polygon

Release 0.8.2

P S Solanki

Oct 23, 2021

CONTENTS:


GitHub

CHAPTER

ONE

GETTING STARTED

Welcome to `polygon`. Read this page to quickly install and configure this library to write your first Polygon Python application.

It is highly recommended to read this page to the full as it has important information

1.1 What you need to have

1. A polygon.io account and your `API key`. Find your api key on Your Dashboard
2. Python version 3.6 or higher. Don't have it installed? Install python

Once you have done these, Proceed to the installation of the library. Skip if already done.

1.2 Installing `polygon`

The recommended method of installation for all users is to install using `pip` from PyPi. A virtual environment is highly recommended but not a necessity.

run the below command in terminal (same for all OS)

```
pip install polygon
```

To confirm the install worked, try importing the package as such

```
import polygon
```

If this doesn't throw any errors, the install worked. You may proceed to next steps now.

1.3 General guide for clients

This section would provide general guidance on the clients without going into specific endpoints as stocks or options.

As you already know polygon.io has two major classes of APIs. The `REST` APIs and `websockets` streaming APIs.

This library implements all of them.

- For REST HTTP endpoints
  - Regular client is implemented for all endpoints.
  - Support for `async` client is also provided. See *Async Support for REST endpoints* for more.

- For websocket streaming endpoints
  - a `callback` based stream client is implemented. See *Callback Streaming*
  - an async based stream client is also implemented. See *Async Streaming*

Be sure to check out our special section *What the Hell are Enums Anyways* for info on `enums` which will be used in many functions in this library to avoid passing error prone data.

A detailed description of how to use the streaming endpoints is provided in the streamer docs linked above.

Need examples? The github repository has a few you could use.

1.3.1 Creating and Using REST HTTP clients

This section aims to outline the general procedure to create and use the http clients in both regular and async programming methods.

First up, you'd import the library. There are many ways to import names from a library and it is highly recommended to complete fundamental python if you're not aware of them.

```
import polygon
```

Now creating a client is as simple as (using stocks and forex clients as examples here)

1. Regular client: `stocks_client = polygon.StocksClient('API_KEY')`
2. Async client: `forex_client = polygon.ForexClient('API_KEY', True)`

You can also specify timeouts on requests. By default the timeout is set to 10 seconds for both connection timeout and read timeout which should be fine for most people. You can specify both connect and read OR either one of them. If you're unsure of what this implies, it's just the max time limit to specify for a request. Don't change it unless you know you need to.

```
# client with a custom timeout. Default is 10 seconds
client = polygon.StocksClient('api_key', connect_timeout=15)

# another one
client = polygon.StocksClient('api_key', connect_timeout=5, read_timeout=5)

# An async one now
client = polygon.StocksClient('key', True, read_timeout=5)

# another async one
client = polygon.StocksClient('key', True, read_timeout=5, connect_timeout=15)
```

Note that It is NOT recommended to hard code your API key or other credentials into your code unless you really have a use case. Instead preferably do one of the following:

1. create a separate python file with credentials, import that file into main file and reference using variable names.
2. Use environment variables.

Now that you have a client, simply call its methods to get data from the API

```
current_price = stocks_client.get_current_price('AMD')
print(f'Current price for AMD is {current_price}')
```

Note that you can have instances of all 5 different types of http clients together. So you can create client for each one of the stocks, options and other APIs

All the clients in the lib support context managers

```
with polygon.StocksClient('KEY') as client:
    last_quote = client.get_last_quote('AMD)
    print(f'Last quote for AMD: {last_quote}')
```

Using context managers ensures that the connections opened up to make requests are closed properly.

You can manually close the connections if you're not using context managers:

1. for regular non-async: `client.close()`
2. for async: `await client.async_close()`

This is not an absolute necessity but rather a good software practice to close out resources when you don't need them.

1.4 Calling the methods/functions

Most methods and functions have sane default values which can be customized as needed. Required parameters need to be supplied as positional arguments (which just means that the order of arguments matter when passing more than one).

Parameters which have special values are supplied as python enums. You can however always pass in your own values but it is recommended to use enums as they mitigate the possibilities of an error.

All enums are available in the module `polygon.enums` and can be imported the way you like.

If you're still unsure about enums, see our dedicated section: *What the Hell are Enums Anyways*

1.5 Return Values

Most methods would by default return a dictionary/list object containing the data from the API. If you need the underlying response object you need to pass in `raw_response=True` in the function call. It might be useful for checking `status_code` or inspecting `headers`.

For 99% users, the default should be good enough.

The underlying response object returned is `requests.models.Response` for regular client and `httpx.Response` for async client. Using `.json()` on the response object gets you the data dict/list

Once you have the response, you can utilize the data in any way that you like. You can push it to a database, create a pandas dataframe, save it to a file or process it the way you like.

Every method's documentation contains a direct link to the corresponding official documentation page where you can see what the keys in the response mean.

1.6 Pagination Support

So quite a few endpoints implement pagination for large response and hence the library implements a simple mechanism to get next page of the response. (support for previous page is also available but not all endpoints will have previous page implementation. The documentation will mention which endpoint has which kinda pagination implementation so make sure you read that)

This blog by polygon explains a few concepts around pagination and other query extensions. A good read overall.

The pagination function simply parses the `next_url` attribute (for next page) and `previous_url` attribute (for previous page) and send an authorized request using your key as header.

The functions will return `False` if there is no next/previous page remaining or the endpoint doesn't support pagination.

All REST clients have these functions and you will use the same function name for all endpoints. See examples below

first here is how the functions for pagination look like: (click on names to see definition - you won't have to import them with this name. They are avaiable with the client you create as shown in examples below)

for usual client: *polygon.base_client.BaseClient.get_next_page()* || *polygon.base_client.BaseClient.get_previous_page()*

For async client: *polygon.base_client.BaseClient.async_get_next_page()* || *polygon.base_client.BaseClient.async_get_previous_page()*

Examples Use

```
# assuming a client is created already
data = client.get_trades(<blah-blah>)

next_page_of_data = client.get_next_page(data)  # getting NEXT page
previous_page_of_data = client.get_previous_page(data)  # getting PREVIOUS page

# ASYNC example
await client.async_get_next_page(data)
await client.async_get_previous_page(data)

# It's wise to check if the value returned is not False.
```

In practice, to get all pages (either next or previous), you'll need a while loop An example:

```
responses = []

response = client.get_trades(<blah-blah>)  # using get_trades as example. you can use it␣
↪on all methods which support pagination
responses.append(response)  # using a list to store all the pages of response. You can␣
↪use your own approach here.

while 'next_url' in response.keys():  # change to 'previous_url' for previous pages
    response = client.get_next_page(response)  # similarly change to get_previous_page␣
↪for previous pages.

    responses.append(response)  # adding further responses to our list. you can use your␣
↪own approach.

print('all pages received.')
```

1.7 Async Support for REST endpoints

As you saw above in the example, the clients have methods for each endpoint. The usual client is a sync client. However support for async is also provided for all the endpoints on all the clients.

Here is how to make use of it (**This info is applicable to ALL rest clients**)

First up, you'd create a client. Earlier you created a client by passing in just your API key. Here you'd create the client with an additional argument.

so instead of something like: `StocksClient('API_KEY')`, you'd do

```
client = StocksClient('KEY', True)    # or use_async=True for second parameter
```

This gives you an async client. Similar to sync, you can have all 5 different clients together.

ALL the methods you'd use for async client have `async_` in front of their sync counterpart names. so `async_get_trades`, `async_get_snapshot` and so on...

So if a method is named `get_trades()` in usual client, in async client you'd have it as `async_get_trades()` and this behavior is true for all methods

Here is how you can use it grab the current price of a symbol

```
import polygon

async def main():
    stocks_client = polygon.StocksClient('API_KEY', True)

    current_price = await stocks_client.async_get_current_price('AMD')
    print(current_price)

if __name__ == '__main__':
    import asyncio
    asyncio.run(main())
```

Note that I'm working towards avoiding this name difference across sync and async clients. Feedback is appreciated.

1.8 Special Points

- All the date parameters in any method/function in the library can be supplied as `datetime.date` or `datetime.datetime` You may also pass in a string in format: `YYYY-MM-DD`.
- You would notice some parameters having `lt`, `lte`, `gt` and `gte` in their names. Those parameters are supposed to be filters for `less than`, `less than or equal to`, `greater than`, `greater than or equal to` respectively. To know more see heading **Query Filter Extensions** in This blog post by polygon To explain: imagine a parameter: `fill_date_lt`. now the date you'll supply would be a filter for values less than the given value and hence you'd get results which have fill_date less than your specified value, which in this case is a date.
- Some endpoints may not return a dictionary and instead return a `list`. The number of such endpoints is very low. Similarly get current price returns a float/integer. I'm working towards reflecting the same in individual method's docs.
- It is highly recommended to use the polygon.io documentation website's quick test functionality to play around with the endpoints.

- Type hinting in function/method definitions indicate what data type does that parameter is supposed to be. If you think the type hinting is incomplete/incorrect, let me know. For example you might ses: `cost: int` which means this parameter `cost` is supposed to be an integer. `adjusted: bool` is another example for a boolean (either `True` or `False`)
- You'll notice some type hints having `Union` in them followed by two or more types inside a square bracket. That simply means the parameter could be of any type from that list in bracket . For example: `price: Union[str, float, int]` means the parameter `price` could be either a string, a float or an integer. You'd notice Union type hints more on return types of the functions/methods.

so far so good? Start by taking a look at the complete docs for endpoints you need. Here is a quick list

- *Stocks*
- *Options*
- *Forex* and *Crypto*
- *Callback Streaming* and *Async Streaming*
- *What the Hell are Enums Anyways*

CHAPTER

TWO

STOCKS

So you have completed the initial steps and are ready to dive deep into endpoints. Read this page to know everything you need to know about using the various Stocks HTTP endpoints.

See *Async Support for REST endpoints* for asynchronous use cases.

Docs below assume you have already read getting started page and know how to create the client. If you do not know how to create the client, first see *General guide for clients* and create client as below. As always you can have all 5 different clients together.

```
import polygon

stocks_client = polygon.StocksClient('KEY')  # for usual sync client
async_stock_client = polygon.StocksClient('KEY', True)  # for an async client
```

2.1 Get Trades

StocksClient.**get_trades**(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get trades for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values. Official Docs

Parameters

- **symbol** – The ticker symbol we want trades for.
- **date** – The date/day of the trades to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **timestamp** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the next page of results. Default: None. I'm trying to think of a good way to implement pagination support for this type of pagination.
- **timestamp_limit** – The maximum timestamp allowed in the results. Default: None
- **reverse** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **limit** – Limit the size of the response, max 50000 and default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.2 Get Quotes

`StocksClient.get_quotes`(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get Quotes for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values. Official Docs

Parameters

- **`symbol`** – The ticker symbol we want quotes for.
- **`date`** – The date/day of the quotes to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **`timestamp`** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the next page of results. Default: None. Thinking of a good way to implement this pagination here.
- **`timestamp_limit`** – The maximum timestamp allowed in the results. Default: None
- **`reverse`** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **`limit`** – Limit the size of the response, max 50000 and default 5000.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.3 Get Last Trade

`StocksClient.get_last_trade`(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent trade for a given stock. Official Docs

Parameters

- **`symbol`** – The ticker symbol of the stock/equity.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.4 Get last Quote

StocksClient.**get_last_quote**(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent NBBO (Quote) tick for a given stock. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.5 Get Daily Open Close

StocksClient.**get_daily_open_close**(*symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the OCHLV and after-hours prices of a stock symbol on a certain date. Official Docs

Parameters

- **symbol** – The ticker symbol we want daily-OCHLV for.
- **date** – The date/day of the daily-OCHLV to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.6 Get Aggregate Bars (Candles)

StocksClient.**get_aggregate_bars**(*symbol: str*, *from_date*, *to_date*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *multiplier: int = 1*, *timespan='day'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a stock over a given date range in custom time window sizes. For example, if `timespan = 'minute'` and `multiplier = '5'` then 5-minute bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **from_date** – The start of the aggregate time window. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime` or `date` or string `YYYY-MM-DD`

- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. See *polygon.enums.SortOrder* for choices. asc default.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **multiplier** – The size of the timespan multiplier. Must be a positive whole number.
- **timespan** – The size of the time window. See *polygon.enums.Timespan* for choices. defaults to day
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

2.7 Get Grouped daily Bars (Candles)

StocksClient.**get_grouped_daily_bars**(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily OCHLV for the entire stocks/equities markets. Official docs

Parameters

- **date** – The date to get the data for. Could be datetime or date or string YYYY-MM-DD
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

2.8 Get Previous Close

StocksClient.**get_previous_close**(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's OCHLV for the specified stock ticker. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.9 Get Snapshot

`StocksClient.get_snapshot(symbol: str, raw_response: bool = False)` → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded stock ticker. Official Docs

Parameters

- **`symbol`** – The ticker symbol of the stock/equity.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.10 Get Snapshot (All)

`StocksClient.get_snapshot_all(symbols: list, raw_response: bool = False)` → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded stock symbols. Official Docs

Parameters

- **`symbols`** – A comma separated list of tickers to get snapshots for.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

2.11 Get Current Price

`StocksClient.get_current_price(symbol: str)` → float

get current market price for the ticker symbol specified.

Uses *`get_last_trade()`* under the hood Official Docs

Parameters **`symbol`** – The ticker symbol of the stock/equity.

Returns The current price. A `KeyError` indicates the request wasn't successful.

2.12 Get Gainers & Losers

StocksClient.**get_gainers_and_losers**(*direction='gainers'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current top 20 gainers or losers of the day in stocks/equities markets. Official Docs

Parameters

- **direction** – The direction of results. Defaults to gainers. See *polygon.enums.SnapshotDirection* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

CHAPTER

THREE

OPTIONS

Read this page to know everything you need to know about using the various Options HTTP endpoints.

See *Async Support for REST endpoints* for asynchronous use cases.

Docs below assume you have already read getting started page and know how to create the client. If you do not know how to create the client, first see *General guide for clients* and create client as below. As always you can have all 5 different clients together.

```
import polygon

options_client = polygon.OptionsClient('KEY')  # for usual sync client
async_options_client = polygon.OptionsClient('KEY', True)  # for an async client
```

3.1 Creating Option Symbols

So when you're working with options (rest/websockets), you'll certainly need the option symbols which contain the information about their underlying symbol, expiry, call_or_put and the strike price in a certain format. Many organizations tend to use different formats to represent these.

Polygon.io tends to use This Format . For those who want to understand how this formatting works, Here is a guide (thanks to Ian from their support team).

Fortunately for you, the library comes with a few functions to help ya out with it. **first function in that list is creating an option symbol**

The library also has two bonus functions which allow you to create and parse option symbols using the format supported by TD Ameritrade. See below for more info on how to use them.

Note that polygon has a rest endpoint in reference API to get all active contracts which you can filter based on many values.

You might have noticed (you didn't notice, did ya?) that polygon endpoints expect a prefix: `O:` before option symbols. For convenience, this library handles all of it internally. what that means for you is that you can pass in option symbols **with or without the prefix O:** and both will be handled. In the below function, you can make the argument `prefix_o=True` to get the prefix in the output. By defaults it returns this format: `AMD211205P00149000` (example symbol)

here is how the function looks. just supply the details.

