Factiva News

Release 0.2.4

Dow Jones

Dec 02, 2022

OVERVIEW

This package simplifies the integration process to the Dow Jones Snapshots and Streams services.

Check out the Dow Jones Developer Platofrm site for more information about the available services.

CHAPTER

ONE

OVERVIEW

Factiva News is a package that provides utilities to ease the integration to Factiva Analytics APIs which is part of the Dow Jones Developer Platform. This packages aims to ease news data operations like estimations, extractions and real-time consumption.

CHAPTER

TWO

INSTALLATION

This package can be installed using PIP, the recommended pocedure is running:

```
pip install -u factiva-news
```

This will install and update the package to the latest version.

CHAPTER

THREE

BUILDING QUERIES

3.1 Where Attribute

The where attribute is the most used attribute to select content by applying different conditions to the available fields. Below there are some code snippets that help building a where statement, and notes that help modify the syntax according to the need.

3.1.1 Range of dates

It works with the standard SQL syntax. If updates will be collected in the future, avoid using an end-date. Alternative fields are: *modification_datetime* and *ingestion_datetime*.

```
publication_datetime >= '2010-01-01 00:00:00' AND publication_datetime <= '2020-06-30␣
↪23:59:59'
```

3.1.2 Filter by *source_codes*

The following clause is useful to select sources by their individual code.

```
UPPER(source_code) IN ('AASFNE', 'HTACCF', 'NLADLW', 'ADVTSR', 'AFNROL', 'AGEEOL', 'AGEE
↪', 'HNASNI', 'APRS', 'ASXTEX', 'AUSTOL')
```

In case sources will be selected by their category or source family, a better option is using *restrictor_codes*. This field is not in the documentation, but the CSE or Integration Team can provide more information like source family codes

```
REGEXP_CONTAINS(restrictor_codes, r'(?i)(^|,)(jpost|nytf|wp|latm|j)($|,)')
```

3.1.3 Filter by *subject_codes*

```
REGEXP_CONTAINS(subject_codes, r'(?i)(^|,
↪)(mcat|ccat|ecat|gglobe|ghea|ghnwi|gcns|gpir|gdatap|greest|grisk|gsci|gspace|gtrans)(
↪$|,)')
```

3.1.4 Filtering by the region where the source is headquarted

```
REGEXP_CONTAINS(region_of_origin, r'(?i)(aust|spain|italy|usa|uk)')
```

3.1.5 Filtering by language

```
LOWER(language_code) IN ('en', 'es', 'it')
```

3.1.6 Filtering by company codes

This is applicable to any company-related fields (about, occur or company_codes and other combinations with identifiers - ISIN, CUSIP…).

```
REGEXP_CONTAINS(company_codes, r'(?i)(^|,
↪)(agbpet|agip|agphng|agpnme|agzgi|altgaz|bbor|brnene|distrg|eenivm|egapg|enichm|enie|enimnt)(
↪$|,)')
```

In case the interest is to ensure at least one company is tagged (the field is not empty), the expressions looks like this

```
LENGTH(company_codes) > 2
```

Filtering for content with at least 1 relevant company

```
LENGTH(company_codes_about) > 0
```

3.1.7 Filtering by Industry code

```
REGEXP_CONTAINS(industry_codes, r'(?i)(^|,)(i1|i25121|i2567)($|,)')
```

3.1.8 Filtering by Executive codes

```
REGEXP_CONTAINS(LOWER(person_codes), r'(?i)(^|,)(76064380|2349856)($|,)')
```

3.1.9 Filtering by the region the article is about

```
REGEXP_CONTAINS(region_codes, r'(?i)(^|,)(aust|spain|italy|usa|uk)($|,)')
```

3.1.10 Filtering by terms in full-text (Keyword search)

```
REGEXP_CONTAINS(CONCAT(title, ' ', IFNULL(snippet, ''), ' ', IFNULL(body, '')), r'(?i)(^
↪|\b)(economic|economy|regulation|deficit|budget\W+tax|central\W+bank)($|.|\b)')
```

More examples are available in the Data Selection Samples in the Dow Jones Developer Portal (https://developer.dowjones.com/site/docs/data_selection_samples/index.gsp#)

Building the where statement. Python concatenates the strings when inside the parenthesis. Mind the extra space at the end of each string.

CHAPTER

FOUR

QUICKSTART

The easiest way to start using *factiva-news* is by running a news extraction volume estimation using the Explain operation.

```
from factiva.news.snapshot import Snapshot
my_query = "publication_datetime >= '2020-01-01 00:00:00' AND LOWER(language_code) = 'en'
↪"
my_snapshot = Snapshot(user_key='abcd1234abcd1234abcd1234abcd1234', query=my_query)
my_snapshot.process_explain()  # This operation can take several minutes to complete
print(my_snapshot.last_explain_job.document_volume)
```

After its execution, the object *last_explain_job* contains details about the job itself and the estimated volume.

