parameters **topic** (*str*) – name of the kafka topic

Yields *dict* – kafka message

Raises `ValueError` – Topic parameter is missing.

list_streams() → List[cerebralcortex.core.metadata_manager.stream.metadata.Metadata]
Get all the available stream names with metadata

Returns list of available streams metadata

Return type List[*Metadata*]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.list_streams()
```

save_data_to_influxdb(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*)
Save data stream to influxdb only for visualization purposes.

Parameters **datastream** (DataStream) – a DataStream object

Returns True if data is ingested successfully or False otherwise

Return type bool

Todo: This needs to be updated with the new structure. Should metadata be stored or not?

Example

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_data_to_influxdb(ds)
```

save_stream(*datastream: cerebralcortex.core.datatypes.datastream.DataStream*, *ingestInfluxDB: bool = False*) → bool
Saves datastream raw data in selected NoSQL storage and metadata in MySQL.

Parameters

- **datastream** (DataStream) – a DataStream object
- **ingestInfluxDB** (*bool*) – Setting this to True will ingest the raw data in InfluxDB as well that could be used to visualize data in Grafana

Returns True if stream is successfully stored or throws an exception

Return type bool

Raises `Exception` – log or throws exception if stream is not stored

Todo: Add functionality to store data in influxdb.

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> ds = DataStream(dataframe, MetaData)
>>> CC.save_stream(ds)
```

search_stream(*stream_name*)

Find all the stream names similar to stream_name arg. For example, passing "location" argument will return all stream names that contain the word location

Returns list of stream names similar to stream_name arg

Return type List[str]

Examples