`polygon.options.options.`**`build_option_symbol`**(*underlying_symbol: str*, *expiry*, *call_or_put*, *strike_price*, *prefix_o: bool = False*)

Build the option symbol from the details provided.

Parameters

- **underlying_symbol** – The underlying stock ticker symbol.
- **expiry** – The expiry date for the option. You can pass this argument as `datetime.datetime` or `datetime.date` object. Or a string in format: `YYMMDD`. Using datetime objects is recommended.
- **call_or_put** – The option type. You can specify: `c` or `call` or `p` or `put`. Capital letters are also supported.
- **strike_price** – The strike price for the option. ALWAYS pass this as one number. `145`, `240.5`, `15.003`, `56`, `129.02` are all valid values. It shouldn't have more than three numbers after decimal point.
- **prefix_o** – Whether or not to prefix the symbol with 'O:'. It is needed by polygon endpoints. However all the library functions will automatically add this prefix if you pass in symbols without this prefix.

Returns The option symbol in the format specified by polygon

Example use:

```
from polygon import build_option_symbol

symbol = build_option_symbol('AMD', date(year=2021, month=12, day=5), 'c', 158)  # date␣
↪is just a datetime.date object

# another one!
symbol = build_option_symbol('NVDA', '211205', 'call', 124.56)
# you can use these variable as you like on polygon's endpoints
```

Bonus Function to create option symbols in TD Ameritrade formatting:

don't use this formatting on polygon endpoints. only on tda. this is just a bonus function.

polygon.options.options.**build_option_symbol_for_tda**(*underlying_symbol: str*, *expiry*, *call_or_put*, *strike_price*)

Only use this function if you need to create option symbol for TD ameritrade API. This function is just a bonus.

Parameters

- **underlying_symbol** – The underlying stock ticker symbol.
- **expiry** – The expiry date for the option. You can pass this argument as `datetime.datetime` or `datetime.date` object. Or a string in format: `MMDDYY`. Using datetime objects is recommended.
- **call_or_put** – The option type. You can specify: `c` or `call` or `p` or `put`. Capital letters are also supported.
- **strike_price** – The strike price for the option. ALWAYS pass this as one number. `145`, `240.5`, `15.003`, `56`, `129.02` are all valid values. It shouldn't have more than three numbers after decimal point.

Returns The option symbol built in the format supported by TD Ameritrade.

Example use:

```
from polygon import build_option_symbol_for_tda

symbol = build_option_symbol_for_tda('AMD', date(year=2021, month=12, day=5), 'c', 158)
↪# date is just a datetime.date object
```

(continues on next page)

(continued from previous page)

```
# another one!
symbol = build_option_symbol_for_tda('NVDA', '120522', 'call', 124.56)
```

3.2 Parsing Option Symbols

So the above function was to build an option symbol from details. This function would help you do the opposite. That is, extracting information from an option symbol.

This function parses the symbol based on This spec. Note that you can pass the value with or without the `O:` prefix. The lib would handle that like it does everywhere else.

Important So it appears that some option symbols as returned by polygon endpoints happen to have a **correction number** within the symbol. The additional number is always between the underlying symbol and expiry. **The lib handles that for you** and hence returns the corrected parsed symbol.

To elaborate: sometimes you'd see something like: `MS1221015C00234000`. Notice the extra 1 right after symbol MS and before expiry 221015. This symbol should actually be `MS221015C00234000` without that 1 (which could be any number based on the info I have from support team).

If you ever need to get the corrected symbol without that additional number, use the lib to parse the symbol and the attribute `option_symbol` would contain the full option symbol without the extra number and any prefixes.

By default the expiry date in the results would be a `datetime.date` object. Change it to `string` to get a string in format `YYYY-MM-DD`

You can choose to get your output in any one out of 3 different formats provided by the lib. To change the format, change the output_format arg in the function below.

The OptionSymbol object (default) by default it would return a *polygon.options.options.OptionSymbol* object. The object would allow you to access values using attributes. For example: `parsed_symbol.expiry`, `parsed_symbol.underlying_symbol`, `parsed_symbol.strike_price`, `parsed_symbol.call_or_put` and `parse_symbol.option_symbol`

output as a list You can also choose to get your output as a `list`. The list would just have all the parsed values as: `[underlying_symbol, expiry, call_or_put, strike_price, option_symbol]`

output as a dict You can also choose to get your results as a `dict`. The dict will have all the values as usual pairs. keys would be: `'underlying_symbol', 'strike_price', 'expiry', 'call_or_put', 'option_symbol'`

While other values are self explanatory, the value `option_symbol` in parsed symbol is simply the full option symbol without any extra correction numbers or prefixes. For example if you passed in `MS221015C00234000`, option_symbol attribute will have the exact same value supplied. If you passed `MS1221015C00234000` or `O:MS221015C00234000`, option_symbol would have `MS221015C00234000` removing those extra numbers and prefixes.

here is how the function looks.

polygon.options.options.**parse_option_symbol**(*option_symbol: str*, *output_format='object'*, *expiry_format='date'*)

Function to parse an option symbol.

Parameters

- **option_symbol** – the symbol you want to parse. Both `TSLA211015P125000` and `O:TSLA211015P125000` are valid

- **output_format** – Output format of the result. defaults to object. Set it to `dict` or `list` as needed.
- **expiry_format** – The format for the expiry date in the results. Defaults to `date` object. change this param to `string` to get the value as a string: `YYYY-MM-DD`

Returns The parsed values either as an object, list or a dict as indicated by `output_format`.

Example use:

```
from polygon import (build_option_symbol, parse_option_symbol)

parsed_details = parse_option_symbol('AMD211205C00156000')

# another one!
parsed_details = parse_option_symbol('AMD211205C00156000', output_format=list)

# another one!
parsed_details = parse_option_symbol('AMD211205C00156000', dict, expiry_format=str)
```

bonus function to parse symbols in TD ameritrade format

The output_format and expiry_format are both exactly the same as above. Only difference is in the formatting.

polygon.options.options.**parse_option_symbol**(*option_symbol: str*, *output_format='object'*, *expiry_format='date'*)

Function to parse an option symbol.

Parameters

- **option_symbol** – the symbol you want to parse. Both `TSLA211015P125000` and `O:TSLA211015P125000` are valid
- **output_format** – Output format of the result. defaults to object. Set it to `dict` or `list` as needed.
- **expiry_format** – The format for the expiry date in the results. Defaults to `date` object. change this param to `string` to get the value as a string: `YYYY-MM-DD`

Returns The parsed values either as an object, list or a dict as indicated by `output_format`.

Example use:

```
from polygon import parse_option_symbol_from_tda

parsed_details = parse_option_symbol_from_tda('GOOG_012122P620')

# another one!
parsed_details = parse_option_symbol_from_tda('TSLA_112020C1360', output_format=list)

# another one!
parsed_details = parse_option_symbol_from_tda('SPY_121622C335', dict, expiry_format=str)
```

3.3 Get Trades

This endpoint supports pagination. The library has support for pagination. See *Pagination Support* for info and examples

`OptionsClient.get_trades`(*option_symbol: str*, *timestamp=None*, *timestamp_lt=None*, *timestamp_lte=None*, *timestamp_gt=None*, *timestamp_gte=None*, *sort='timestamp'*, *limit: int = 100*, *order='asc'*, *raw_response: bool = False*)

Get trades for an options ticker symbol in a given time range. Note that you need to have an option symbol in correct format for this endpoint. You can use *polygon.reference_apis.reference_api.ReferenceClient.get_option_contracts()* to query option contracts using many filter parameters such as underlying symbol etc. Official Docs

Parameters

- **option_symbol** – The options ticker symbol to get trades for. for eg `O:TSLA210903C00700000`. you can pass the symbol with or without the prefix `O:`
- **timestamp** – Query by trade timestamp. You can supply a `date`, `datetime` object or a `nanosecond UNIX timestamp` or a string in format: `YYYY-MM-DD`.
- **timestamp_lt** – query results where timestamp is less than the supplied value
- **timestamp_lte** – query results where timestamp is less than or equal to the supplied value
- **timestamp_gt** – query results where timestamp is greater than the supplied value
- **timestamp_gte** – query results where timestamp is greater than or equal to the supplied value
- **sort** – Sort field used for ordering. Defaults to timestamp. See *polygon.enums.OptionTradesSort* for available choices.
- **limit** – Limit the number of results returned. Defaults to 100. max is 50000.
- **order** – order of the results. Defaults to `asc`. See *polygon.enums.SortOrder* for info and available choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

3.4 Get Last Trade

`OptionsClient.get_last_trade`(*ticker: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent trade for a given options contract. Official Docs

Parameters

- **ticker** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

3.5 Get Previous Close

`OptionsClient.get_previous_close`(*ticker: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified option contract. Official Docs

Parameters

- **`ticker`** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`
- **`adjusted`** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

CHAPTER

FOUR

REFERENCE APIS

Read this page to know everything you need to know about using the various References HTTP endpoints.

See *Async Support for REST endpoints* for asynchronous use cases.

Docs below assume you have already read getting started page and know how to create the client. If you do not know how to create the client, first see *General guide for clients* and create client as below. As always you can have all 5 different clients together.

```
import polygon

reference_client = polygon.ReferenceClient('KEY')  # for usual sync client
async_reference_client = polygon.ReferenceClient('KEY', True)  # for an async client
```

4.1 Get Tickers

This endpoint supports pagination. The library has support for pagination. See *Pagination Support* for info and examples

ReferenceClient.**get_tickers**(*symbol: str = ''*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *symbol_type=''*, *market=''*, *exchange: str = ''*, *cusip: Optional[str] = None*, *cik: str = ''*, *date=None*, *search: Optional[str] = None*, *active: bool = True*, *sort='ticker'*, *order='asc'*, *limit: int = 100*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Query all ticker symbols which are supported by Polygon.io. This API currently includes Stocks/Equities, Crypto, and Forex. Official Docs

Parameters

- **symbol** – Specify a ticker symbol. Defaults to empty string which queries all tickers.
- **ticker_lt** – Return results where this field is less than the value given
- **ticker_lte** – Return results where this field is less than or equal to the value given
- **ticker_gt** – Return results where this field is greater than the value given
- **ticker_gte** – Return results where this field is greater than or equal to the value given
- **symbol_type** – Specify the type of the tickers. See *polygon.enums.TickerType* for common choices. Find all supported types via the Ticker Types API Defaults to empty string which queries all types.
- **market** – Filter by market type. By default all markets are included. See *polygon.enums.TickerMarketType* for available choices.

- **exchange** – Specify the primary exchange of the asset in the ISO code format. Find more information about the ISO codes at the ISO org website. Defaults to empty string which queries all exchanges.
- **cusip** – Specify the CUSIP code of the asset you want to search for. Find more information about CUSIP codes on their website Defaults to empty string which queries all CUSIPs
- **cik** – Specify the CIK of the asset you want to search for. Find more information about CIK codes at their website Defaults to empty string which queries all CIKs.
- **date** – Specify a point in time to retrieve tickers available on that date. Defaults to the most recent available date. Could be `datetime`, `date` or a string `YYYY-MM-DD`
- **search** – Search for terms within the ticker and/or company name. for eg `MS` will match matching symbols
- **active** – Specify if the tickers returned should be actively traded on the queried date. Default is True
- **sort** – The field to sort the results on. Default is ticker. If the search query parameter is present, sort is ignored and results are ordered by relevance. See *polygon.enums.TickerSortType* for available choices.
- **order** – The order to sort the results on. Default is asc. See *polygon.enums.SortOrder* for available choices.
- **limit** – Limit the size of the response, default is 100 and max is 1000. `Pagination` is supported by the pagination function below
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.2 Get Ticker Types

`ReferenceClient.`**`get_ticker_types_v3`**(*asset_class=None*, *locale=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a mapping of ticker types to their descriptive names. Official Docs

Parameters

- **asset_class** – Filter by asset class. see *polygon.enums.AssetClass* for choices
- **locale** – Filter by locale. See *polygon.enums.Locale* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.3 Get Ticker Details

ReferenceClient.**get_ticker_details**(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get details for a ticker symbol's company/entity. This provides a general overview of the entity with information such as name, sector, exchange, logo and similar companies.

This endpoint will be replaced by *get_ticker_details_vx()* in future. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.4 Get Ticker Details vX

ReferenceClient.**get_ticker_details_vx**(*symbol: str*, *date=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

This API is Experimental and will replace *get_ticker_details()* in future.

Get a single ticker supported by Polygon.io. This response will have detailed information about the ticker and the company behind it. Official Docs

Parameters

- **symbol** – The ticker symbol of the asset.
- **date** – Specify a point in time to get information about the ticker available on that date. When retrieving information from SEC filings, we compare this date with the period of report date on the SEC filing. Defaults to the most recent available date.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.5 Get Option Contracts

This endpoint supports pagination. The library has support for pagination. See *Pagination Support* for info and examples

ReferenceClient.**get_option_contracts**(*underlying_ticker: Optional[str] = None*, *ticker: Optional[str] = None*, *contract_type=None*, *expiration_date=None*, *expiration_date_lt=None*, *expiration_date_lte=None*, *expiration_date_gt=None*, *expiration_date_gte=None*, *order='asc'*, *sort=None*, *limit=100*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

List currently active options contracts Official Docs

Parameters

- **underlying_ticker** – Query for contracts relating to an underlying stock ticker.
- **ticker** – Query for a contract by option ticker.
- **contract_type** – Query by the type of contract. see *polygon.enums.OptionsContractType* for choices
- **expiration_date** – Query by contract expiration date. either `datetime`, `date` or string `YYYY-MM-DD`
- **expiration_date_lt** – expiration date less than given value
- **expiration_date_lte** – expiration date less than equal to given value
- **expiration_date_gt** – expiration_date greater than given value
- **expiration_date_gte** – expiration_date greater than equal to given value
- **order** – Order of results. See *polygon.enums.SortOrder* for choices.
- **sort** – Sort field for ordering. See *polygon.enums.OptionsContractsSortType* for choices.
- **limit** – Number of results to return
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.6 Get Ticker News

This endpoint supports pagination. The library has support for pagination. See *Pagination Support* for info and examples

`ReferenceClient.get_ticker_news`(*symbol: Optional[str] = None*, *limit: int = 100*, *order='desc'*, *sort='published_utc'*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *published_utc=None*, *published_utc_lt=None*, *published_utc_lte=None*, *published_utc_gt=None*, *published_utc_gte=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent news articles relating to a stock ticker symbol, including a summary of the article and a link to the original source. Official Docs

Parameters

- **symbol** – To get news mentioning the name given. Defaults to empty string which doesn't filter tickers
- **limit** – Limit the size of the response, default is 100 and max is 1000. Use pagination helper function for larger responses.
- **order** – Order the results. See *polygon.enums.SortOrder* for choices.
- **sort** – The field key to sort. See *polygon.enums.TickerNewsSort* for choices.
- **ticker_lt** – Return results where this field is less than the value.
- **ticker_lte** – Return results where this field is less than or equal to the value.