CHAPTER

FIVE

SNAPSHOT PACKAGE

5.1 Snapshot

class factiva.news.snapshot.**Snapshot**(*user_key=None*, *user_stats=False*, *query=None*, *snapshot_id=None*)

Bases: BulkNewsBase

Represent a Factiva Snapshot Class.

Class that represents a Factiva Snapshot.

Parameters

- **user_key** (*str or UserKey*) – String containing the 32-character long APi Key. If not provided, the constructor will try to obtain its value from the FACTIVA_USERKEY environment variable.
- **user_stats** (*boolean, optional (Default: False)*) – Indicates if user data has to be pulled from the server. This operation fills account detail properties along with maximum, used and remaining values. It may take several seconds to complete.
- **query** (*str or SnapshotQuery, optional*) – Query used to run any of the Snapshot-related operations. If a str is provided, a simple query with a *where* clause is created. If other query fields are required, either provide the SnapshotQuery object at creation, or set the appropriate object values after creation. This parameter is not compatible with snapshot_id.
- **snapshot_id** (*str, optional*) – String containing the 10-character long Snapshot ID. This parameter is not compatible with query.

See also:

Stream
Class that represents the continuous Factiva News document stream.

Examples

Creating a new Snapshot with an key string and a Where statement. Then, running a full Explain process.

```
>>> from factiva.news.snapshot import Snapshot
>>> my_key = "abcd1234abcd1234abcd1234abcd1234"
>>> my_query = "publication_datetime >= '2020-01-01 00:00:00' AND␣
↪LOWER(language_code) = 'en'"
>>> my_snapshot = Snapshot(user_key=my_key, query=my_query)
```

(continues on next page)

(continued from previous page)

```
>>> my_snapshot.process_explain()
106535
```

Creating a new Snapshot from an UserKey and a SnapshotQuery instances. Then, running a full Analytics process.

```
>>> my_user = UserKey()
>>> my_query = SnapshotQuery("publication_datetime >= '2020-01-01 00:00:00' AND␣
↪LOWER(language_code) = 'en'")
>>> my_query.frequency = 'YEAR'
>>> my_query.group_by_source_code = True
>>> my_query.top = 20
>>> my_snapshot = Snapshot(user_key=my_user, query=my_query)
>>> analytics_df = my_snapshot.process_analytics()
>>> analytics_df.head()
      count  publication_datetime  source_code
    0   20921   1995    NGCIOS
    1   20371   1995    LATAM
    2   18303   1995    REUTES
    3   10593   1995    EXPNSI
    4   4212    1995    MUNDO
```

download_extraction_files(*download_path=None*)

Download the list of files listed in the Snapshot.last_extraction_job.files.

Downloads the list of files listed in the Snapshot.last_extraction_job.files property, and stores them in a folder indicated by *download_path*. If no *download_path* is provided, then files are stored in a folder with the same name as the snapshot ID.

Parameters

download_path (*str, optional*) – String containing the file path on where to store the files. If not provided, files are stored in a folder with the same name as the update ID.

Returns

Boolean

Return type

True if the files were correctly downloaded, False otherwise.

download_update_files(*download_path=None*)

Download the list of files listed in the Snapshot.last_update_job.files property.

Download the list of files listed in the Snapshot.last_update_job.files property, and stores them in a folder indicated by *download_path*. If no *download_path* is provided, then files are stored in a folder with the same name as the update ID.

Parameters

download_path (*str, optional*) – String containing the file path on where to store the files. If not provided, files are stored in a folder with the same name as the update ID.

Raises

- RuntimeError when an update job has not been submitted. –

Returns

Boolean

Return type
True if the files were correctly downloaded, False otherwise.

`file_format = ''`

`file_list = []`

`folder_path = ''`

`get_analytics_job_results()`

Obtain the Analytics job results from the Factiva Snapshots API.

Obtains the Analytics job results from the Factiva Snapshots API. Results are stored in the *last_analytics_job* class property.

Returns
Boolean – otherwise.

Return type
True if the data was retrieved successfully. An Exception

`get_explain_job_results()`

Obtain the Explain job results from the Factiva Snapshots API.

Obtains the Explain job results from the Factiva Snapshots API. Results are stored in the *last_explain_job* class property.

Returns
Boolean – otherwise.

Return type
True if the data was retrieved successfully. An Exception

`get_explain_job_samples(num_samples=10)`

Obtain the Explain job samples from the Factiva Snapshots API.

Returns a list of up to 100 sample documents (no full-text) which includes title and metadata fields.

Parameters
num_samples (*int, optional (Default: 10)*) – Number of sample documents to get explained by a job

Returns
Boolean – otherwise.