```
>>> CC = Kernel("/directory/path/of/configs/")
>>> CC.search_stream("battery")
>>> ["BATTERY--org.md2k.motionsense--MOTION_SENSE_HRV--LEFT_WRIST", "BATTERY--
↪org.md2k.phonesensor--PHONE".....]
```

set_cache_value(*key: str*, *value: str*) → bool

Creates a new cache entry in the cache. Values are overwritten for existing keys.

Parameters

- **key** – key in the cache
- **value** – value associated with the key

Returns True on successful insert or False otherwise.

Return type bool

Raises `ValueError` – if key is None or empty

store_or_update_Kafka_offset(*topic: str*, *topic_partition: str*, *offset_start: str*, *offset_until: str*) → bool

Store or Update kafka topic offsets. Offsets are used to track what messages have been processed.

Parameters

- **topic** (*str*) – name of the kafka topic
- **topic_partition** (*str*) – partition number
- **offset_start** (*str*) – starting of offset
- **offset_until** (*str*) – last processed offset

Raises

- `ValueError` – All params are required.
- `Exception` – Cannot add/update kafka offsets because ERROR-MESSAGE

Returns returns True if offsets are add/updated or throws an exception.

Return type bool

update_auth_token(*username: str*, *auth_token: str*, *auth_token_issued_time: datetime.datetime*, *auth_token_expiry_time: datetime.datetime*) → bool

Update an auth token in SQL database to keep user stay logged in. Auth token valid duration can be changed in configuration files.

Notes

This method is used by API-server to store newly created auth-token

Parameters

- **username** (*str*) – username of a user
- **auth_token** (*str*) – issued new auth token
- **auth_token_issued_time** (*datetime*) – datetime when the old auth token was issue
- **auth_token_expiry_time** (*datetime*) – datetime when the token will get expired

Raises ValueError – Auth token and auth-token issue/expiry time cannot be None/empty.

Returns Returns True if the new auth token is set or False otherwise.

Return type bool

upload_object(*bucket_name: str*, *object_name: str*, *object_filepath: str*) → bool

Upload an object in a bucket aka folder of object storage system.

Parameters

- **bucket_name** (*str*) – name of the bucket
- **object_name** (*str*) – name of the object to be uploaded
- **object_filepath** (*str*) – it shall contain full path of a file with file name (e.g., /home/nasir/obj.zip)

Returns True if object successfully uploaded. On failure, returns an error with dict {"error":"error-message"}

Return type bool

Raises ValueError – Bucket name cannot be empty/None.

9.2 CerebralCortex Data importer

9.2.1 Subpackages

cerebralcortex.data_importer.data_parsers package

Submodules

cerebralcortex.data_importer.data_parsers.csv_parser module

csv_data_parser(*line: str*) → list

parse each row of data file into list of values (timestamp, localtime, val1, val2....)

Parameters **line** (*str*) –

Returns (timestamp, localtime, val1, val2....)

Return type list

cerebralcortex.data_importer.data_parsers.mcerebrum module

mcerebrum_data_parser(*line: str*) → list
parse each row of data file into list of values (timestamp, localtime, val1, val2....)

Parameters **line** (*str*) –

Returns (timestamp, localtime, val1, val2....)

Return type list

cerebralcortex.data_importer.data_parsers.util module

assign_column_names_types(*df: <module 'pandas' from '/home/docs/checkouts/readthedocs.org/user_builds/cerebralcortex-kernel/envs/stable/lib/python3.6/site-packages/pandas/__init__.py'>*, *metadata: dict = None*) → <module 'pandas' from '/home/docs/checkouts/readthedocs.org/user_builds/cerebralcortex-kernel/envs/stable/lib/python3.6/site-packages/pandas/__init__.py'>
Change column names to the names defined in metadata->data_descriptor block

Parameters

- **df** (*pandas*) – pandas dataframe
- **metadata** (*dict*) – metadata of the data

Returns pandas dataframe

assign_column_names_types_strict(*df: <module 'pandas' from '/home/docs/checkouts/readthedocs.org/user_builds/cerebralcortex-kernel/envs/stable/lib/python3.6/site-packages/pandas/__init__.py'>*, *metadata: dict = None*) → <module 'pandas' from '/home/docs/checkouts/readthedocs.org/user_builds/cerebralcortex-kernel/envs/stable/lib/python3.6/site-packages/pandas/__init__.py'>
Change column names to the names defined in metadata->data_descriptor block

Parameters

- **df** (*pandas*) – pandas dataframe
- **metadata** (*dict*) – metadata of the data

Returns pandas dataframe

Module contents

mcerebrum_data_parser(*line: str*) → list
parse each row of data file into list of values (timestamp, localtime, val1, val2....)

Parameters **line** (*str*) –

Returns (timestamp, localtime, val1, val2....)

Return type list

csv_data_parser(*line: str*) → list
parse each row of data file into list of values (timestamp, localtime, val1, val2....)

Parameters **line** (*str*) –

Returns (timestamp, localtime, val1, val2....)

Return type list

cerebralcortex.data_importer.metadata_parsers package

Submodules

cerebralcortex.data_importer.metadata_parsers.mcerebrum module

convert_json_to_metadata_obj(*metadata: dict*, *annotation_name: str*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