- **ticker_gt** – Return results where this field is greater than the value
- **ticker_gte** – Return results where this field is greater than or equal to the value.
- **published_utc** – A date string `YYYY-MM-DD` or `datetime` for published date time filters.
- **published_utc_lt** – Return results where this field is less than the value given
- **published_utc_lte** – Return results where this field is less than or equal to the value given
- **published_utc_gt** – Return results where this field is greater than the value given
- **published_utc_gte** – Return results where this field is greater than or equal to the value given
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.7 Get Stock dividends

`ReferenceClient.`**`get_stock_dividends`**(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a list of historical dividends for a stock, including the relevant dates and the amount of the dividend. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.8 Get Stock Financials

`ReferenceClient.`**`get_stock_financials`**(*symbol: str*, *limit: int = 100*, *report_type=None*, *sort=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historical financial data for a stock ticker. This API will be replaced by *`get_stock_financials_vx()`* in future. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **limit** – Limit the number of results. Defaults to 100
- **report_type** – Specify a type of report to return. see *`polygon.enums.StockReportType`* for choices. Defaults to `None`
- **sort** – The key for sorting the results. see *`polygon.enums.StockFinancialsSortType`* for choices.

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.9 Get Stock financials vX

`ReferenceClient.get_stock_financials_vx`(*ticker: Optional[str] = None*, *cik: Optional[str] = None*, *company_name: Optional[str] = None*, *company_name_search: Optional[str] = None*, *sic: Optional[str] = None*, *filing_date=None*, *filing_date_lt=None*, *filing_date_lte=None*, *filing_date_gt=None*, *filing_date_gte=None*, *period_of_report_date=None*, *period_of_report_date_lt=None*, *period_of_report_date_lte=None*, *period_of_report_date_gt=None*, *period_of_report_date_gte=None*, *time_frame=None*, *include_sources: bool = False*, *order='asc'*, *limit: int = 50*, *sort='filing_date'*, *raw_response: bool = False*)

Get historical financial data for a stock ticker. The financials data is extracted from XBRL from company SEC filings using this methodology Official Docs

This API is experimental and will replace *get_stock_financials()* in future.

Parameters

- **ticker** – Filter query by company ticker.
- **cik** – filter the Query by `central index key (CIK)` Number
- **company_name** – filter the query by company name
- **company_name_search** – partial match text search for company names
- **sic** – Query by `standard industrial classification (SIC)`
- **filing_date** – Query by the date when the filing with financials data was filed. `datetime/date` or string `YYYY-MM-DD`
- **filing_date_lt** – filter for filing date less than given value
- **filing_date_lte** – filter for filing date less than equal to given value
- **filing_date_gt** – filter for filing date greater than given value
- **filing_date_gte** – filter for filing date greater than equal to given value
- **period_of_report_date** – query by The period of report for the filing with financials data. `datetime/date` or string in format: `YYY-MM-DD`.
- **period_of_report_date_lt** – filter for period of report date less than given value
- **period_of_report_date_lte** – filter for period of report date less than equal to given value
- **period_of_report_date_gt** – filter for period of report date greater than given value
- **period_of_report_date_gte** – filter for period of report date greater than equal to given value

- **time_frame** – Query by timeframe. Annual financials originate from 10-K filings, and quarterly financials originate from 10-Q filings. Note: Most companies do not file quarterly reports for Q4 and instead include those financials in their annual report, so some companies my not return quarterly financials for Q4. See *polygon.enums.StockFinancialsTimeframe* for choices.
- **include_sources** – Whether or not to include the xpath and formula attributes for each financial data point. See the xpath and formula response attributes for more info. `False` by default
- **order** – Order results based on the sort field. 'asc' by default. See *polygon.enums.SortOrder* for choices.
- **limit** – number of max results to obtain. defaults to 50.
- **sort** – Sort field key used for ordering. 'filing_date' default. see *polygon.enums.StockFinancialsSortKey* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.10 Get Stock Splits

`ReferenceClient.`**`get_stock_splits`**(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a list of historical stock splits for a ticker symbol, including the execution and payment dates of the stock split, and the split ratio. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.11 Get Market Holidays

`ReferenceClient.`**`get_market_holidays`**(*raw_response: bool = False*) → Union[requests.models.Response, dict]

Get upcoming market holidays and their open/close times. Official Docs

Parameters **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

4.12 Get Market Status

ReferenceClient.**get_market_status**(*raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current trading status of the exchanges and overall financial markets. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.13 Get Condition Mappings

ReferenceClient.**get_condition_mappings**(*tick_type='trades'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a unified numerical mapping for conditions on trades and quotes. Each feed/exchange uses its own set of codes to identify conditions, so the same condition may have a different code depending on the originator of the data. Polygon.io defines its own mapping to allow for uniformly identifying a condition across feeds/exchanges. Official Docs

Parameters

- **tick_type** – The type of ticks to return mappings for. Defaults to 'trades'. See *polygon.enums.ConditionMappingTickType* for choices.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.14 Get Conditions

ReferenceClient.**get_conditions**(*asset_class=None*, *data_type=None*, *id=None*, *sip=None*, *order=None*, *limit: int = 50*, *sort='name'*, *raw_response: bool = False*)

List all conditions that Polygon.io uses. Official Docs

Parameters

- **asset_class** – Filter for conditions within a given asset class. See *polygon.enums.AssetClass* for choices. Defaults to all assets.
- **data_type** – Filter by data type. See *polygon.enums.ConditionsDataType* for choices. defaults to all.
- **id** – Filter for conditions with a given ID
- **sip** – Filter by SIP. If the condition contains a mapping for that SIP, the condition will be returned.
- **order** – Order results. See *polygon.enums.SortOrder* for choices.
- **limit** – limit the number of results. defaults to 50.

- **sort** – Sort field used for ordering. Defaults to 'name'. See *polygon.enums.ConditionsSortKey* for choices.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.15 Get Exchanges

ReferenceClient.**get_exchanges**(*asset_class=None*, *locale=None*, *raw_response: bool = False*)
List all exchanges that Polygon.io knows about. Official Docs

Parameters

- **asset_class** – filter by asset class. See *polygon.enums.AssetClass* for choices.
- **locale** – Filter by locale name. See *polygon.enums.Locale*
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.16 Get Locales

ReferenceClient.**get_locales**(*raw_response: bool = False*) → Union[requests.models.Response, dict]
Get a list of locales currently supported by Polygon.io. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

4.17 Get Markets

ReferenceClient.**get_markets**(*raw_response: bool = False*) → Union[requests.models.Response, dict]
Get a list of markets that are currently supported by Polygon.io. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

CHAPTER

FIVE

FOREX

Read this page to know everything you need to know about using the various Forex HTTP endpoints.

See *Async Support for REST endpoints* for asynchronous use cases.

Docs below assume you have already read getting started page and know how to create the client. If you do not know how to create the client, first see *General guide for clients* and create client as below. As always you can have all 5 different clients together.

```
import polygon

forex_client = polygon.ForexClient('KEY')  # for usual sync client
async_forex_client = polygon.ForexClient('KEY', True)  # for an async client
```

Note that most endpoints require you to specify the currency pairs as separate symbols (a `from_symbol` and a `to_symbol`).

however a few endpoints require you to supply them as one combined symbol. An example would be the `get_aggregates_bars` method. In those methods, the symbol is expected to have a prefix `C:` before the currency symbol names. **but the library allows you to specify the symbol with or without the prefix**. See the relevant method's docs for more information on what the parameters expect.

5.1 Get Historic forex ticks

`ForexClient.get_historic_forex_ticks`(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historic trade ticks for a forex currency pair. Official Docs

Parameters

- **`from_symbol`** – The "from" symbol of the forex currency pair.
- **`to_symbol`** – The "to" symbol of the forex currency pair.
- **`date`** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **`offset`** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm thinking about a good way to implement this type of pagination in the lib which doesn't have a `next_url` in the response attributes.
- **`limit`** – Limit the size of the response, max 10000. Default 500

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.2 Get Last Quote

`ForexClient.get_last_quote`(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the last trade tick for a forex currency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the forex currency pair.
- **to_symbol** – The "to" symbol of the forex currency pair.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.3 Get Aggregate Bars (Candles)

`ForexClient.get_aggregate_bars`(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a forex pair over a given date range in custom time window sizes. For example, if `timespan = 'minute'` and `multiplier = '5'` then `5-minute` bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair. eg: `C:EURUSD`. You can supply with or without prefix `C:`
- **from_date** – The start of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. see *polygon.enums.SortOrder* for available choices. Defaults to `asc` which is oldest at the top.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.4 Get Grouped Daily Bars (Candles)

`ForexClient.get_grouped_daily_bars`(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire forex markets. Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.5 Get Previous Close

`ForexClient.get_previous_close`(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified forex pair. Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.6 Get Gainers & Losers

ForexClient.**get_gainers_and_losers**(*direction='gainers'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current top 20 gainers or losers of the day in forex markets. Official docs

Parameters

- **direction** – The direction of the snapshot results to return. See *polygon.enums.SnapshotDirection* for available choices. Defaults to Gainers.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

5.7 Real Time currency conversion

ForexClient.**real_time_currency_conversion**(*from_symbol: str*, *to_symbol: str*, *amount: float*, *precision: int = 2*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get currency conversions using the latest market conversion rates. Note than you can convert in both directions. For example USD to CAD or CAD to USD. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **amount** – The amount to convert,
- **precision** – The decimal precision of the conversion. Defaults to 2 which is 2 decimal places accuracy.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

CHAPTER

SIX

CRYPTO

Read this page to know everything you need to know about using the various Crypto HTTP endpoints.

See *Async Support for REST endpoints* for asynchronous use cases.

Docs below assume you have already read getting started page and know how to create the client. If you do not know how to create the client, first see *General guide for clients* and create client as below. As always you can have all 5 different clients together.

```
import polygon

crypto_client = polygon.CryptoClient('KEY')  # for usual sync client
async_crypto_client = polygon.CryptoClient('KEY', True)  # for an async client
```

Note that most endpoints require you to specify the currency pairs as separate symbols (a `from_symbol` and a `to_symbol`).

however a few endpoints require you to supply them as one combined symbol. An example would be the `get_aggregates_bars` method. In those methods, the symbol is expected to have a prefix `X:` before the currency symbol names. **but the library allows you to specify the symbol with or without the prefix**. See the relevant method's docs for more information on what the parameters expect.

6.1 Get Historic Trades

CryptoClient.**get_historic_trades**(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historic trade ticks for a cryptocurrency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the crypto pair.
- **to_symbol** – The "to" symbol of the crypto pair.
- **date** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **offset** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm trying to think of a good way to implement pagination in the library for these endpoints which do not return a `next_url` attribute.
- **limit** – Limit the size of the response, max 10000. Default 500

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.2 Get Last Trade

`CryptoClient.get_last_trade`(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the last trade tick for a cryptocurrency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.3 Get Daily Open Close

`CryptoClient.get_daily_open_close`(*from_symbol: str*, *to_symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the open, close prices of a cryptocurrency symbol on a certain day. Official Docs:

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **date** – The date of the requested open/close. Could be `datetime`, `date` or string `YYYY-MM-DD`.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.4 Get Aggregate Bars (Candles)

CryptoClient.**get_aggregate_bars**(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a cryptocurrency pair over a given date range in custom time window sizes. For example, if `timespan='minute'` and `multiplier='5'` then 5-minute bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without prefix `X:`
- **from_date** – The start of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Order of sorting the results. See *polygon.enums.SortOrder* for available choices. Defaults to `asc` (oldest at the top)
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.5 Get Grouped Daily Bars (Candles)

CryptoClient.**get_grouped_daily_bars**(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire cryptocurrency market. Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.6 Get Previous Close

CryptoClient.**get_previous_close**(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified cryptocurrency pair. Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without the prefix `X:`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.7 Get Snapshot All

CryptoClient.**get_snapshot_all**(*symbols: list*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded cryptocurrency symbols Official Docs

Parameters

- **symbols** – A list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.8 Get Snapshot

CryptoClient.**get_snapshot**(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded cryptocurrency symbol. Official Docs

Parameters

- **symbol** – Symbol of the currency pair. eg: `X:BTCUSD`. you can specify with or without prefix `X:`
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

6.9 Get Level 2 Book

`CryptoClient.get_level2_book(`*symbol: str*, *raw_response: bool = False*`)` → Union[requests.models.Response, dict]

Get the current level 2 book of a single ticker. This is the combined book from all of the exchanges. Official Docs

Parameters

- **`symbol`** – The cryptocurrency ticker. eg: `X:BTCUSD`. You can specify with or without the prefix `` `X: ``
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

CHAPTER

SEVEN

CALLBACK STREAMING

A convenient wrapper around the Streaming API

IMPORTANT Polygon.io allows one simultaneous connection to one cluster at a time (clusters: stocks, options, forex, crypto). which means 4 total concurrent streams (Of course you need to have subscriptions for them).

Connecting to a cluster which already has an existing stream connected to it would result in existing connection getting dropped and new connection would be established

Note that This page describes the callback based streaming client. If you're looking for async based streaming client, See *Async Streaming*

Also note that callback based streamer is supposed to get a builtin functionality to reconnect in the library. Async streamer has it already. It's on TODO for this client. Have a reconnect mechanism to share? Share in discussions or on the wiki.

7.1 Creating the client

Creating a client is just creating an instance of `polygon.StreamClient`. Note that this expects a few arguments where most of them have default values.

This is how the initializer looks like:

`StreamClient.__init__`(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *on_message=None*, *on_close=None*, *on_error=None*, *enable_connection_logs: bool = False*)

Initializes the callback function based stream client Official Docs

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **cluster** – Which market/cluster to connect to. See *polygon.enums.StreamCluster* for choices. NEVER connect to the same cluster again if there is an existing stream connected to it. The existing connection would be dropped and new one will be established. You can have up to 4 concurrent streams connected to 4 different clusters.
- **host** – Host url to connect to. Default is real time. See *polygon.enums.StreamHost* for choices.
- **on_message** – The function to be called when data is received. This is primary function you'll write to process the data from the stream. The function should accept one and only one `arg` (message). Default handler is *_default_on_msg()*.
- **on_close** – The function to be called when stream is closed. Function should accept two args (close_status_code, close_message). Default handler is *_default_on_close()*

- **on_error** – Function to be called when an error is encountered. Function should accept one arg (exception object). Default handler is *_default_on_error()*
- **enable_connection_logs** – Whether or not to print debug info related to the stream connection. Helpful for debugging.

Example use:

```
import polygon

stream_client = polygon.StreamClient('KEY', 'stocks', on_message=my_own_handler_
↪function)  # in the simplest form
```

Note that you don't have to call login methods as the library does it internally itself.

7.2 Starting the Stream

Once you have a stream client, you can start the stream thread by calling the method: `start_stream_thread`.

This method has default values which should be good enough for most people. For those who need customization, here is how it looks like:

`StreamClient.`**start_stream_thread**(*ping_interval: int = 21*, *ping_timeout: int = 20*, *ping_payload: str = ''*, *skip_utf8_validation: bool = True*)

Starts the Stream. This will not block the main thread and it spawns the streamer in its own thread.