Return type
True if the data was retrieved successfully. An Exception

`get_extraction_job_results()`

Obtain the Extraction job results from the Factiva Snapshots API.

Obtains the Extraction job results from the Factiva Snapshots API. Results are stored in the *last_extraction_job* class property.

Returns
Boolean – otherwise.

Return type
True if the data was retrieved successfully. An Exception

get_update_job_results()

Obtain of the Update Job results from the Factiva Snapshots API.

Obtains the Update Job results from the Factiva Snapshots API. Results are stored in the *last_update_job* class property.

Raises
- RuntimeError when an update job has not beed submitted. –

Returns
Boolean – otherwise.

Return type
True if the data was retrieved successfully. An Exception

last_analytics_job = None

last_explain_job = None

last_extraction_job = None

last_update_job = None

news_data = None

process_analytics()

Submit an Analytics job to the Factiva Snapshots API.

Submits an Analytics job to the Factiva Snapshots API, using the same parameters used by *submit_analytics_job*. Then, monitors the job until its status change to *JOB_STATE_DONE*. Finally, retrieves and stores the results in the property *last_analytics_job*.

Returns
Boolean – otherwise.

Return type
True if the analytics processing was successful. An Exception

Examples

Process analytics job

```
>>> query_clause = "publication_datetime >= '2018-01-01 00:00:00' AND␣
↪publication_datetime <= '2018-01-02 00:00:00' AND LOWER(language_code) =
↪'en'"
>>> my_snapshot = Snapshot(user_key='abcd1234abcd1234abcd1234abcd1234',␣
↪query=query_clause)
>>> my_snapshot.process_analytics()
>>> print(my_snapshot.last_analytics_job.data)
    publication_datetime   count
0               2018-01  950516
1               2018-02  929795
2               2018-03  998663
3               2018-04  935845
4               2018-05  894903
5               2018-06  876938
6               2018-07  867509
```

(continues on next page)

(continued from previous page)

```
7               2018-08  793283
8               2018-09  858963
9               2018-10  957739
10              2018-11  917355
11              2018-12   38401
```

process_explain()

Submit an Explain job to the Factiva Snapshots API.

Submits an Explain job to the Factiva Snapshots API, using the same parameters used by *submit_explain_job*. Then, monitors the job until its status change to *JOB_STATE_DONE*. Finally, retrieves and stores the results in the property *last_explain_job*.

Returns

Boolean – otherwise.

Return type

True if the explain processing was successful. An Exception

Examples

Process explain job from snapshot

```
>>> query_clause = "publication_datetime >= '2018-01-01 00:00:00' AND␣
↪publication_datetime <= '2018-01-02 00:00:00' AND LOWER(language_code) =
↪'en'"
>>> my_snapshot = Snapshot(user_key='abcd1234abcd1234abcd1234abcd1234',␣
↪query=query_clause)
>>> try:
...         my_snapshot.process_explain()
>>> except RuntimeError:
...         print('There was an error with the API call')
>>>
>>> print(my_snapshot.last_explain_job.document_volume)
450483
```

process_extraction(*download_path=None*)

Submit an Extraction job to the Factiva Snapshots API.

Submits an Extraction job to the Factiva Snapshots API, using the same parameters used by *submit_extraction_job*. Then, monitors the job until its status change to *JOB_STATE_DONE*. The final status is retrieved and stored in the property *last_extraction_job*, which among other properties, contains the list of files to download. The process then downloads all files to the specified *download_path*. If no download path is provided, files are stored in a folder named equal to the *snapshot_id* property. The process ends after all files are downloaded.

Because the whole processing takes places in a single call, it's expected that the execution of this operation takes several minutes, or even hours.

Parameters

download_path (*str, optional*) – String containing the file path on where to store the files. If not provided, files are stored in a folder with the same name as the update ID.

Returns
Boolean – otherwise.

Return type
True if the extraction processing was successful. An Exception

Examples

Process extraction job.

```
>>> query_clause = "publication_datetime >= '2018-01-01 00:00:00' AND␣
↪publication_datetime <= '2018-01-02 00:00:00' AND LOWER(language_code) =
↪'en'"
>>> my_snapshot = Snapshot(user_key='abcd1234abcd1234abcd1234abcd1234',␣
↪query=query_clause)
>>> my_snapshot.process_extraction(path='../downloads/data')
```

process_update(*update_type*, *download_path=None*)

Submit an Update job to the Factiva Snapshots API.

Submits an Update job to the Factiva Snapshots API using the same parameters used by *submit_update_job*. Then, monitors the job until its status change to *JOB_STATE_DONE*. The final status is retrieved and stored in the property *last_update_job*, which among other properties, contains the list of files to download. The process then downloads all files to the specified *download_path*. If no download path is provided, files are stored in a folder named equal to the *last_update_job.job_id* property.