Convert old mcerebrum metadata json files in to new CC-kernel 3.x compatible format

Parameters

- **metadata** (*dict*) – mcerebrum old metadata format
- **annotation_name** (*str*) – name of annotation stream

Returns Metadata object

get_platform_metadata(*metadata: dict*) → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
Build platform metadata out of old mcerebrum metadata format.

Parameters **metadata** (*dict*) – old mecerebrum metadata

Returns Metadata class object

Return type *Metadata*

mcerebrum_metadata_parser(*metadata: dict*) → dict
Convert mcerebrum old metadata format to CC-kernel version 3.x metadata format

Parameters **metadata** (*dict*) – mcerebrum old metadata format

Returns {"platform_metadata":platform_metadata, "stream_metadata":metadata}

Return type dict

new_data_descript_frmt(*data_descriptor: dict*) → dict
convert old mcerebrum data descriptor format to CC-kernel 3.x format

Parameters **data_descriptor** (*dict*) – old mcerebrum data descriptor format

Returns {"name":"..", "type:"..", "attributes":{...}....}

Return type dict

new_module_metadata(*ec: dict*) → dict
convert old mcerebrum data execution_context format to CC-kernel 3.x format

Parameters **ec** (*dict*) – old mcerebrum execution_context block

Returns {"name":".....}

Return type dict

Module contents

mcerebrum_metadata_parser(*metadata: dict*) → dict
Convert mcerebrum old metadata format to CC-kernel version 3.x metadata format

Parameters **metadata** (*dict*) – mcerebrum old metadata format

Returns {"platform_metadata":platform_metadata, "stream_metadata":metadata}

Return type dict

cerebralcortex.data_importer.util package

Submodules

cerebralcortex.data_importer.util.directory_scanners module

dir_scanner(*dir_path: str*, *data_file_extension: list = []*, *allowed_filename_pattern: str = None*, *get_dirs: bool = False*)
Generator method to iterate over directories and return file/dir

Parameters

- **dir_path** (*str*) – path of main directory that needs to be iterated over
- **data_file_extension** (*list*) – file extensions that must be excluded during directory scanning
- **allowed_filename_pattern** (*str*) – regex expression to get file names matched to the regex
- **get_dirs** (*bool*) – set it true to get directory name as well

Yields filename with its full path

cerebralcortex.data_importer.util.helper_methods module

rename_column_name(*column_name*)

Module contents

9.2.2 Submodules

9.2.3 cerebralcortex.data_importer.ingest module

import_dir(*cc_config: dict*, *input_data_dir: str*, *user_id: str = None*, *data_file_extension: list = []*, *allowed_filename_pattern: str = None*, *allowed_streamname_pattern: str = None*, *ignore_streamname_pattern: str = None*, *batch_size: int = None*, *compression: str = None*, *header: int = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*, *metadata_parser: Callable = None*, *data_parser: Callable = None*, *gen_report: bool = False*)
Scan data directory, parse files and ingest data in cerebralcortex backend.

Parameters

- **cc_config** (*str*) – cerebralcortex config directory
- **input_data_dir** (*str*) – data directory path
- **user_id** (*str*) – user id. Currently import_dir only supports parsing directory associated with a user
- **data_file_extension** (*list[str]*) – (optional) provide file extensions (e.g., .doc) that must be ignored
- **allowed_filename_pattern** (*str*) – (optional) regex of files that must be processed.
- **allowed_streamname_pattern** (*str*) – (optional) regex of stream-names to be processed only
- **ignore_streamname_pattern** (*str*) – (optional) regex of stream-names to be ignored during ingestion process
- **batch_size** (*int*) – (optional) using this parameter will turn on spark parallelism. batch size is number of files each worker will process
- **compression** (*str*) – pass compression name if csv files are compressed
- **header** (*str*) – (optional) row number that must be used to name columns. None means file does not contain any header
- **metadata** (Metadata) – (optional) Same metadata will be used for all the data files if this parameter is passed. If metadata is passed then metadata_parser cannot be passed.
- **metadata_parser** (*python function*) – a parser that can parse json files and return a valid MetaData object. If metadata_parser is passed then metadata parameter cannot be passed.
- **data_parser** (*python function*) – a parser than can parse each line of data file. import_dir read data files as a list of lines of a file. data_parser will be applied on all the rows.
- **gen_report** (*bool*) – setting this to True will produce a console output with total failures occurred during ingestion process.

Notes

Each csv file should contain a metadata file. Data file and metadata file should have same name. For example, data.csv and data.json. Metadata files should be json files.

Todo: Provide sample metadata file URL

---

import_file(*cc_config: dict*, *user_id: str*, *file_path: str*, *allowed_streamname_pattern: str = None*, *ignore_streamname_pattern: str = None*, *compression: str = None*, *header: int = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*, *metadata_parser: Callable = None*, *data_parser: Callable = None*)