Parameters

- **ping_interval** – client would send a `ping` every specified number of seconds to server to keep connection alive. Set to 0 to disable pinging. Defaults to 21 seconds
- **ping_timeout** – Timeout in seconds if a `pong` (response to ping from server) is not received. The Stream is terminated as it is considered to be dead if no pong is received within the specified timeout. default: 20 seconds
- **ping_payload** – The option message to be sent with the ping. Better to leave it empty string.
- **skip_utf8_validation** – Whether to skip utf validation of messages. Defaults to True. Setting it to False may result in performance downgrade

Returns None

Example use:

```
import polygon

stream_client = polygon.StreamClient('KEY', 'stocks', on_message=my_own_handler_function)

stream_client.start_stream_thread()

# subscriptions here.
```

7.3 Important Concepts

Important stuff to know before you connect your first stream. Note that when writing applications, you should create the client and start the stream thread before subscribing.

7.3.1 Subscribing/Unsubscribing to Streams

All subscription methods have names in pattern `subscribe_service_name` and `unsubscribe_service_name`.

Symbols names must be specified as a list of symbols: `['AMD', 'NVDA', 'LOL']` is the correct way to specify symbols. Not specifying a list of symbols results in the action being applied to `ALL` tickers in that service. Note that either of `[]`, `None`, `['*']` or `'all'` as value of symbols would also results in ALL tickers.

The library allows specifying a string as for symbol argument, but only do that if you have the absolute need to. Most people should just specify a list. Note that a list of single ticker is accepted.

7.3.2 Handling messages

Your handler function should accept two arguments. You can ignore the first argument which is going to be the websocket instance itself. The second argument is the actual message.

```
def sample_handler(ws, msg):
    print(msg)
```

Once you have the message in your callback handler function, you can process it the way you want. print it out, write it to a file, push it to a redis queue, write to a database, offload to a multi-threaded queue. Just whatever.

The default handler for the messages is `_default_on_msg` which does some checks on messages having event as `status`. and prints out other messages. Messages from polygon having the key `ev` equal to `status` are status updates from polygon about login and relevant actions you take (ev indicates event)

The data messages will have different `ev` value than the string 'status'. The ev values for those would match the *polygon.enums.StreamServicePrefix* values.

You can specify your own handlers for other callbacks (`on_error`, `on_close` etc) too or leave those to defaults.

if you choose to override default handlers for `on_error` and `on_close`, here is how they need to be written

`on_error` handler must accept two arguments. You can ignore the first argument which is just the websocket instance itself. The second argument is going to be the actual error

```
def sample_error_handler(ws, error):
    print(error)
```

`on_close` handler must accept three arguments. you can ignore the first arg which is just the websocket instance itself. The second arg is close code, and third would be the close message. note that this handler is only called when the stream is being closed.

```
def sample_close_handler(ws, close_code, close_msg):
    print(f'Stream close with code: {close_code} || msg: {close_msg}')
```

7.3.3 Closing Stream

To turn off the streamer and shut down the websockets connection gracefully, it is advised to call `stream_client.close_stream()` method when closing the application. Not an absolute necessity but a good software practice.

7.4 Stocks Streams

7.4.1 Stock Trades

`StreamClient.`**`subscribe_stock_trades`**(*symbols: Optional[list] = None*)
Stream real-time trades for given stock ticker symbol(s).

Parameters **`symbols`** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

`StreamClient.`**`unsubscribe_stock_trades`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.4.2 Stock Quotes

`StreamClient.`**`subscribe_stock_quotes`**(*symbols: Optional[list] = None*)
Stream real-time Quotes for given stock ticker symbol(s).

Parameters **`symbols`** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

`StreamClient.`**`unsubscribe_stock_quotes`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.4.3 Stock Minute Aggregates (OCHLV)

`StreamClient.`**`subscribe_stock_minute_aggregates`**(*symbols: Optional[list] = None*)
Stream real-time minute aggregates for given stock ticker symbol(s).

Parameters **`symbols`** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

`StreamClient.`**`unsubscribe_stock_minute_aggregates`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.4.4 Stock Second Aggregates (OCHLV)

`StreamClient.`**`subscribe_stock_second_aggregates`**(*symbols: Optional[list] = None*)
Stream real-time second aggregates for given stock ticker symbol(s).

Parameters **`symbols`** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

`StreamClient.`**`unsubscribe_stock_second_aggregates`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.4.5 Stock Limit Up Limit Down (LULD)

StreamClient.**subscribe_stock_limit_up_limit_down**(*symbols: Optional[list] = None*)
Stream real-time LULD events for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

StreamClient.**unsubscribe_stock_limit_up_limit_down**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.4.6 Stock Imbalances

StreamClient.**subscribe_stock_imbalances**(*symbols: Optional[list] = None*)
Stream real-time Imbalance Events for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

StreamClient.**unsubscribe_stock_imbalances**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

7.5 Options Streams

7.5.1 Options Trades

StreamClient.**subscribe_option_trades**(*symbols: Optional[list] = None*)
Stream real-time Options Trades for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

StreamClient.**unsubscribe_option_trades**(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Trades for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

7.5.2 Options Minute Aggregates (OCHLV)

StreamClient.**subscribe_option_minute_aggregates**(*symbols: Optional[list] = None*)
Stream real-time Options Minute Aggregates for given Options contract(s).

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL tickers in the market. you can pass **with or without** the prefix `O:`

Returns None

StreamClient.**unsubscribe_option_minute_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Minute aggregates for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

7.5.3 Options Second Aggregates (OCHLV)

`StreamClient.`**`subscribe_option_second_aggregates`**(*symbols: Optional[list] = None*)
Stream real-time Options Second Aggregates for given Options contract(s).

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL tickers in the market. you can pass **with or without** the prefix `O:`

Returns None

`StreamClient.`**`unsubscribe_option_second_aggregates`**(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Second Aggregates for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

7.6 Forex Streams

7.6.1 Forex Quotes

`StreamClient.`**`subscribe_forex_quotes`**(*symbols: Optional[list] = None*)
Stream real-time forex quotes for given forex pair(s).

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

`StreamClient.`**`unsubscribe_forex_quotes`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

7.6.2 Forex Minute Aggregates (OCHLV)

`StreamClient.`**`subscribe_forex_minute_aggregates`**(*symbols: Optional[list] = None*)
Stream real-time forex Minute Aggregates for given forex pair(s).

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

`StreamClient.`**`unsubscribe_forex_minute_aggregates`**(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

7.7 Crypto Streams

7.7.1 Crypto Trades

`StreamClient.`**`subscribe_crypto_trades`**(*symbols: Optional[list] = None*)
Stream real-time Trades for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

`StreamClient.`**`unsubscribe_crypto_trades`**(*symbols: Optional[list] = None*)
Unsubscribe real-time trades for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

7.7.2 Crypto Quotes

`StreamClient.`**`subscribe_crypto_quotes`**(*symbols: Optional[list] = None*)
Stream real-time Quotes for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

`StreamClient.`**`unsubscribe_crypto_quotes`**(*symbols: Optional[list] = None*)
Unsubscribe real-time quotes for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

7.7.3 Crypto Minute Aggregates (OCHLV)

StreamClient.**subscribe_crypto_minute_aggregates**(*symbols: Optional[list] = None*)
Stream real-time Minute Aggregates for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

StreamClient.**unsubscribe_crypto_minute_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe real-time minute aggregates for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

7.7.4 Crypto Level 2 Book

StreamClient.**subscribe_crypto_level2_book**(*symbols: Optional[list] = None*)
Stream real-time level 2 book data for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

StreamClient.**unsubscribe_crypto_level2_book**(*symbols: Optional[list] = None*)
Unsubscribe real-time level 2 book data for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

CHAPTER

EIGHT

ASYNC STREAMING

A convenient wrapper around the Streaming API

IMPORTANT Polygon.io allows one simultaneous connection to one cluster at a time (clusters: stocks, options, forex, crypto). which means 4 total concurrent streams (Of course you need to have subscriptions for them).

Connecting to a cluster which already has an existing stream connected to it would result in existing connection getting dropped and new connection would be established

Note that This page describes the asyncio based streaming client. If you're looking for callback based streaming client, See *Callback Streaming*

Also note that async client has a reconnection mechanism built into it already. It is very basic at the moment. It resubscribes to the same set of services it already had before the disconnection and restores the handlers when reconnection establishes. More info in starting the stream below.

It also exposes a few methods which you could use to create your own reconnect mechanism. Method *polygon.streaming.async_streaming.AsyncStreamClient.reconnect()* is one of them

Have a reconnect mechanism to share? Share in discussions or on the wiki.

8.1 Creating the client

Creating a client is just creating an instance of `polygon.AsyncStreamClient`. Note that this expects a few arguments where most of them have default values.

This is how the initializer looks like:

`AsyncStreamClient.__init__`(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *ping_interval: int = 20*, *ping_timeout: bool = 19*, *max_message_size: int = 1048576*, *max_memory_queue: int = 32*, *read_limit: int = 65536*, *write_limit: int = 65536*)

Initializes the stream client for async streaming Official Docs

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **cluster** – Which market/cluster to connect to. See *polygon.enums.StreamCluster* for choices. NEVER connect to the same cluster again if there is an existing stream connected to it. The existing connection would be dropped and new one will be established. You can have up to 4 concurrent streams connected to 4 different clusters.
- **host** – Host url to connect to. Default is real time. See *polygon.enums.StreamHost* for choices
- **ping_interval** – Send a `ping` to server every specified number of seconds to keep the connection alive. Defaults to 20 seconds. Setting to 0 disables pinging.

- **ping_timeout** – The number of seconds to wait after sending a ping for the response (`pong`). If no response is received from the server in those many seconds, stream is considered dead and exits with code `1011`. Defaults to 19 seconds.
- **max_message_size** – The max_size parameter enforces the maximum size for incoming messages in bytes. The default value is `1 MiB` (not MB). None disables the limit. If a message larger than the maximum size is received, `recv()` will raise `ConnectionClosedError` and the connection will be closed with code `1009`
- **max_memory_queue** – sets the maximum length of the queue that holds incoming messages. The default value is `32`. None disables the limit. Messages are added to an in-memory queue when they're received; then `recv()` pops from that queue
- **read_limit** – sets the high-water limit of the buffer for incoming bytes. The low-water limit is half the high-water limit. The default value is `64 KiB`, half of asyncio's default. Don't change if you are unsure of what it implies.
- **write_limit** – The write_limit argument sets the high-water limit of the buffer for outgoing bytes. The low-water limit is a quarter of the high-water limit. The default value is `64 KiB`, equal to asyncio's default. Don't change if you're unsure what it implies.

Example use:

```
import polygon

stream_client = polygon.AsyncStreamClient('KEY', 'stocks')  # in the simplest form
```

Note that you don't have to call login methods as the library does it internally itself.

8.2 Starting the Stream

Once you have a stream client, you MUST subscribe to streams before you start the main stream loop. Note that you can alter your subscriptions from other coroutines easily even after starting the main stream loop. See subscriptions methods below this section to know how to subscribe to streams.

AFTER you have called your initial subscription methods, you have two ways to start the main stream loop.

8.2.1 Without using the built-in reconnect functionality

In this case you'd need to have your own while loop, like so:

```
# assuming we create the client and sub to stream here already.
while 1:
    await stream_client.handle_messages()
```

and that's basically it. handle_message would take care of receiving messages and calling appropriate handlers (see below section for info on that aspect). You may want to implement your own reconnect mechanism here.

If that's your use case, you can basically ignore the below section completely.

8.2.2 Using the built-in reconnect functionality

here you don't need any outer while loop of your own. The lib has inner while loops and mechanisms to trap disconnection errors and will attempt to reconnect.

Note that this function is basic and not perfect yet and will continue to improve as we move ahead. If you figure out a way to implement reconnection, feel free to share that in discussions or on the wiki.

simple use example

```
# assuming we already have a client subscribed to streams
await stream_client.handle_messages(reconnect=True)
```

That's it. This should be enough for most users. For those who need more control over the behavior here; this is how the method definition looks like:

async AsyncStreamClient.**handle_messages**(*reconnect: bool = False*, *max_reconnection_attempts=5*, *reconnection_delay=5*)

The primary method to start the stream. Connects & Logs in by itself. Allows Reconnecting by simply altering a parameter (subscriptions are persisted across reconnected streams)

Parameters

- **reconnect** – If this is `False` (default), it simply awaits the next message and calls the appropriate handler. Uses the *_default_process_message()* if no handler was specified. You should use the statement inside a while loop in that case. Setting it to True creates an inner loop which traps disconnection errors except login failed due to invalid Key, and reconnects to the stream with the same subscriptions it had earlier before getting disconnected.
- **max_reconnection_attempts** – Determines how many times should the program attempt to reconnect in case of failed attempts. The Counter is reset as soon as a successful connection is re-established. Setting it to False disables the limit which is NOT recommended unless you know you got a situation. This value is ignored if `reconnect` is False (The default). Defaults to 5.
- **reconnection_delay** – Number of seconds to wait before attempting to reconnect after a failed reconnection attempt or a disconnection. This value is ignored if `reconnect` is False (the default). Defaults to 5.

Returns None

8.3 Subscribing/Unsubscribing to Streams

All subscription methods have names in pattern `subscribe_service_name` and `unsubscribe_service_name`.

Symbols names must be specified as a list of symbols: `['AMD', 'NVDA', 'LOL']` is the correct way to specify symbols. Not specifying a list of symbols results in the action being applied to `ALL` tickers in that service. Note that either of `[]`, `None`, `['*']` or `'all'` as value of symbols would also results in ALL tickers.

The library allows specifying a string as for symbol argument, but only do that if you have the absolute need to. Most people should just specify a list. Note that a list of single ticker is accepted.

The Second argument on all unsubscribe methods is the `handler_function` which represents the handler function you'd like the library to call when a message from that service is received. You can have one handler for multiple services. Not supplying a handler results in the library using the default message handler.

All methods are async coroutines which need to be awaited.

```
await stream_client.subscribe_stock_trades(['AMD', 'NVDA'], handler_function=my_handler_function)
```

8.4 Handling Messages

your handler functions should accept one argument which indicates the message.

```
async def sample_handler(msg):
    print(f'Look at me! I am the handler now. {msg}')
```

Note that you can also use a sync function as handler

```
def sample_handler(msg):
    print(f'I am also a handler. But sync.. {msg}')
```

Once you have the message in your callback handler function, you can process it the way you want. print it out, write it to a file, push it to a redis queue, write to a database, offload to a multi-threaded queue. Just whatever.

The default handler for the messages is `_default_process_message`.

8.5 Changing message handler functions while stream is running

Library allows you to change your handlers after your main stream loop has started running.

The function you'd need is:

async AsyncStreamClient.**change_handler**(*service_prefix*, *handler_function*)

Change your handler function for a service. Can be used to update handlers dynamically while stream is running.

Parameters

- **service_prefix** – The Prefix of the service you want to change handler for. see *polygon.enums.StreamServicePrefix* for choices.
- **handler_function** – The new handler function to assign for this service

Returns None

Note that you should never need to change handler for `status` (which handles `ev` messages) unless you know you got a situation. Service prefixes just indicate which service (eg stock trades? options aggregates?) you want to change the handler.

8.6 Closing the Stream

To turn off the streamer and shut down the websockets connection gracefully, it is advised to `await stream_client.close_stream()` when closing the application. Not an absolute necessity but a good software practice.