Because the whole processing takes places in a single call, it's expected that the execution of this operation takes several minutes, or even hours.

Parameters

- **update_type** (*str*) – String containing the update type to submit a job. Could be 'additions', 'replacements' or 'deletes'.
- **download_path** (*str, optional*) – String containing the file path on where to store the files. If not provided, files are stored in a folder with the same name as the update ID.

Returns
Boolean – otherwise.

Return type
True if the update processing was successful. An Exception

Examples

Process update job with type 'additions'

```
>>> previous_snapshot = Snapshot(user_key=my_user, snapshot_id='sdjjekl93j')
>>> previous_snapshot.process_update('additions', download_path=f'./
↪{previous_snapshot.snapshot_id}/additions/')
```

query = None

`submit_analytics_job()`

Submit an Analytics job to the Factiva Snapshots API.

Submits an Analytics job to the Factiva Snapshots API, using the assigned user in the *user_key*, and SnapshotQuery in the *query* properties.

Returns
Boolean

Return type
True if the submission was successful. An Exception otherwise.

`submit_explain_job()`

Submit an Explain job to the Factiva Snapshots API.

Submits an Explain job to the Factiva Snapshots API, using the assigned user in the *user_key*, and SnapshotQuery in the *query* properties.

Returns
Boolean

Return type
True if the submission was successful. An Exception otherwise.

`submit_extraction_job()`

Submit an Extraction job to the Factiva Snapshots API.

Submits an Extraction job to the Factiva Snapshots API, using the assigned user in the *user_key*, and SnapshotQuery in the *query* properties.

Returns
Boolean

Return type
True if the submission was successful. An Exception otherwise.

`submit_update_job(update_type)`

Submit an Update Job to the Factiva Snapshots API.

Submits an Update Job to the Factiva Snapshots API, using the assigned user in the *user_key* and *snapshot_id* asigned to the instance and the *update_type* passed as parameter. Assigns the submitted job to the *last_update_job* property.

Parameters
update_type (*str*) – String containing the update type to submit a job. Could be 'additions', 'replacements' or 'deletes'.

Returns
Boolean

Return type
True if the submission was successful. An Exception otherwise.

5.2 Snapshot Jobs

class factiva.news.snapshot.**ExplainJob**(*user_key*)

Bases: BulkNewsJob

Represent the operation of creating an explain from Factiva Snapshots API.

document_volume = 0

extraction_type = 'documents'

get_endpoint_url()

Get endpoint URL.

get_job_id(*source*)

Get job ID.

set_job_data(*source*)

Set job data.

class factiva.news.snapshot.**AnalyticsJob**(*user_key*)

Bases: BulkNewsJob

Represent the operation of creating Analtyics from Factiva Snapshots API.

data = []

get_endpoint_url()

Get endpoint URL.

get_job_id(*source*)

Get job ID.

set_job_data(*source*)

Sets job data.

class factiva.news.snapshot.**ExtractionJob**(*snapshot_id=None*, *user_key=None*)

Bases: BulkNewsJob

Class that represents the operation of creating a Snapshot from Factiva Snapshots API.

file_format = ''

files = []

get_endpoint_url()

Obtain endpoint URL.

get_job_id(*source*)

Obtain Job ID.

process_job(*payload=None*, *path=None*)

Override method from parent class to call the method for downloading the files once the snapshot has been completed.

Overrides method from parent class to call the method for downloading the files once the snapshot has been completed.

Parameters

- **payload** (*str, Optional*) – String containg the snapshot instance.
- **path** (*str, Optional*) – String containg the path where to store the snapshots files that are downloaded from the snapshot. If no path is given, the files will be stored in a folder named after the snapshot_id in the current working directory.

set_job_data(*source*)

Set job data.

class factiva.news.snapshot.**UpdateJob**(*update_type=None*, *snapshot_id=None*, *update_id=None*, *user_key=None*)

Bases: *ExtractionJob*

Represent the Snapshot Updates.

Class that represents the Snapshot Updates. There can be three types of updates: additions, replacements and deletes.

Parameters

- **update_type** (*str, Optional*) – String describing the type of update that this job represents. Requires snapshot_id to be provided as well. Not compatible with update_id
- **snapshot_id** (*str, Optional*) – String containing the id of the snapshot that is being updated. Requires update_type to be provided as well. Not compatible with update_id
- **update_id** (*str, Optional*) – String containing the id of an update job that has been created previously. Both update_type and snapshot_id can be obtained from this value. Not compatible with update_type nor snapshot_id

Raises

- Exception when fields that are not compatible are provided or when not enough parameters are provided to create the job. –

get_endpoint_url()

Get endpoint URL.

get_job_id(*source*)

Get job ID from source.