Import a single file and its metadata into cc-storage.

Parameters

- **cc_config** (*str*) – cerebralcortex config directory

- **user_id** (*str*) – user id. Currently import_dir only supports parsing directory associated with a user
- **file_path** (*str*) – file path
- **allowed_streamname_pattern** (*str*) – (optional) regex of stream-names to be processed only
- **ignore_streamname_pattern** (*str*) – (optional) regex of stream-names to be ignored during ingestion process
- **compression** (*str*) – pass compression name if csv files are compressed
- **header** (*str*) – (optional) row number that must be used to name columns. None means file does not contain any header
- **metadata** (Metadata) – (optional) Same metadata will be used for all the data files if this parameter is passed. If metadata is passed then metadata_parser cannot be passed.
- **metadata_parser** (*python function*) – a parser that can parse json files and return a valid MetaData object. If metadata_parser is passed then metadata parameter cannot be passed.
- **data_parser** (*python function*) – a parser than can parse each line of data file. import_dir read data files as a list of lines of a file. data_parser will be applied on all the rows.
- **Notes** –
- **csv file should contain a metadata file. Data file and metadata file should have same name. For example, data.csv and data.json.** (*Each*) –
- **files should be json files.** (Metadata) –

Returns False in case of an error

Return type bool

print_stats_table(*ingestion_stats: dict*)

Print import data stats in table.

Parameters **ingestion_stats** (*dict*) – basic import statistics. {"fault_type": [], "total_faults": []}

save_data(*df: object*, *cc_config: dict*, *user_id: str*, *stream_name: str*)

save dataframe to cc storage system

Parameters

- **df** (*pandas*) – dataframe
- **cc_config** (*str*) – cerebralcortex config directory
- **user_id** (*str*) – user id
- **stream_name** (*str*) – name of the stream

9.2.4 cerebralcortex.data_importer.main module

9.2.5 Module contents

import_file(*cc_config: dict*, *user_id: str*, *file_path: str*, *allowed_streamname_pattern: str = None*, *ignore_streamname_pattern: str = None*, *compression: str = None*, *header: int = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*, *metadata_parser: Callable = None*, *data_parser: Callable = None*)

Import a single file and its metadata into cc-storage.

Parameters

- **cc_config** (*str*) – cerebralcortex config directory
- **user_id** (*str*) – user id. Currently import_dir only supports parsing directory associated with a user
- **file_path** (*str*) – file path
- **allowed_streamname_pattern** (*str*) – (optional) regex of stream-names to be processed only
- **ignore_streamname_pattern** (*str*) – (optional) regex of stream-names to be ignored during ingestion process
- **compression** (*str*) – pass compression name if csv files are compressed
- **header** (*str*) – (optional) row number that must be used to name columns. None means file does not contain any header
- **metadata** (Metadata) – (optional) Same metadata will be used for all the data files if this parameter is passed. If metadata is passed then metadata_parser cannot be passed.
- **metadata_parser** (*python function*) – a parser that can parse json files and return a valid MetaData object. If metadata_parser is passed then metadata parameter cannot be passed.
- **data_parser** (*python function*) – a parser than can parse each line of data file. import_dir read data files as a list of lines of a file. data_parser will be applied on all the rows.
- **Notes** –
- **csv file should contain a metadata file. Data file and metadata file should have same name. For example, data.csv and data.json.** (*Each*) –
- **files should be json files.** (Metadata) –

Returns False in case of an error

Return type bool

import_dir(*cc_config: dict*, *input_data_dir: str*, *user_id: str = None*, *data_file_extension: list = []*, *allowed_filename_pattern: str = None*, *allowed_streamname_pattern: str = None*, *ignore_streamname_pattern: str = None*, *batch_size: int = None*, *compression: str = None*, *header: int = None*, *metadata: cerebralcortex.core.metadata_manager.stream.metadata.Metadata = None*, *metadata_parser: Callable = None*, *data_parser: Callable = None*, *gen_report: bool = False*)

Scan data directory, parse files and ingest data in cerebralcortex backend.

Parameters

- **cc_config** (*str*) – cerebralcortex config directory
- **input_data_dir** (*str*) – data directory path
- **user_id** (*str*) – user id. Currently import_dir only supports parsing directory associated with a user
- **data_file_extension** (*list[str]*) – (optional) provide file extensions (e.g., .doc) that must be ignored
- **allowed_filename_pattern** (*str*) – (optional) regex of files that must be processed.
- **allowed_streamname_pattern** (*str*) – (optional) regex of stream-names to be processed only
- **ignore_streamname_pattern** (*str*) – (optional) regex of stream-names to be ignored during ingestion process
- **batch_size** (*int*) – (optional) using this parameter will turn on spark parallelism. batch size is number of files each worker will process