8.7 Stock Streams

8.7.1 Stock Trades

async AsyncStreamClient.**subscribe_stock_trades**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time trades for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL tickers.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_trades**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.7.2 Stock Quotes

async AsyncStreamClient.**subscribe_stock_quotes**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time quotes for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL tickers.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_quotes**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.7.3 Stock Minute Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_stock_minute_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_minute_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.7.4 Stock Second Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_stock_second_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Seconds Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_second_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.7.5 Stock Limit Up Limit Down (LULD)

async AsyncStreamClient.**subscribe_stock_limit_up_limit_down**(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time LULD Events for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_limit_up_limit_down**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.7.6 Stock Imbalances

async AsyncStreamClient.**subscribe_stock_imbalances**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Imbalance Events for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_stock_imbalances**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

8.8 Options Streams

8.8.1 Options Trades

async AsyncStreamClient.**subscribe_option_trades**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time options trades for provided ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_option_trades**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

8.8.2 Options Minute Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_option_minute_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time options minute aggregates for given ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`

- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_option_minute_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

8.8.3 Options Second Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_option_second_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time options second aggregates for given ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_option_second_aggregates**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

8.9 Forex Streams

8.9.1 Forex Quotes

async AsyncStreamClient.**subscribe_forex_quotes**(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Forex Quotes for provided symbol(s)

Parameters

- **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_forex_quotes**(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied forex symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

8.9.2 Forex Minute Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_forex_minute_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Forex Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_forex_minute_aggregates**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied forex symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

8.10 Crypto Streams

8.10.1 Crypto Trades

async AsyncStreamClient.**subscribe_crypto_trades**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Crypto Trades for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_crypto_trades**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

8.10.2 Crypto Quotes

async AsyncStreamClient.**subscribe_crypto_quotes**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Crypto Quotes for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_crypto_quotes**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

8.10.3 Crypto Minute Aggregates (OCHLV)

async AsyncStreamClient.**subscribe_crypto_minute_aggregates**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Crypto Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_crypto_minute_aggregates**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

8.10.4 Crypto Level 2 Book

async AsyncStreamClient.**subscribe_crypto_level2_book**(*symbols: Optional[list] = None*, *handler_function=None*)

Get Real time Crypto Level 2 Book Data for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async AsyncStreamClient.**unsubscribe_crypto_level2_book**(*symbols: Optional[list] = None*)

Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

CHAPTER

NINE

WHAT THE HELL ARE ENUMS ANYWAYS

Sooooo… you've had enough of these `enums` and finally decided to know what the hell they actually are and why you should care about them.

Well read this page to get your answers.

You should have seen them on many methods' documentation as argument choices.

First up, does everyone need them? that depends on their use case. enums in this library are only used on some endpoints, especially the ones in reference APIs and some basic uses in stream clients. So if someone only needs to ochlv chart data, they probably won't need to use enums.

If you notice any value which is supported by the API but not included in the enums, Let me Know using discussions

9.1 What are they

Simplest non technical terms definition They are a way to define pseudo constants (read constants) in python (python doesn't have anything as constants. That's why enums are precious :D). They have many use cases other than constants but for this library you only need to know this far.

For example consider the enum *polygon.enums.AssetClass* which has 4 values inside of it. The values are just class attribute and you can access them just like you'd access any other class attribute. `print(polygon.enums.AssetClass.STOCKS)` would print the string `stocks`. so in another words this enum class has 4 member enums which can be used to specify the value wherever needed. Like this `some_function(arg1, asset=AssetClass.STOCKS)`.

when you pass in an enum to a function or a method, it is equal to passing in the value of that enum.

so instead of `some_function(arg1, asset=AssetClass.STOCKS)` i could have said `some_function(arg1, asset='stocks')` and both mean the same thing.

Here are All the enums of this library in one place

9.2 Then why not just pass in raw values? Why do we need enums?

I mean you could do that. In fact many people would still do that despite the notes here (I'll be watching you all :/).

but think about it this way, can you have enums for a parameter which expects a person's name? Of course not. Because there isn't any constant value (or a fixed set of values) to choose from.

but can i have enums for TickerTypes? Yes. Because it has a set of fixed values and the API would not return the correct data if the value passed in is different than the ones which are in the fixed set.

Using enums

- Avoids passing in incorrect values.
- Avoids typing mistakes while passing in parameter values (I'm looking at you `TRAILING_TWELVE_MONTHS_ANNUALIZED`)
- gives you a fixed set of values to choose from and you don't have to hit and trial to know supported values.
- And finally, IDE autocomplete would make your life even easier while writing code that makes use of enums

Finally, it's not an absolute necessity to use enums but they are very much recommended.

9.3 Okay how do I use them

To start off, like any other name, you'd need to import the names. Now there are many ways to do that and it's up to your coding preferences. Make use of your IDE auto-completions to make it easier to fill in enums.

Some common ways are

9.3.1 Approach 1 - importing all enums at once

```
import polygon  # which you already do for using other clients so nothing new to import␣
↪here

# now you can use enums as

client.some_function(other_args, arg=polygon.enums.TickerType.ADRC)

# OR
import polygon.enums as enums

client.some_function(other_args, arg=enums.TickerType.ETF)
```

as you see this allows you to access all enums without having to import each one individually. But this also mean you'd be typing longer names (not big of an issue considering IDE completions).

Note that importing all enums doesn't have any resource overhead so don't worry about enums eating your RAM.

9.3.2 Approach 2 - importing just the enums you need

This approach is nicer for cases when you only specifically need a few enums.

```
from polygon.enums import TickerType

# using it as
client.some_function(other_args, arg=TickerType.CS)

# OR
from polygon.enums import (TickerType, AssetClass)

client.some_function(other_args, arg=TickerType.CS)

client.some_other_function(other_args, arg=TickerType.CS, other_arg=AssetClass.STOCKS)
```

9.3.3 Other Approaches

You could use any other import syntax if you like. such as `from polygon.enums import *` but I wouldn't recommend wild card imports.

CHAPTER

TEN

GETTING HELP

I see you're stuck at something. don't worry, everyone does. Need a hand? Here is how you can get help.

- See if you can find the relevant info in *FAQs* or *Community Wikis*
- See if there is an Open Issue or a Pull Request related to your concern already.
- See if your issue has been discussed already in one of the *Discussions*
- If you believe the issue could be on polygon.io end, get in touch with their support team. They're quite helpful. There is a button in bottom right corner of every documentation page

Once you have gone through these and haven't found your answer, you can

- Start a Discussion. You can ask your questions in general channel or create a QnA discussion from left.

If your question is more of a bug report, you can raise a new issue or feature request with adequate information.

Remember that Issues is not a good place to ask for general help.

Always make sure to provide enough information when asking for help. This includes

- Your Operating system (Linux? Windows?)
- Your execution environment (Pycharm? VSC? A usual terminal? a cloud instance? a rasp pi?)
- Your python version and `polygon` version. always ensure you are on the latest version of the library. You can update if you're not using command `pip install --upgrade polygon`
- The full stack traceback and error message if any. Do not attempt to describe error messages in your own languages. Sometimes messages don't mean what they say
- The code which causes the error. If your code is supposed to be secret, write a sample script which can reproduce the issue. Always make sure to remove sensitive info from logs/code

CHAPTER

ELEVEN

BUGS, DISCUSSIONS, WIKIS, FAQS

This section provides info on Issues tracker, Discussions functionality, community wikis and FAQs.

11.1 Bug Reports or Feature Requests

Got a bug/report to report or a feature request? You're in the right place.

Before submitting, make sure you have enough information to provide. It is advised to follow the provided template but feel free to use your own. Just ensure you provide the following info:

- Your Operating system (Linux? Windows?)
- Your execution environment (Pycharm? VSC? A usual terminal? a cloud instance? a rasp pi?)
- Your python version and `polygon` version. always ensure you are on the latest version of the library. You can update if you're not using command `pip install --upgrade polygon`
- The full stack traceback and error message if any. Do not attempt to describe error messages in your own languages. Sometimes messages don't mean what they say
- The code which causes the error. If your code is supposed to be secret, write a sample script which can reproduce the issue. Always make sure to remove sensitive info from logs/code

In case of feature requests, describe what functionality would you like to be added to the library.

Open issues/feature requests here

11.2 Discussions

Discussions are meant to be a place for discussing general stuff which is not worth having an open issue for.

there are two discussion channels by default, one meant for everyone and other meant for contributors/developers

while it is possible to create your own discussions, it is preferred to keep it to those two channels unless needed.

11.3 Community Wikis

The community wiki is a place for everything which the community finds useful for others but isn't in the documentation. every article is just a title and the description text. written in good old markdown. You can write plain text too if you're unsure of what markdown is.

Figured out how to achieve a specific task? Found something interesting? share it with the community by creating a wiki page. Every contribution is significant so don't hesitate.

Read the wiki articles, you may find your answers there.

11.4 FAQs

This is a handpicked collection of common questions and answers about the lib and endpoints in general. A must read if you're looking for answers.

FAQs are added here as soon I have any solid conclusions about a useful question.

CHAPTER

TWELVE

CONTRIBUTING AND LICENSE

12.1 Contributing to the library

A bug you can fix? Improving documentation? Just wanna structure the code better? Every improvement matters.

Read this small guide to know how you can start contributing.

If this is your first time contributing to an open source project, Welcome. You'd probably want to contribute to something you are confident about

Want to discuss anything related to the lib? head over to Developer Discussions. You may also use discussions to ask anything related to contributions or library in general.

12.1.1 Picking up what to work on

If you already know what you're going to work on, Great! If you don't or just wanna explore the options; below are the places to look at:

1. Take a look at open issues and see which ones you can work on.
2. Anything which could be improved in the documentation or readme ?
3. Any new endpoints introduced by polygon.io which are not in the library?
4. Any changes to endpoints which are already in the lib but not adjusted according to the new changes?

Once you know what to work on, you can proceed with setting up your environment.

12.1.2 Setting Up the Development Environment

May not be needed for documentation improvements.

Dependencies are listed in requirements.txt. The list has `sphinx` and `sphinx_rtd_theme` which are only meant to build documentation.

It is highly recommended to install the dependencies in a virtual environment to avoid messing with your global interpreter.

```
pip install virtualenv
virtualenv venv
. venv/bin/activate
```

The last instruction above is for *nix machines. For windows `.\venv\Scripts\activate.bat` (or similar) is used

Install the requirements using

```
pip install -r requirements.txt
```

Now you can make your changes

12.1.3 Testing your changes

Currently the project uses the actual endpoints to perform tests (Suggestions/PRs for better testing mechanism are welcome)

All test files are under directory `tests`. You'd need a valid polygon API key to perform the tests as they are right now. If you don't have a subscription, just make the changes, test them the way you like and raise the PR. I'll test the changes before merging.

However if you made changes to the documentation, run the below commands to build locally and test the documentation

```
cd docs
make html
```

The built docs would be placed under `docs/_build/_html`. Open `index.html` here in a browser and see your changes. When you're happy with them, raise the PR.

Remember to document your changes like this library does already.

12.2 License

Don't kid yourself. You don't care what license does the project use, do you? Anyways the project is licensed under MIT License. See License for more details.

CHAPTER

THIRTEEN

LIBRARY INTERFACE DOCUMENTATION

Here is the Entire Library Interface reference.

13.1 Base Client

class polygon.base_client.**BaseClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This is the **base client class** for all other REST clients which inherit from this class and implement their own endpoints on top of it.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

close()

Closes the `requests.Session` and frees up resources. It is recommended to call this method in your exit handlers Note that this is meant for sync programming only. Use *async_close()* for async.

async async_close()
Closes the httpx.AsyncClient and frees up resources. It is recommended to call this method in your exit handlers. This method should be awaited as this is a coroutine. Note that this is meant for async programming only. Use close() for sync.

_get_response(*path: str*, *params: Optional[dict] = None*, *raw_response: bool = True*) → Union[requests.models.Response, dict]
Get response on a path. Meant to be used internally but can be used if you know what you're doing. To be used by sync client only. For async access, see _get_async_response()

Parameters

- **path** – RESTful path for the endpoint. Available on the docs for the endpoint right above its name.
- **params** – Query Parameters to be supplied with the request. These are mapped 1:1 with the endpoint.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to check the status code or inspect the headers. Defaults to True which returns the Response object.

Returns A Response object by default. Make raw_response=False to get JSON decoded Dictionary

async _get_async_response(*path: str*, *params: Optional[dict] = None*, *raw_response: bool = True*) → Union[httpx.Response, dict]
Get response on a path - meant to be used internally but can be used if you know what you're doing - to be used by async client only. For sync access, see _get_response()

Parameters

- **path** – RESTful path for the endpoint. Available on the docs for the endpoint right above its name.
- **params** – Query Parameters to be supplied with the request. These are mapped 1:1 with the endpoint.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to check the status code or inspect the headers. Defaults to True which returns the Response object.

Returns A Response object by default. Make raw_response=False to get JSON decoded Dictionary

get_next_page_by_url(*url: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the next page of a response. The URl is returned within next_url attribute on endpoints which support pagination (eg the tickers endpoint). If the response doesn't contain this attribute, either all pages were received or the endpoint doesn't have pagination. Meant for internal use primarily.

Note that this method is meant for sync programming. See async_get_next_page_by_url() for async.

Parameters

- **url** – The next URL. As contained in next_url of the response.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of raw_response. Defaults to Dict.

async async_get_next_page_by_url(*url: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the next page of a response. The URl is returned within `next_url` attribute on endpoints which support pagination (eg the tickers endpoint). If the response doesn't contain this attribute, either all pages were received or the endpoint doesn't have pagination. Meant for internal use primarily.

Note that this method is meant for async programming. See *get_next_page_by_url()* for sync.

Parameters

- **url** – The next URL. As contained in `next_url` of the response.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of raw_response. Defaults to Dict.

get_next_page(*old_response: Union[requests.models.Response, dict]*, *raw_response: bool = False*) → Union[requests.models.Response, dict, bool]

Get the next page using the most recent old response. This function simply parses the next_url attribute from the existing response and uses it to get the next page. Returns False if there is no next page remaining (which implies that you have reached the end of all pages or the endpoint doesn't support pagination).

Parameters

- **old_response** – The most recent existing response. Can be either `Response` Object or Dictionaries
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_next_page(*old_response: Union[httpx.Response, dict]*, *raw_response: bool = False*) → Union[httpx.Response, dict, bool]

Get the next page using the most recent old response. This function simply parses the next_url attribute from the existing response and uses it to get the next page. Returns False if there is no next page remaining (which implies that you have reached the end of all pages or the endpoint doesn't support pagination) - Async method

Parameters

- **old_response** – The most recent existing response. Can be either `Response` Object or Dictionaries
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_previous_page(*old_response: Union[requests.models.Response, dict]*, *raw_response: bool = False*) → Union[requests.models.Response, dict, bool]

Get the previous page using the most recent old response. This function simply parses the previous_url attribute from the existing response and uses it to get the previous page. Returns False if there is no previous page remaining (which implies that you have reached the start of all pages or the endpoint doesn't support pagination).