snapshot_id = None

update_type = None

5.3 Snapshot Query

class factiva.news.snapshot.**SnapshotQuery**(*where*, *includes=None*, *excludes=None*, *select_fields=None*, *limit=0*, *file_format='avro'*, *frequency='MONTH'*, *date_field='publication_datetime'*, *group_by_source_code=None*, *group_dimensions=None*, *top=10*)

Bases: BulkNewsQuery

Implement Snapshot Query class definition.

date_field = ''

file_format = ''

frequency = ''

get_analytics_query()

Obtain analytics Query.

get_explain_query()

Obtain Base Query.

get_extraction_query()

Obtain the string querying Factiva.

group_by_source_code = None

group_dimensions = None

limit = 0

top = 0

CHAPTER

SIX

NEWS.STREAM

6.1 Stream

class factiva.news.stream.**Stream**(*stream_id=None*, *snapshot_id=None*, *query=''*, *user_key=None*, *user_stats=False*)

Bases: object

Represent a Stream workflow for Factiva API.

Parameters

- **stream_id** (*str*) – represents a given stream by its id if exists there is no need to have a query or a given snapshot id
- **snapshot_id** (*str*) – represents a snapshot by its id if exists it will be used to create a new stream
- **query** (*str*) – represents a query if exists it will be used to create a new stream
- **user_key** (*str*) – constructor will asign a stream user which has the access to Pubsub client, authentication headers and urls
- **required)** (*Auth method 2 (All)*) –
- **user_key** – if the user_key is not passed it can be created based on the api key param
- **user_stats** (*bool*) – if the user_key is not passed it can be created based on the request info param
- **required)** –
- **user_id** (*str*) – if the user_key is not passed it can be created based on the user id param
- **client_id** (*str*) – if the user_key is not passed it can be created based on the client id param
- **password** (*str*) – if the user_key is not passed it can be created based on the password param

Examples

Creating a new Stream directly:

```
>>> stream_query_test = Stream(
    user_key='abcd1234abcd1234abcd1234abcd1234',
    user_stats=True,
    snapshot_id='<snapshot-id>',
    )
>>> print(stream_query_test.create())
>>>                       attributes                          id     ␣
↪relationships                                      type
job_status     JOB_STATE_PENDING  dj-synhub-extraction-...               NaN  ␣
↪                                        stream
subscriptions                NaN  dj-synhub-extraction-...          {'data': [{
↪'id': 'dj-synhub-extraction-...         stream
```

property all_subscriptions: List[str]

List all subscriptions to a stream.

consume_async_messages(*callback=None*, *subscription_id=None*, *ack_enabled=False*)

Consume messages (News) from a pubsub subscription in async.

Parameters

- **callback** (*function*) – is used for processing a message
- **subscription_id** (*str*) – is used for connecting to pubsub
- **ack_enabled** (*boolean*) – is used for acknowledging a message

Raises

ValueError – when subscription id is invalid:

consume_messages(*callback=None*, *subscription_id=None*, *maximum_messages=None*, *batch_size=None*, *ack_enabled=False*)

Consume messages (News) from a pubsub subscription in sync.

Parameters

- **callback** (*function*) – is used for processing a message
- **subscription_id** (*str*) – is used for connecting to pubsub
- **maximum_messages** (*int*) – is used for consuming a specific number of messages
- **batch_size** (*int*) – the limit of the batch expected
- **ack_enabled** (*boolean*) – is used for acknowledging a message

Raises

ValueError – when subscription id is invalid:

create() → StreamResponse

Create a stream instance.

There are two available options: Create a stream using a query Create a stream using a snapshot id

Returns

of the current stream

Return type
StreamResponse which contains all information

Raises
ValueError – snapshot_id and query are undefined:

create_default_subscription(*response*)

Create the default subscriptions at initialization.

Adds the subscriptions to subscriptions dict

Parameters
response (*dict*) – is used for setting every subscription which exists inside the stream

create_subscription() → str

Create another subscription for an existing stream.

Returns
the new subscription id

Return type
String which represents

Raises
RuntimeError – when unable to create a subscription:

delete() → StreamResponse

Delete a stream.

Returns
of the current which is expected to be CANCELLED

Return type
StreamResponse which contains all information

Raises

- **ValueError** – when stream id is undefined:
- **RuntimeError** – when the stream does not exists:
- **RuntimeError** – when exists an unexpected HTTP error:

delete_subscription(*sus_id*) → bool

Delete subscription for an existing stream.

Parameters
sus_id (*str*) – is the representation of a given subscription planned to be deleted

Returns
was successfully done

Return type
boolean which represents if the delete

Raises

- **ValueError** – when there is invalid subscription id:
- **RuntimeError** – when unable to delete a subscription:

get_all_streams() → dict

Obtain streams from a given user.