- **compression** (*str*) – pass compression name if csv files are compressed
- **header** (*str*) – (optional) row number that must be used to name columns. None means file does not contain any header
- **metadata** (Metadata) – (optional) Same metadata will be used for all the data files if this parameter is passed. If metadata is passed then metadata_parser cannot be passed.
- **metadata_parser** (*python function*) – a parser that can parse json files and return a valid MetaData object. If metadata_parser is passed then metadata parameter cannot be passed.
- **data_parser** (*python function*) – a parser than can parse each line of data file. import_dir read data files as a list of lines of a file. data_parser will be applied on all the rows.
- **gen_report** (*bool*) – setting this to True will produce a console output with total failures occurred during ingestion process.

Notes

Each csv file should contain a metadata file. Data file and metadata file should have same name. For example, data.csv and data.json. Metadata files should be json files.

Todo: Provide sample metadata file URL

9.3 CerebralCortex Algorithms

9.3.1 Subpackages

cerebralcortex.algorithms.gps package

Submodules

cerebralcortex.algorithms.gps.gps_clustering module

get_centermost_point (*cluster: object*) → object

Parameters **cluster** –

Returns

Return type object

gps_clusters (*data: object*) → object
Computes the clusters

Return type object

Parameters

- **data** (*list*) – list of interpolated gps data
- **geo_fence_distance** (*float*) – Maximum distance between points in a

cluster :param int min_points_in_cluster: Minimum number of points in a cluster :return: list of cluster-centroids coordinates

Module contents

gps_clusters (*data: object*) → object
Computes the clusters

Return type object

Parameters

- **data** (*list*) – list of interpolated gps data
- **geo_fence_distance** (*float*) – Maximum distance between points in a

cluster :param int min_points_in_cluster: Minimum number of points in a cluster :return: list of cluster-centroids coordinates

9.3.2 Module contents

gps_clusters (*data: object*) → object
Computes the clusters

Return type object

Parameters

- **data** (*list*) – list of interpolated gps data
- **geo_fence_distance** (*float*) – Maximum distance between points in a

cluster :param int min_points_in_cluster: Minimum number of points in a cluster :return: list of cluster-centroids coordinates

process_ecg (*data: object*) → object

rr_interval_feature_extraction (*data: object*) → object

stress_prediction (*data: object*) → object

stress_episodes_estimation (*stress_data: object*) → object

9.4 CerebralCortex Test Suite

9.4.1 Subpackages

cerebralcortex.test_suite.util package

Submodules

cerebralcortex.test_suite.util.data_helper module

gen_phone_battery_data() → object
Create pyspark dataframe with some sample phone battery data

Returns pyspark dataframe object with columns: ["timestamp", "offset", "battery_level", "ver", "user"]

Return type DataFrame

gen_phone_battery_data2() → object
Create pyspark dataframe with some sample phone battery data

Returns pyspark dataframe object with columns: ["timestamp", "offset", "battery_level", "ver", "user"]

Return type DataFrame

gen_phone_battery_metadata() → cerebralcortex.core.metadata_manager.stream.metadata.Metadata
Create Metadata object with some sample metadata of phone battery data

Returns metadata of phone battery stream

Return type *Metadata*

Module contents

9.4.2 Submodules

9.4.3 cerebralcortex.test_suite.test_kafka module

class TestKafkaMessaging
Bases: object

test_01_produce_message()
Produce a message on kafka topic

test_02_consume_message()
Consume kafka messages in a topic

9.4.4 cerebralcortex.test_suite.test_main module

class TestCerebralCortex(*methodName='runTest'*)
Bases: unittest.case.TestCase, *cerebralcortex.test_suite.test_stream.DataStreamTest*

setUp()
Setup test params to being testing with.

Notes

DO NOT CHANGE PARAMS DEFINED UNDER TEST-PARAMS! OTHERWISE TESTS WILL FAIL.
These values are hardcoded in util/data_helper file as well.

test_00()
This test will create required entries in sql database.

test_9999_last()
Delete all the sample test data folder/files and sql entries

9.4.5 cerebralcortex.test_suite.test_object_storage module

class TestObjectStorage
Bases: object

test_01_bucket()
Perform all bucket related tests

test_03_bucket_objects()
Perform all object related tests

9.4.6 cerebralcortex.test_suite.test_sql_storage module

class SqlStorageTest
Bases: object

test_01_is_stream()

test_02_get_stream_versions()

test_03_get_stream_name()

test_04_get_stream_metadata_hash()

test_05_get_user_id()

test_06_get_user_name()

test_07_get_all_users()

test_08_get_user_metadata()

test_09_encrypt_user_password()

test_10_connect()

9.4.7 cerebralcortex.test_suite.test_stream module

class DataStreamTest
Bases: object

test_01_save_stream()
Test functionality related to save a stream

test_05_map_window_to_stream()

9.4.8 Module contents

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

D

E

F

G

H

I

J

K

L

M

T

U

W