Parameters

- **old_response** – The most recent existing response. Can be either Response Object or Dictionaries
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_previous_page(*old_response: Union[httpx.Response, dict]*, *raw_response: bool = False*) → Union[httpx.Response, dict, bool]

Get the previous page using the most recent old response. This function simply parses the previous_url attribute from the existing response and uses it to get the previous page. Returns False if there is no previous page remaining (which implies that you have reached the start of all pages or the endpoint doesn't support pagination) - Async method

Parameters

- **old_response** – The most recent existing response. Can be either Response Object or Dictionaries
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

13.2 Stocks Client

class polygon.stocks.stocks.**StocksClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This class implements all the Stocks REST endpoints. Note that you should always import names from top level. eg: `from polygon import StocksClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = StocksClient('MY_API_KEY')` Once you have the client, you can call its methods to get data from the APIs. All methods have sane default values and almost everything can be customized.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

get_trades(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get trades for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values. Official Docs

Parameters

- **symbol** – The ticker symbol we want trades for.
- **date** – The date/day of the trades to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **timestamp** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the next page of results. Default: None. I'm trying to think of a good way to implement pagination support for this type of pagination.
- **timestamp_limit** – The maximum timestamp allowed in the results. Default: None
- **reverse** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **limit** – Limit the size of the response, max 50000 and default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_quotes(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get Quotes for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values. Official Docs

Parameters

- **symbol** – The ticker symbol we want quotes for.
- **date** – The date/day of the quotes to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **timestamp** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the

next page of results. Default: None. Thinking of a good way to implement this pagination here.

- **timestamp_limit** – The maximum timestamp allowed in the results. Default: None
- **reverse** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **limit** – Limit the size of the response, max 50000 and default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_last_trade`(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the most recent trade for a given stock. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_last_quote`(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the most recent NBBO (Quote) tick for a given stock. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_daily_open_close`(*symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the OCHLV and after-hours prices of a stock symbol on a certain date. Official Docs

Parameters

- **symbol** – The ticker symbol we want daily-OCHLV for.
- **date** – The date/day of the daily-OCHLV to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *multiplier: int = 1*, *timespan='day'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a stock over a given date range in custom time window sizes. For example, if `timespan = 'minute'` and `multiplier = '5'` then 5-minute bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **from_date** – The start of the aggregate time window. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. See *polygon.enums.SortOrder* for choices. `asc` default.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **multiplier** – The size of the timespan multiplier. Must be a positive whole number.
- **timespan** – The size of the time window. See *polygon.enums.Timespan* for choices. defaults to `day`
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_grouped_daily_bars(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily OCHLV for the entire stocks/equities markets. Official docs

Parameters

- **date** – The date to get the data for. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_previous_close(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's OCHLV for the specified stock ticker. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.

- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded stock ticker. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_current_price(*symbol: str*) → float
get current market price for the ticker symbol specified.

Uses *get_last_trade()* under the hood Official Docs

Parameters **symbol** – The ticker symbol of the stock/equity.

Returns The current price. A `KeyError` indicates the request wasn't successful.

get_snapshot_all(*symbols: list*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded stock symbols. Official Docs

Parameters

- **symbols** – A comma separated list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_gainers_and_losers(*direction='gainers'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the current top 20 gainers or losers of the day in stocks/equities markets. Official Docs

Parameters

- **direction** – The direction of results. Defaults to gainers. See *polygon.enums.SnapshotDirection* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_trades(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get trades for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values - Async method Official Docs

Parameters

- **symbol** – The ticker symbol we want trades for.
- **date** – The date/day of the trades to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **timestamp** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the next page of results. Default: None. I'm trying to think of a good way to implement pagination support for this type of pagination.
- **timestamp_limit** – The maximum timestamp allowed in the results. Default: None
- **reverse** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **limit** – Limit the size of the response, max 50000 and default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_quotes(*symbol: str*, *date*, *timestamp: Optional[int] = None*, *timestamp_limit: Optional[int] = None*, *reverse: bool = True*, *limit: int = 5000*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get Quotes for a given ticker symbol on a specified date. The response from polygon seems to have a `map` attribute which gives a mapping of attribute names to readable values - Async method Official Docs

Parameters

- **symbol** – The ticker symbol we want quotes for.
- **date** – The date/day of the quotes to retrieve. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **timestamp** – The timestamp offset, used for pagination. Timestamp is the offset at which to start the results. Using the `timestamp` of the last result as the offset will give you the next page of results. Default: None. Thinking of a good way to implement this pagination here.
- **timestamp_limit** – The maximum timestamp allowed in the results. Default: None
- **reverse** – Reverse the order of the results. Default True: oldest first. Make it False for Newest first
- **limit** – Limit the size of the response, max 50000 and default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_last_trade(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get the most recent trade for a given stock - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_last_quote(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get the most recent NBBO (Quote) tick for a given stock - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_daily_open_close(*symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get the OCHLV and after-hours prices of a stock symbol on a certain date - Async method Official Docs

Parameters

- **symbol** – The ticker symbol we want daily-OCHLV for.
- **date** – The date/day of the daily-OCHLV to retrieve. Could be datetime or date or string YYYY-MM-DD
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *multiplier: int = 1*, *timespan='day'*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get aggregate bars for a stock over a given date range in custom time window sizes. For example, if timespan = 'minute' and multiplier = '5' then 5-minute bars will be returned - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **from_date** – The start of the aggregate time window. Could be datetime or date or string YYYY-MM-DD

- **to_date** – The end of the aggregate time window. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. See *polygon.enums.SortOrder* for choices. `asc` default.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **multiplier** – The size of the timespan multiplier. Must be a positive whole number.
- **timespan** – The size of the time window. See *polygon.enums.Timespan* for choices. defaults to `day`
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_grouped_daily_bars(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the daily OCHLV for the entire stocks/equities markets - Async method Official docs

Parameters

- **date** – The date to get the data for. Could be `datetime` or `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_previous_close(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the previous day's OCHLV for the specified stock ticker - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded stock ticker - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_current_price(*symbol: str*) → float

get current market price for the ticker symbol specified - Async method

Uses *async_get_last_trade()* under the hood Official Docs

Parameters **symbol** – The ticker symbol of the stock/equity.

Returns The current price. A `KeyError` indicates the request wasn't successful.

async async_get_snapshot_all(*symbols: list*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded stock symbols - Async method Official Docs

Parameters

- **symbols** – A comma separated list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_gainers_and_losers(*direction='gainers'*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current top 20 gainers or losers of the day in stocks/equities markets - Asnyc method Official Docs

Parameters

- **direction** – The direction of results. Defaults to gainers. See *polygon.enums.SnapshotDirection* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

13.3 Options Client

polygon.options.options.**build_option_symbol**(*underlying_symbol: str*, *expiry*, *call_or_put*, *strike_price*, *prefix_o: bool = False*)

Build the option symbol from the details provided.

Parameters

- **underlying_symbol** – The underlying stock ticker symbol.
- **expiry** – The expiry date for the option. You can pass this argument as `datetime.datetime` or `datetime.date` object. Or a string in format: `YYMMDD`. Using datetime objects is recommended.
- **call_or_put** – The option type. You can specify: `c` or `call` or `p` or `put`. Capital letters are also supported.
- **strike_price** – The strike price for the option. ALWAYS pass this as one number. `145`, `240.5`, `15.003`, `56`, `129.02` are all valid values. It shouldn't have more than three numbers after decimal point.
- **prefix_o** – Whether or not to prefix the symbol with 'O:'. It is needed by polygon endpoints. However all the library functions will automatically add this prefix if you pass in symbols without this prefix.

Returns The option symbol in the format specified by polygon

polygon.options.options.**parse_option_symbol**(*option_symbol: str*, *output_format='object'*, *expiry_format='date'*)

Function to parse an option symbol.

Parameters

- **option_symbol** – the symbol you want to parse. Both `TSLA211015P125000` and `O:TSLA211015P125000` are valid
- **output_format** – Output format of the result. defaults to object. Set it to `dict` or `list` as needed.
- **expiry_format** – The format for the expiry date in the results. Defaults to `date` object. change this param to `string` to get the value as a string: `YYYY-MM-DD`

Returns The parsed values either as an object, list or a dict as indicated by `output_format`.

polygon.options.options.**build_option_symbol_for_tda**(*underlying_symbol: str*, *expiry*, *call_or_put*, *strike_price*)

Only use this function if you need to create option symbol for TD ameritrade API. This function is just a bonus.

Parameters

- **underlying_symbol** – The underlying stock ticker symbol.
- **expiry** – The expiry date for the option. You can pass this argument as `datetime.datetime` or `datetime.date` object. Or a string in format: `MMDDYY`. Using datetime objects is recommended.
- **call_or_put** – The option type. You can specify: `c` or `call` or `p` or `put`. Capital letters are also supported.
- **strike_price** – The strike price for the option. ALWAYS pass this as one number. `145`, `240.5`, `15.003`, `56`, `129.02` are all valid values. It shouldn't have more than three numbers after decimal point.

Returns The option symbol built in the format supported by TD Ameritrade.

polygon.options.options.**parse_option_symbol_from_tda**(*option_symbol: str*, *output_format='object'*, *expiry_format='date'*)

Function to parse an option symbol in format supported by TD Ameritrade.

Parameters

- **option_symbol** – the symbol you want to parse. Both `TSLA211015P125000` and `O:TSLA211015P125000` are valid
- **output_format** – Output format of the result. defaults to object. Set it to `dict` or `list` as needed.
- **expiry_format** – The format for the expiry date in the results. Defaults to `date` object. change this param to `string` to get the value as a string: `YYYY-MM-DD`

Returns The parsed values either as an object, list or a dict as indicated by `output_format`.

class polygon.options.options.**OptionSymbol**(*option_symbol: str*, *output_format*, *expiry_format*, *symbol_format='polygon'*)

The custom object for parsed details from option symbols.

__init__(*option_symbol: str*, *output_format*, *expiry_format*, *symbol_format='polygon'*)

Parses the details from symbol and creates attributes for the object.

Parameters

- **option_symbol** – the symbol you want to parse. Both `TSLA211015P125000` and `O:TSLA211015P125000` are valid
- **expiry_format** – The format for the expiry date in the results. Defaults to `date` object. change this param to `string` to get the value as a string: `YYYY-MM-DD`
- **symbol_format** – Which formatting spec to use. Defaults to polygon. also supports `tda` which is the format supported by TD Ameritrade

class polygon.options.options.**OptionsClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This class implements all the Options REST endpoints. Note that you should always import names from top level. eg: `from polygon import OptionsClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = OptionsClient('MY_API_KEY')` Once you have the client, you can call its methods to get data from the APIs. All methods have sane default values and almost everything can be customized.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

get_trades(*option_symbol: str*, *timestamp=None*, *timestamp_lt=None*, *timestamp_lte=None*, *timestamp_gt=None*, *timestamp_gte=None*, *sort='timestamp'*, *limit: int = 100*, *order='asc'*, *raw_response: bool = False*)

Get trades for an options ticker symbol in a given time range. Note that you need to have an option symbol in correct format for this endpoint. You can use *polygon.reference_apis.reference_api.ReferenceClient.get_option_contracts()* to query option contracts using many filter parameters such as underlying symbol etc. Official Docs

Parameters

- **option_symbol** – The options ticker symbol to get trades for. for eg `O:TSLA210903C00700000`. you can pass the symbol with or without the prefix `O:`
- **timestamp** – Query by trade timestamp. You can supply a `date`, `datetime` object or a `nanosecond UNIX timestamp` or a string in format: `YYYY-MM-DD`.
- **timestamp_lt** – query results where timestamp is less than the supplied value
- **timestamp_lte** – query results where timestamp is less than or equal to the supplied value
- **timestamp_gt** – query results where timestamp is greater than the supplied value
- **timestamp_gte** – query results where timestamp is greater than or equal to the supplied value
- **sort** – Sort field used for ordering. Defaults to timestamp. See *polygon.enums.OptionTradesSort* for available choices.
- **limit** – Limit the number of results returned. Defaults to 100. max is 50000.
- **order** – order of the results. Defaults to `asc`. See *polygon.enums.SortOrder* for info and available choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

get_last_trade(*ticker: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent trade for a given options contract. Official Docs

Parameters

- **ticker** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

get_previous_close(*ticker: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified option contract. Official Docs

Parameters

- **ticker** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

async async_get_trades(*option_symbol: str*, *timestamp=None*, *timestamp_lt=None*, *timestamp_lte=None*, *timestamp_gt=None*, *timestamp_gte=None*, *sort='timestamp'*, *limit: int = 100*, *order='asc'*, *raw_response: bool = False*)

Get trades for an options ticker symbol in a given time range. Note that you need to have an option symbol in correct format for this endpoint. You can use *polygon.reference_apis.reference_api.ReferenceClient.async_get_option_contracts()* to query option contracts using many filter parameters such as underlying symbol etc. Official Docs

Parameters

- **option_symbol** – The options ticker symbol to get trades for. for eg `O:TSLA210903C00700000`. you can pass the symbol with or without the prefix `O:`
- **timestamp** – Query by trade timestamp. You can supply a `date`, `datetime` object or a `nanosecond UNIX timestamp` or a string in format: `YYYY-MM-DD`.
- **timestamp_lt** – query results where timestamp is less than the supplied value
- **timestamp_lte** – query results where timestamp is less than or equal to the supplied value
- **timestamp_gt** – query results where timestamp is greater than the supplied value
- **timestamp_gte** – query results where timestamp is greater than or equal to the supplied value
- **sort** – Sort field used for ordering. Defaults to timestamp. See *polygon.enums.OptionTradesSort* for available choices.
- **limit** – Limit the number of results returned. Defaults to 100. max is 50000.
- **order** – order of the results. Defaults to `asc`. See *polygon.enums.SortOrder* for info and available choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

async async_get_last_trade(*ticker: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get the most recent trade for a given options contract - Async Official Docs

Parameters

- **ticker** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

async async_get_previous_close(*ticker: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]
Get the previous day's open, high, low, and close (OHLC) for the specified option contract - Async Official Docs

Parameters

- **ticker** – The ticker symbol of the options contract. Eg: `O:TSLA210903C00700000`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to false to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns Either a Dictionary or a Response object depending on value of `raw_response`. Defaults to Dict.

polygon.options.options.**ensure_prefix**(*symbol: str*)
Ensure that the option symbol has the prefix `O:` as needed by polygon endpoints. If it does, make no changes. If it doesn't, add the prefix and return the new value.