Return type
Json object -> list of objects containing information about every stream (id, link, state, etc)

Raises
RuntimeError – when exists an unexpected HTTP error:

get_info() → StreamResponse

Query a stream by its id.

Returns
of the current sream

Return type
StreamResponse which contains all information

Raises

- **ValueError** – when stream id is undefined:
- **RuntimeError** – when the stream does not exists:
- **RuntimeError** – when exists an unexpected HTTP error:

get_suscription_by_id(*susbcription_id*) → *Subscription*

get_suscription_by_index(*index*) → *Subscription*

get_suscription_id_by_index(*index*) → str

listener = None

set_all_subscriptions()

Allow a user to set all subscriptions from a stream to local storage.

Return type
Dataframe which contains the state about the current stream

Raises
ValueError – when stream id is undefined:

snapshot_id = None

stream_id = None

property stream_url: str

List Stream's URL address.

stream_user = None

subscriptions = {}

6.2 Subscription

class factiva.news.stream.**Subscription**(*stream_id=None*, *id=None*, *subscription_type=None*)

Bases: object

Represent a Subscription inside a stream.

Class that represents a Subscription inside a stream. There a two possible operations for a Subscription: - Create new one based on an existing Stream - Delete an existing subscription from a Stream

Parameters

- **url** (*str*) – url used to create/delete a subscription
- **stream_id** (*str*) – represents a given stream by its id

Raises

ValueError – when a stream_id is undefined:

Examples

Creating a new Subscription directly:

```
>>> subscription = Subscription(<stream_id>)
>>> created_subs = subscription.create(
        headers={'authorization': 'user-key'}
    )
>>> print(created_subs)
>>> { "id": "dj-synhub-extraction-*HH**", "type": "subscription" }
```

SUBSCRIPTION_IDX = 0

consume_async_messages(*callback=None*, *ack_enabled=False*)

Consume async messages is a listener function.

Consume async messages is a listener function which consumes the current messages (News) from a pub-sub subscription in async

Parameters

- **callback** (*function*) – is used for processing a message
- **ack_enabled** (*boolean*) – is used for acknowledging a message

Raises

RuntimeError – when listener is not yet init:

consume_messages(*callback=None*, *maximum_messages=None*, *batch_size=None*, *ack_enabled=False*)

Consume messages from a pubsub subscription in sync.

Consume messages is a listener function which consumes the current messages (News) from a pubsub subscription in sync

Parameters

- **callback** (*function*) – is used for processing a message
- **maximum_messages** (*int*) – is used for consuming a specific number of messages
- **batch_size** (*int*) – the limit of the batch expected
- **ack_enabled** (*boolean*) – is used for acknowledging a message

Raises

RuntimeError – when listener is not yet init:

create(*headers=None*)

Create a subscription for a given stream instance.

Create subscription allows a user to create another subscription to a given stream

Parameters
headers (*dict*) – which contains the token/acces key for authorization

Returns

- *Data which contains*
- *subscription's id and type created*

Raises

- **ValueError** – when a stream_id is undefined:
- **RuntimeError** – when Unexpected API response happens:

create_listener(*user*)

Create a listener in a separate step.

Create listener allows to create a listener in a separate step for avoiding undefined subscription id

Parameters
user (*StreamUser*) – user which possess access to any credentials/client needed for listener

Raises
RuntimeError – when user is not a StreamUser:

delete(*headers=None*) → bool

Delete subscription for a given stream.

Delete subscription allows a user to delete a subscription to a given stream

Parameters
headers (*dict*) – which contains the token/acces key for authorization

Return type
bool value which shows if the subscription is complete deleted

Raises
RuntimeError – when Unexpected API response happens:

id = None

listener = None

stream_id = None

subscription_type = None

6.3 Listener

class factiva.news.stream.**Listener**(*subscription_id=None*, *stream_user=None*)

Bases: object

Class that represents a Listener for Google Pubsub.

Parameters

- **stream_user** (*Stream User*) – constructor will asign a stream user which has the access to the proper url and headers which are going to be used for:
  - Checking the exceeded documents

- Consuming messages (articles) in sync
- Consuming messages (articles) in async

- **subscription_id** (*str*) – is used by Pubsub to consume messages in async/sync

Examples

Creating a new Listener directly:

```
>>> listener = Listener(
        stream_user=StreamUser(
            user_key='**************************1234',
            request_info=False,
            user_id='******-svcaccount@dowjones.com',
            client_id='**************************5678',
            password='*******'
        )
    )
>>> def callback(message, subscription_id):
>>>     print('Subscription ID: {}: Message: {}'.format(
            subscription_id, message
        ))
>>> print(listener.listen(
        callback,
        subscription_id='<subscription-id>',
        maximum_messages=10
        ))
Received news message with ID: DJDN******************
Subscription ID: dj-synhub-stream-********************_
km******-filtered-******:
Message: {'an': 'DJDN0********************',
'document_type': 'article', 'action': 'rep',
'source_code': 'DJDN', 'source_name': 'Dow Jones ---- -----',
'publication_date': '2021-05-20T08:00:10.255Z',
'publication_datetime': '2021-05-20T08:00:10.255Z',
'modification_date': '2021-05-20T08:04:56.175Z',
'modification_datetime': '2021-05-20T08:02:54.000Z',
'ingestion_datetime': '2021-05-20T08:00:13.000Z',
'title': "----- ------ ------ ------ 2020",
'snippet': '', 'body': "\nOn Thursday -- --- ----,
--- Plc. announced its ----- ---- --- \n
......
}
```

FIRST_OBJECT = 0

check_exceeded_thread()

Check exceeded thread function.

creates threads for checking if the doc count has been exceeded

listen(*callback=<function default_callback>*, *maximum_messages=None*, *batch_size=10*, *ack_enabled=False*)

Listen function.

listens the current messages (News) from a pubsub subscription in sync

Parameters

- **callback** (*function*) – is used for processing a message
- **maximum_messages** (*int*) – is used for consuming a specific number of messages
- **batch_size** (*int*) – the limit of the batch expected
- **ack_enabled** (*boolean*) – flag for acknowledge a message

Raises

- **ValueError:** – When maximum_messages is undefined
- **GoogleAPICallError:** – When there is no valid instance to pull from When something unexpected happened with Pubsub client

listen_async(*callback=<function default_callback>*, *ack_enabled=False*)

Listen async function.

listens the current messages (News) from a pubsub subscription in async

Parameters

- **callback** (*function*) – is used for processing a message
- **ack_enabled** (*boolean*) – flag for acknowledge a message

property stream_id_uri

Property for retrieving the stream id uri.

CHAPTER

SEVEN

NEWS.TAXONOMY

7.1 Taxonomy

class factiva.news.taxonomy.**Taxonomy**(*user_key=None*)

Bases: object

Class that represents the taxonomy available within the Snapshots API.

Parameters

user_key (*str or UserKey*) – String containing the 32-character long APi Key. If not provided, the constructor will try to obtain its value from the FACTIVA_USERKEY environment variable.

Examples

Creating a taxonomy instance providing the user key

```
>>> t = Taxonomy(u='abcd1234abcd1234abcd1234abcd1234')
```

Creating a taxonomy instance with an existing UserKey instance

```
>>> u = UserKey('abcd1234abcd1234abcd1234abcd1234')
>>> t = Taxonomy(user_key=u)
```

categories = []

get_categories() → list

Request for a list of available taxonomy categories.

Return type

List of available taxonomy categories.

Raises

RuntimeError – When API request returns unexpected error:

Examples

This method is called with in the __init__ method, so the categories can be accessed as is.

```
>>> taxonomy = Taxonomy()
>>> print(taxonomy.categories)
['news_subjects', 'regions', 'companies', 'industries', 'executives']
```

Calling the method on its own

```
>>> taxonomy = Taxonomy()
>>> print(taxonomy.get_categories())
['news_subjects', 'regions', 'companies', 'industries', 'executives']
```

get_category_codes(*category*) → DataFrame

Request for available codes in the taxonomy for the specified category.

Parameters
category (*str*) – String with the name of the taxonomy category to request the codes from

Return type
Dataframe containing the codes for the specified category

Raises

- **ValueError** – When category is not of a valid type:
- **RuntimeError** – When API request returns unexpected error:

Examples

Getting the codes for the 'industries' category

```
>>> taxonomy = Taxonomy()
>>> industry_codes = taxonomy.get_category_codes('industries')
>>> print(industry_codes)
        code                description
0     i25121             Petrochemicals
1     i14001          Petroleum Refining
2       i257             Pharmaceuticals
3     iphrws  Pharmaceuticals Wholesale
4       i643      Pharmacies/Drug Stores
```

get_company(*code_type*, *company_codes*) → DataFrame

Request information about either a single company or a list of companies.

Parameters

- **code_type** (*str*) – String describing the code type used to request the information about the company. E.g. isin, ticker.
- **company_code** (*str or list*) – Single company code (str) or list of company codes to translate.

Return type
Dataframe with the information about the requested company(ies)

Raises

- **ValueError** – When any given argument is not of the expected type:
- **RuntimeError:** –
  - When both company and companies arguments are set - When API request returns unexpected error

Examples

Get data for a single company using the code type 'isin'

```
>>> taxonomy = Taxonomy()
>>> single_company_data = taxonomy.get_company('isin', company_code=
↪'ABCNMST00394')
>>> print(single_company_data)
             id  fcode           common_name
0  ABCNMST00394  ABCYT  Systemy Company S.A.
```

Get data for multiple companies sugin the code type 'isin'

```
>>> taxonomy = Taxonomy()
>>> multiple_companies_data = taxonomy.get_company('isin', company_codes=[
↪'ABC3E53433100', 'XYZ233341067', 'MN943181045'])
>>> print(multiple_companies_data)
             id   fcode      common_name
0  ABC3E5343310  MCABST  M**************
1  XYZ233341067   AXYZC    A************
2  MN9431810453     MNM     M***********
```

get_identifiers() → list

Request for a list of available taxonomy categories.