Parameters **symbol** – the option symbol to check

13.4 References Client

class polygon.reference_apis.reference_api.**ReferenceClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)
These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This class implements all the References REST endpoints. Note that you should always import names from top level. eg: `from polygon import ReferenceClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = ReferenceClient('MY_API_KEY')` Once you have the client, you can call its methods to get data from the APIs. All methods have sane default values and almost everything can be customized.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

get_tickers(*symbol: str = ''*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *symbol_type=''*, *market=''*, *exchange: str = ''*, *cusip: Optional[str] = None*, *cik: str = ''*, *date=None*, *search: Optional[str] = None*, *active: bool = True*, *sort='ticker'*, *order='asc'*, *limit: int = 100*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Query all ticker symbols which are supported by Polygon.io. This API currently includes Stocks/Equities, Crypto, and Forex. Official Docs

Parameters

- **symbol** – Specify a ticker symbol. Defaults to empty string which queries all tickers.
- **ticker_lt** – Return results where this field is less than the value given
- **ticker_lte** – Return results where this field is less than or equal to the value given
- **ticker_gt** – Return results where this field is greater than the value given
- **ticker_gte** – Return results where this field is greater than or equal to the value given
- **symbol_type** – Specify the type of the tickers. See *polygon.enums.TickerType* for common choices. Find all supported types via the Ticker Types API Defaults to empty string which queries all types.
- **market** – Filter by market type. By default all markets are included. See *polygon.enums.TickerMarketType* for available choices.
- **exchange** – Specify the primary exchange of the asset in the ISO code format. Find more information about the ISO codes at the ISO org website. Defaults to empty string which queries all exchanges.
- **cusip** – Specify the `CUSIP` code of the asset you want to search for. Find more information about CUSIP codes on their website Defaults to empty string which queries all CUSIPs
- **cik** – Specify the `CIK` of the asset you want to search for. Find more information about CIK codes at their website Defaults to empty string which queries all CIKs.

- **date** – Specify a point in time to retrieve tickers available on that date. Defaults to the most recent available date. Could be `datetime`, `date` or a string `YYYY-MM-DD`
- **search** – Search for terms within the ticker and/or company name. for eg `MS` will match matching symbols
- **active** – Specify if the tickers returned should be actively traded on the queried date. Default is True
- **sort** – The field to sort the results on. Default is ticker. If the search query parameter is present, sort is ignored and results are ordered by relevance. See *polygon.enums.TickerSortType* for available choices.
- **order** – The order to sort the results on. Default is asc. See *polygon.enums.SortOrder* for available choices.
- **limit** – Limit the size of the response, default is 100 and max is 1000. `Pagination` is supported by the pagination function below
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

static get_ticker_types(**args*, ***kwargs*) → None

DEPRECATED! Replaced by *get_ticker_types_v3()*. This method will be removed in a future version from the library.

Get a mapping of ticker types to their descriptive names. Official Docs

get_ticker_types_v3(*asset_class=None*, *locale=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a mapping of ticker types to their descriptive names. Official Docs

Parameters

- **asset_class** – Filter by asset class. see *polygon.enums.AssetClass* for choices
- **locale** – Filter by locale. See *polygon.enums.Locale* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_ticker_details(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get details for a ticker symbol's company/entity. This provides a general overview of the entity with information such as name, sector, exchange, logo and similar companies.

This endpoint will be replaced by *get_ticker_details_vx()* in future. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_ticker_details_vx`(*symbol: str*, *date=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

This API is Experimental and will replace *`get_ticker_details()`* in future.

Get a single ticker supported by Polygon.io. This response will have detailed information about the ticker and the company behind it. Official Docs

Parameters

- **`symbol`** – The ticker symbol of the asset.
- **`date`** – Specify a point in time to get information about the ticker available on that date. When retrieving information from SEC filings, we compare this date with the period of report date on the SEC filing. Defaults to the most recent available date.
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_option_contracts`(*underlying_ticker: Optional[str] = None*, *ticker: Optional[str] = None*, *contract_type=None*, *expiration_date=None*, *expiration_date_lt=None*, *expiration_date_lte=None*, *expiration_date_gt=None*, *expiration_date_gte=None*, *order='asc'*, *sort=None*, *limit=100*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

List currently active options contracts Official Docs

Parameters

- **`underlying_ticker`** – Query for contracts relating to an underlying stock ticker.
- **`ticker`** – Query for a contract by option ticker.
- **`contract_type`** – Query by the type of contract. see *`polygon.enums.OptionsContractType`* for choices
- **`expiration_date`** – Query by contract expiration date. either `datetime`, `date` or string `YYYY-MM-DD`
- **`expiration_date_lt`** – expiration date less than given value
- **`expiration_date_lte`** – expiration date less than equal to given value
- **`expiration_date_gt`** – expiration_date greater than given value
- **`expiration_date_gte`** – expiration_date greater than equal to given value
- **`order`** – Order of results. See *`polygon.enums.SortOrder`* for choices.
- **`sort`** – Sort field for ordering. See *`polygon.enums.OptionsContractsSortType`* for choices.
- **`limit`** – Number of results to return
- **`raw_response`** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_ticker_news(*symbol: Optional[str] = None*, *limit: int = 100*, *order='desc'*, *sort='published_utc'*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *published_utc=None*, *published_utc_lt=None*, *published_utc_lte=None*, *published_utc_gt=None*, *published_utc_gte=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the most recent news articles relating to a stock ticker symbol, including a summary of the article and a link to the original source. Official Docs

Parameters

- **symbol** – To get news mentioning the name given. Defaults to empty string which doesn't filter tickers
- **limit** – Limit the size of the response, default is 100 and max is 1000. Use pagination helper function for larger responses.
- **order** – Order the results. See *polygon.enums.SortOrder* for choices.
- **sort** – The field key to sort. See *polygon.enums.TickerNewsSort* for choices.
- **ticker_lt** – Return results where this field is less than the value.
- **ticker_lte** – Return results where this field is less than or equal to the value.
- **ticker_gt** – Return results where this field is greater than the value
- **ticker_gte** – Return results where this field is greater than or equal to the value.
- **published_utc** – A date string `YYYY-MM-DD` or `datetime` for published date time filters.
- **published_utc_lt** – Return results where this field is less than the value given
- **published_utc_lte** – Return results where this field is less than or equal to the value given
- **published_utc_gt** – Return results where this field is greater than the value given
- **published_utc_gte** – Return results where this field is greater than or equal to the value given
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_stock_dividends(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a list of historical dividends for a stock, including the relevant dates and the amount of the dividend. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_stock_financials(*symbol: str*, *limit: int = 100*, *report_type=None*, *sort=None*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historical financial data for a stock ticker. This API will be replaced by *get_stock_financials_vx()* in future. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **limit** – Limit the number of results. Defaults to 100
- **report_type** – Specify a type of report to return. see *polygon.enums.StockReportType* for choices. Defaults to `None`
- **sort** – The key for sorting the results. see *polygon.enums.StockFinancialsSortType* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_stock_financials_vx(*ticker: Optional[str] = None*, *cik: Optional[str] = None*, *company_name: Optional[str] = None*, *company_name_search: Optional[str] = None*, *sic: Optional[str] = None*, *filing_date=None*, *filing_date_lt=None*, *filing_date_lte=None*, *filing_date_gt=None*, *filing_date_gte=None*, *period_of_report_date=None*, *period_of_report_date_lt=None*, *period_of_report_date_lte=None*, *period_of_report_date_gt=None*, *period_of_report_date_gte=None*, *time_frame=None*, *include_sources: bool = False*, *order='asc'*, *limit: int = 50*, *sort='filing_date'*, *raw_response: bool = False*)

Get historical financial data for a stock ticker. The financials data is extracted from XBRL from company SEC filings using this methodology Official Docs

This API is experimental and will replace *get_stock_financials()* in future.

Parameters

- **ticker** – Filter query by company ticker.
- **cik** – filter the Query by `central index key (CIK)` Number
- **company_name** – filter the query by company name
- **company_name_search** – partial match text search for company names
- **sic** – Query by `standard industrial classification (SIC)`
- **filing_date** – Query by the date when the filing with financials data was filed. `datetime/date` or string `YYYY-MM-DD`
- **filing_date_lt** – filter for filing date less than given value
- **filing_date_lte** – filter for filing date less than equal to given value
- **filing_date_gt** – filter for filing date greater than given value
- **filing_date_gte** – filter for filing date greater than equal to given value
- **period_of_report_date** – query by The period of report for the filing with financials data. `datetime/date` or string in format: `YYY-MM-DD`.
- **period_of_report_date_lt** – filter for period of report date less than given value

- **period_of_report_date_lte** – filter for period of report date less than equal to given value
- **period_of_report_date_gt** – filter for period of report date greater than given value
- **period_of_report_date_gte** – filter for period of report date greater than equal to given value
- **time_frame** – Query by timeframe. Annual financials originate from 10-K filings, and quarterly financials originate from 10-Q filings. Note: Most companies do not file quarterly reports for Q4 and instead include those financials in their annual report, so some companies my not return quarterly financials for Q4. See *polygon.enums.StockFinancialsTimeframe* for choices.
- **include_sources** – Whether or not to include the xpath and formula attributes for each financial data point. See the xpath and formula response attributes for more info. `False` by default
- **order** – Order results based on the sort field. 'asc' by default. See *polygon.enums.SortOrder* for choices.
- **limit** – number of max results to obtain. defaults to 50.
- **sort** – Sort field key used for ordering. 'filing_date' default. see *polygon.enums.StockFinancialsSortKey* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_stock_splits(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a list of historical stock splits for a ticker symbol, including the execution and payment dates of the stock split, and the split ratio. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_market_holidays(*raw_response: bool = False*) → Union[requests.models.Response, dict]

Get upcoming market holidays and their open/close times. Official Docs

Parameters **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_market_status(*raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current trading status of the exchanges and overall financial markets. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_condition_mappings(*tick_type='trades'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get a unified numerical mapping for conditions on trades and quotes. Each feed/exchange uses its own set of codes to identify conditions, so the same condition may have a different code depending on the originator of the data. Polygon.io defines its own mapping to allow for uniformly identifying a condition across feeds/exchanges. Official Docs

Parameters

- **tick_type** – The type of ticks to return mappings for. Defaults to 'trades'. See *polygon.enums.ConditionMappingTickType* for choices.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_conditions(*asset_class=None*, *data_type=None*, *id=None*, *sip=None*, *order=None*, *limit: int = 50*, *sort='name'*, *raw_response: bool = False*)

List all conditions that Polygon.io uses. Official Docs

Parameters

- **asset_class** – Filter for conditions within a given asset class. See *polygon.enums.AssetClass* for choices. Defaults to all assets.
- **data_type** – Filter by data type. See *polygon.enums.ConditionsDataType* for choices. defaults to all.
- **id** – Filter for conditions with a given ID
- **sip** – Filter by SIP. If the condition contains a mapping for that SIP, the condition will be returned.
- **order** – Order results. See *polygon.enums.SortOrder* for choices.
- **limit** – limit the number of results. defaults to 50.
- **sort** – Sort field used for ordering. Defaults to 'name'. See *polygon.enums.ConditionsSortKey* for choices.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_exchanges(*asset_class=None*, *locale=None*, *raw_response: bool = False*)

List all exchanges that Polygon.io knows about. Official Docs

Parameters

- **asset_class** – filter by asset class. See *polygon.enums.AssetClass* for choices.

- **locale** – Filter by locale name. See *polygon.enums.Locale*
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

static get_stock_exchanges(**args*, ***kwargs*)
DEPRECATED! Replaced by *get_exchanges()*. This method will be removed in a future version from the library

static get_crypto_exchanges(**args*, ***kwargs*)
DEPRECATED! Replaced by *get_exchanges()*. This method will be removed in a future version from the library

get_locales(*raw_response: bool = False*) → Union[requests.models.Response, dict]
Get a list of locales currently supported by Polygon.io. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_markets(*raw_response: bool = False*) → Union[requests.models.Response, dict]
Get a list of markets that are currently supported by Polygon.io. Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_tickers(*symbol: str = ''*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *symbol_type=''*, *market=''*, *exchange: str = ''*, *cusip: Optional[str] = None*, *cik: str = ''*, *date: Optional[Union[str, datetime.date, datetime.datetime]] = None*, *search: Optional[str] = None*, *active: bool = True*, *sort='ticker'*, *order: str = 'asc'*, *limit: int = 100*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Query all ticker symbols which are supported by Polygon.io. This API currently includes Stocks/Equities, Crypto, and Forex - Assync method Official Docs

Parameters

- **symbol** – Specify a ticker symbol. Defaults to empty string which queries all tickers.
- **ticker_lt** – Return results where this field is less than the value given
- **ticker_lte** – Return results where this field is less than or equal to the value given
- **ticker_gt** – Return results where this field is greater than the value given
- **ticker_gte** – Return results where this field is greater than or equal to the value given
- **symbol_type** – Specify the type of the tickers. See *polygon.enums.TickerType* for common choices. Find all supported types via the Ticker Types API Defaults to empty string which queries all types.

- **market** – Filter by market type. By default all markets are included. See *polygon.enums.TickerMarketType* for available choices.
- **exchange** – Specify the primary exchange of the asset in the ISO code format. Find more information about the ISO codes at the ISO org website. Defaults to empty string which queries all exchanges.
- **cusip** – Specify the CUSIP code of the asset you want to search for. Find more information about CUSIP codes on their website Defaults to empty string which queries all CUSIPs
- **cik** – Specify the CIK of the asset you want to search for. Find more information about CIK codes at their website Defaults to empty string which queries all CIKs.
- **date** – Specify a point in time to retrieve tickers available on that date. Defaults to the most recent available date. Could be `datetime`, `date` or a string `YYYY-MM-DD`
- **search** – Search for terms within the ticker and/or company name. for eg `MS` will match matching symbols
- **active** – Specify if the tickers returned should be actively traded on the queried date. Default is True
- **sort** – The field to sort the results on. Default is ticker. If the search query parameter is present, sort is ignored and results are ordered by relevance. See *polygon.enums.TickerSortType* for available choices.
- **order** – The order to sort the results on. Default is asc. See *polygon.enums.SortOrder* for available choices.
- **limit** – Limit the size of the response, default is 100 and max is 1000. `Pagination` is supported by the pagination function below
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async static async_get_ticker_types(**args*, ***kwargs*) → None

DEPRECATED! Replaced by *async_get_ticker_types_v3()*. This method will be removed in a future version from the library.