Return type
List of available taxonomy categories.

Raises
RuntimeError – When API request returns unexpected error:

Examples

This method is called with in the __init__ method, so the categories can be accessed as is.

```
>>> taxonomy = Taxonomy()
>>> print(taxonomy.categories)
['news_subjects', 'regions', 'companies', 'industries', 'executives']
```

Calling the method on its own

```
>>> taxonomy = Taxonomy()
>>> print(taxonomy.get_categories())
['news_subjects', 'regions', 'companies', 'industries', 'executives']
```

get_multiple_companies(*code_type*, *company_codes*) → DataFrame

Request information about a list of companies.

Parameters

- **code_type** (*str*) – String describing the code type used to request the information about the company. E.g. isin, ticker.
- **companies_codes** (*list*) – List containing the company codes to request information about

Return type
DataFrame containing the company information

Raises
RuntimeError – When API request returns unexpected error:

Examples

Get multiple companies data using the code type 'isin' and a company codes list

```
>>> taxonomy = Taxonomy()
>>> companies_data = taxonomy.get_multiple_companies('isin', ['ABC3E53433100
↪', 'XYZ233341067', 'MN943181045'])
>>> print(companies_data)
             id   fcode      common_name
0  ABC3E5343310  MCABST  M**************
1  XYZ233341067   AXYZC    A************
2  MN9431810453     MNM     M***********
```

get_single_company(*code_type*, *company_code*) → DataFrame

Request information about a single company.

Parameters

- **code_type** (*str*) – String describing the code type used to request the information about the company. E.g. isin, ticker.
- **company_code** (*str*) – String containing the company code

Return type
DataFrame containing the company information

Raises
RuntimeError – When API request returns unexpected error:

Examples

Get the company data using the code type 'isin' and the company code 'ABCNMST00394'

```
>>> taxonomy = Taxonomy()
>>> company_data = taxonomy.get_single_company('isin', 'ABCNMST00394')
>>> print(company_data)
             id  fcode           common_name
0  ABCNMST00394  ABCYT  Systemy Company S.A.
```

```
identifiers = []
```

7.2 Company

class factiva.news.taxonomy.**Company**(*user_key=None*)

Bases: object

Class that represents the company available within the Snapshots API.

Parameters

user_key (*str or UserKey*) – String containing the 32-character long APi Key. If not provided, the constructor will try to obtain its value from the FACTIVA_USERKEY environment variable.

Examples

Creating a company instance providing the user key

```
>>> c = Company(u='abcd1234abcd1234abcd1234abcd1234')
```

Creating a company instance with an existing UserKey instance

```
>>> u = UserKey('abcd1234abcd1234abcd1234abcd1234', True)
>>> c = Company(user_key=u)
```

point_in_time_download_all(*identifier*, *file_name*, *file_format*, *to_save_path=None*, *add_timestamp=False*) → str

Returns a file with the historical and current identifiers for each category and news coded companies.

Parameters

- **identifier** (*str*) – A company identifier type
- **file_name** (*str*) – Name to be used as local filename
- **file_format** (*str*) – Format of the file
- **to_save_path** (*str, optional*) – Path to be used to store the file
- **add_timestamp** (*bool, optional*) – Flag to determine if include timestamp info at the filename

Returns

Dowloaded file path

Return type

str

Raises

- **ValueError** – When the user is not allowed to permorm this operation:
- **ValueError** – When the identifier requested is not valid:
- **ValueError** – When the format file requested is not valid:

point_in_time_query(*identifier*, *value*) → dict

Returns the resolved Factiva code and date ranges when the instrument from the identifier, was valid.

Parameters

- **identifier** (*str*) – A company identifier type
- **value** (*str*) – Identifier value

Returns

Factiva code and date ranges from a company

Return type

dict

Raises

- **ValueError** – When the user is not allowed to permorm this operation:
- **ValueError** – When the identifier requested is not valid:

user_key = None

validate_point_time_request(*identifier*)

Validate if the user is allowes to perform company operation and if the identifier given is valid

Parameters

identifier (*str*) – A company identifier type

Raises

- **ValueError** – When the user is not allowed to permorm this operation:
- **ValueError** – When the identifier requested is not valid:

INDEX

I

L

N

P

Q

S

T

U

V