Get a mapping of ticker types to their descriptive names. Official Docs

async async_get_ticker_types_v3(*asset_class=None*, *locale=None*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get a mapping of ticker types to their descriptive names - Async method Official Docs

Parameters

- **asset_class** – Filter by asset class. see *polygon.enums.AssetClass* for choices
- **locale** – Filter by locale. See *polygon.enums.Locale* for choices
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_ticker_details(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get details for a ticker symbol's company/entity. This provides a general overview of the entity with information such as name, sector, exchange, logo and similar companies - Async method

This endpoint will be replaced by *async_get_ticker_details_vx()* in future. Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_ticker_details_vx(*symbol: str*, *date=None*, *raw_response: bool = False*) → Union[httpx.Response, dict]

This API is Experimental and will replace *async_get_ticker_details()* in future - Async method

Get a single ticker supported by Polygon.io. This response will have detailed information about the ticker and the company behind it. Official Docs

Parameters

- **symbol** – The ticker symbol of the asset.
- **date** – Specify a point in time to get information about the ticker available on that date. When retrieving information from SEC filings, we compare this date with the period of report date on the SEC filing. Defaults to the most recent available date.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_option_contracts(*underlying_ticker: Optional[str] = None*, *ticker: Optional[str] = None*, *contract_type=None*, *expiration_date=None*, *expiration_date_lt=None*, *expiration_date_lte=None*, *expiration_date_gt=None*, *expiration_date_gte=None*, *order='asc'*, *sort=None*, *limit: int = 50*, *raw_response: bool = False*) → Union[httpx.Response, dict]

List currently active options contracts - Async method Official Docs

Parameters

- **underlying_ticker** – Query for contracts relating to an underlying stock ticker.
- **ticker** – Query for a contract by option ticker.
- **contract_type** – Query by the type of contract. see *polygon.enums.OptionsContractType* for choices
- **expiration_date** – Query by contract expiration date. either `datetime`, `date` or string `YYYY-MM-DD`
- **expiration_date_lt** – expiration date less than given value
- **expiration_date_lte** – expiration date less than equal to given value

- **expiration_date_gt** – expiration_date greater than given value
- **expiration_date_gte** – expiration_date greater than equal to given value
- **order** – Order of results. See *polygon.enums.SortOrder* for choices.
- **sort** – Sort field for ordering. See *polygon.enums.OptionsContractsSortType* for choices.
- **limit** – Number of results to return
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_ticker_news(*symbol: Optional[str] = None*, *limit: int = 100*, *order='desc'*, *sort='published_utc'*, *ticker_lt=None*, *ticker_lte=None*, *ticker_gt=None*, *ticker_gte=None*, *published_utc=None*, *published_utc_lt=None*, *published_utc_lte=None*, *published_utc_gt=None*, *published_utc_gte=None*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the most recent news articles relating to a stock ticker symbol, including a summary of the article and a link to the original source - Async method Official Docs

Parameters

- **symbol** – To get news mentioning the name given. Defaults to empty string which doesn't filter tickers
- **limit** – Limit the size of the response, default is 100 and max is 1000. Use pagination helper function for larger responses.
- **order** – Order the results. See *polygon.enums.SortOrder* for choices.
- **sort** – The field key to sort. See *polygon.enums.TickerNewsSort* for choices.
- **ticker_lt** – Return results where this field is less than the value.
- **ticker_lte** – Return results where this field is less than or equal to the value.
- **ticker_gt** – Return results where this field is greater than the value
- **ticker_gte** – Return results where this field is greater than or equal to the value.
- **published_utc** – A date string `YYYY-MM-DD` or `datetime` for published date time filters.
- **published_utc_lt** – Return results where this field is less than the value given
- **published_utc_lte** – Return results where this field is less than or equal to the value given
- **published_utc_gt** – Return results where this field is greater than the value given
- **published_utc_gte** – Return results where this field is greater than or equal to the value given
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_stock_dividends(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get a list of historical dividends for a stock, including the relevant dates and the amount of the dividend - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_stock_financials(*symbol: str*, *limit: int = 100*, *report_type=None*, *sort=None*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get historical financial data for a stock ticker. This API will be replaced by *async_get_stock_financials_vx()* in future - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.
- **limit** – Limit the number of results. Defaults to 100
- **report_type** – Specify a type of report to return. see *polygon.enums.StockReportType* for choices. Defaults to `None`
- **sort** – The key for sorting the results. see *polygon.enums.StockFinancialsSortType* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_stock_financials_vx(*ticker: Optional[str] = None*, *cik: Optional[str] = None*, *company_name: Optional[str] = None*, *company_name_search: Optional[str] = None*, *sic: Optional[str] = None*, *filing_date=None*, *filing_date_lt=None*, *filing_date_lte=None*, *filing_date_gt=None*, *filing_date_gte=None*, *period_of_report_date=None*, *period_of_report_date_lt=None*, *period_of_report_date_lte=None*, *period_of_report_date_gt=None*, *period_of_report_date_gte=None*, *time_frame=None*, *include_sources: bool = False*, *order='asc'*, *limit: int = 50*, *sort='filing_date'*, *raw_response: bool = False*)

Get historical financial data for a stock ticker. The financials data is extracted from XBRL from company SEC filings using this methodology - Async method Official Docs

This API is experimental and will replace *async_get_stock_financials()* in future.

Parameters

- **ticker** – Filter query by company ticker.
- **cik** – filter the Query by `central index key (CIK)` Number

- **company_name** – filter the query by company name
- **company_name_search** – partial match text search for company names
- **sic** – Query by `standard industrial classification (SIC)`
- **filing_date** – Query by the date when the filing with financials data was filed. `datetime/date` or string `YYYY-MM-DD`
- **filing_date_lt** – filter for filing date less than given value
- **filing_date_lte** – filter for filing date less than equal to given value
- **filing_date_gt** – filter for filing date greater than given value
- **filing_date_gte** – filter for filing date greater than equal to given value
- **period_of_report_date** – query by The period of report for the filing with financials data. `datetime/date` or string in format: `YYY-MM-DD`.
- **period_of_report_date_lt** – filter for period of report date less than given value
- **period_of_report_date_lte** – filter for period of report date less than equal to given value
- **period_of_report_date_gt** – filter for period of report date greater than given value
- **period_of_report_date_gte** – filter for period of report date greater than equal to given value
- **time_frame** – Query by timeframe. Annual financials originate from 10-K filings, and quarterly financials originate from 10-Q filings. Note: Most companies do not file quarterly reports for Q4 and instead include those financials in their annual report, so some companies my not return quarterly financials for Q4. See *polygon.enums.StockFinancialsTimeframe* for choices.
- **include_sources** – Whether or not to include the xpath and formula attributes for each financial data point. See the xpath and formula response attributes for more info. `False` by default
- **order** – Order results based on the sort field. 'asc' by default. See *polygon.enums.SortOrder* for choices.
- **limit** – number of max results to obtain. defaults to 50.
- **sort** – Sort field key used for ordering. 'filing_date' default. see *polygon.enums.StockFinancialsSortKey* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_stock_splits(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get a list of historical stock splits for a ticker symbol, including the execution and payment dates of the stock split, and the split ratio - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the stock/equity.

- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_market_holidays(*raw_response: bool = False*) → Union[httpx.Response, dict]
Get upcoming market holidays and their open/close times - Async method Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_market_status(*raw_response: bool = False*) → Union[httpx.Response, dict]
Get the current trading status of the exchanges and overall financial markets - Async method Official Docs

Parameters **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_condition_mappings(*tick_type='trades'*, *raw_response: bool = False*) → Union[httpx.Response, dict]
Get a unified numerical mapping for conditions on trades and quotes. Each feed/exchange uses its own set of codes to identify conditions, so the same condition may have a different code depending on the originator of the data. Polygon.io defines its own mapping to allow for uniformly identifying a condition across feeds/exchanges - Async method Official Docs

Parameters

- **tick_type** – The type of ticks to return mappings for. Defaults to 'trades'. See *polygon.enums.ConditionMappingTickType* for choices.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_conditions(*asset_class=None*, *data_type=None*, *id=None*, *sip=None*, *order=None*, *limit: int = 50*, *sort='name'*, *raw_response: bool = False*)
List all conditions that Polygon.io uses - Async method Official Docs

Parameters

- **asset_class** – Filter for conditions within a given asset class. See *polygon.enums.AssetClass* for choices. Defaults to all assets.
- **data_type** – Filter by data type. See *polygon.enums.ConditionsDataType* for choices. defaults to all.
- **id** – Filter for conditions with a given ID
- **sip** – Filter by SIP. If the condition contains a mapping for that SIP, the condition will be returned.

- **order** – Order results. See *polygon.enums.SortOrder* for choices.
- **limit** – limit the number of results. defaults to 50.
- **sort** – Sort field used for ordering. Defaults to 'name'. See *polygon.enums.ConditionsSortKey* for choices.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_exchanges(*asset_class=None*, *locale=None*, *raw_response: bool = False*)
List all exchanges that Polygon.io knows about - Async method Official Docs

Parameters

- **asset_class** – filter by asset class. See *polygon.enums.AssetClass* for choices.
- **locale** – Filter by locale name. See *polygon.enums.Locale*
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async static async_get_stock_exchanges(***kwargs*)
DEPRECATED! Replaced by *async_get_exchanges()*. This method will be removed in a future version from the library

async static async_get_crypto_exchanges(***kwargs*)
DEPRECATED! Replaced by *async_get_exchanges()*. This method will be removed in a future version from the library

async async_get_locales(*raw_response: bool = False*) → Union[httpx.Response, dict]
Get a list of locales currently supported by Polygon.io - Async method Official Docs

Parameters **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_markets(*raw_response: bool = False*) → Union[httpx.Response, dict]
Get a list of markets that are currently supported by Polygon.io - Async method Official Docs

Parameters **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

13.5 Forex Client

class polygon.forex.forex_api.**ForexClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This class implements all the Forex REST endpoints. Note that you should always import names from top level. eg: `from polygon import ForexClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = ForexClient('MY_API_KEY')` Once you have the client, you can call its methods to get data from the APIs. All methods have sane default values and almost everything can be customized.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

get_historic_forex_ticks(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historic trade ticks for a forex currency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the forex currency pair.
- **to_symbol** – The "to" symbol of the forex currency pair.
- **date** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **offset** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm thinking about a good way to implement this type of pagination in the lib which doesn't have a `next_url` in the response attributes.

- **limit** – Limit the size of the response, max 10000. Default 500
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_last_quote(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the last trade tick for a forex currency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the forex currency pair.
- **to_symbol** – The "to" symbol of the forex currency pair.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a forex pair over a given date range in custom time window sizes. For example, if timespan = 'minute' and multiplier = '5' then 5-minute bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair. eg: C:EURUSD. You can supply with or without prefix C:
- **from_date** – The start of the aggregate time window. Could be datetime, date or string YYYY-MM-DD
- **to_date** – The end of the aggregate time window. Could be datetime, date or string YYYY-MM-DD
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. see *polygon.enums.SortOrder* for available choices. Defaults to asc which is oldest at the top.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

get_grouped_daily_bars(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire forex markets. Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_previous_close(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified forex pair. Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_snapshot_all(*symbols: list*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded forex symbols Official Docs

Parameters

- **symbols** – A list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded forex symbol. Official Docs

Parameters

- **symbol** – Symbol of the forex pair. eg: `C:EURUSD`. You can supply with or without prefix `C:`.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_gainers_and_losers`(*direction='gainers'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current top 20 gainers or losers of the day in forex markets. Official docs

Parameters

- **direction** – The direction of the snapshot results to return. See *polygon.enums.SnapshotDirection* for available choices. Defaults to Gainers.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`real_time_currency_conversion`(*from_symbol: str*, *to_symbol: str*, *amount: float*, *precision: int = 2*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get currency conversions using the latest market conversion rates. Note than you can convert in both directions. For example USD to CAD or CAD to USD. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **amount** – The amount to convert,
- **precision** – The decimal precision of the conversion. Defaults to 2 which is 2 decimal places accuracy.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`async async_get_historic_forex_ticks`(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get historic trade ticks for a forex currency pair - Async method. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the forex currency pair.
- **to_symbol** – The "to" symbol of the forex currency pair.
- **date** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **offset** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm thinking about a good way to implement this type of pagination in the lib which doesn't have a `next_url` in the response attributes.
- **limit** – Limit the size of the response, max 10000. Default 500

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_last_quote(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the last trade tick for a forex currency pair - Async method Official Docs

Parameters

- **from_symbol** – The "from" symbol of the forex currency pair.
- **to_symbol** – The "to" symbol of the forex currency pair.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get aggregate bars for a forex pair over a given date range in custom time window sizes. For example, if `timespan = 'minute'` and `multiplier = '5'` then `5-minute` bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair. eg: `C:EURUSD`. You can supply with or without prefix `C:`
- **from_date** – The start of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Sort the results by timestamp. see *polygon.enums.SortOrder* for available choices. Defaults to `asc` which is oldest at the top.
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_grouped_daily_bars(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire forex markets - Async method Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_previous_close(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified forex pair - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the forex pair.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_snapshot_all(*symbols: list*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded forex symbols - Async method Official Docs

Parameters

- **symbols** – A list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded forex symbol - Async method Official Docs

Parameters

- **symbol** – Symbol of the forex pair. eg: `C:EURUSD`. You can supply with or without prefix `C:`.

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_gainers_and_losers(*direction='gainers'*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current top 20 gainers or losers of the day in forex markets. Official docs

Parameters

- **direction** – The direction of the snapshot results to return. See *polygon.enums.SnapshotDirection* for available choices. Defaults to Gainers.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_real_time_currency_conversion(*from_symbol: str*, *to_symbol: str*, *amount: float*, *precision: int = 2*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get currency conversions using the latest market conversion rates. Note than you can convert in both directions. For example USD to CAD or CAD to USD - Async method Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **amount** – The amount to convert,
- **precision** – The decimal precision of the conversion. Defaults to 2 which is 2 decimal places accuracy.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

13.6 Crypto Client

class polygon.crypto.crypto_api.**CryptoClient**(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

This class implements all the crypto REST endpoints. Note that you should always import names from top level. eg: `from polygon import CryptoClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = CryptoClient('MY_API_KEY')` Once you have the client, you can call its methods to get data from the APIs. All methods have sane default values and almost everything can be customized.

Any method starting with `async_` in its name is meant to be for async programming. All methods have their sync and async counterparts. Any async method must be awaited while non-async (or sync) methods should be called directly.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

It is also a very good idea to visit the official documentation. I highly recommend using the UI there to play with the endpoints a bit. Observe the data you receive as the actual data received through python lib is exactly the same as shown on their page when you click `Run Query`.

__init__(*api_key: str*, *use_async: bool = False*, *connect_timeout: int = 10*, *read_timeout: int = 10*)

Initiates a Client to be used to access all the endpoints.

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **use_async** – Set to True to get an async client. Defaults to False which returns a non-async client.
- **connect_timeout** – The connection timeout in seconds. Defaults to 10. basically the number of seconds to wait for a connection to be established. Raises a `ConnectTimeout` if unable to connect within specified time limit.
- **read_timeout** – The read timeout in seconds. Defaults to 10. basically the number of seconds to wait for date to be received. Raises a `ReadTimeout` if unable to connect within the specified time limit.

get_historic_trades(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get historic trade ticks for a cryptocurrency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the crypto pair.
- **to_symbol** – The "to" symbol of the crypto pair.
- **date** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **offset** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm trying to think of a good way to implement pagination in the library for these endpoints which do not return a `next_url` attribute.
- **limit** – Limit the size of the response, max 10000. Default 500
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_last_trade(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the last trade tick for a cryptocurrency pair. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_daily_open_close(*from_symbol: str*, *to_symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the open, close prices of a cryptocurrency symbol on a certain day. Official Docs:

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **date** – The date of the requested open/close. Could be `datetime`, `date` or string `YYYY-MM-DD`.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get aggregate bars for a cryptocurrency pair over a given date range in custom time window sizes. For example, if `timespan='minute'` and `multiplier='5'` then 5-minute bars will be returned. Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without prefix `X:`
- **from_date** – The start of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices

- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Order of sorting the results. See *polygon.enums.SortOrder* for available choices. Defaults to `asc` (oldest at the top)
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_grouped_daily_bars`(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire cryptocurrency market. Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_previous_close`(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified cryptocurrency pair. Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without the prefix `X:`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

`get_snapshot_all`(*symbols: list*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded cryptocurrency symbols Official Docs

Parameters

- **symbols** – A list of tickers to get snapshots for.

- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded cryptocurrency symbol. Official Docs

Parameters

- **symbol** – Symbol of the currency pair. eg: `X:BTCUSD`. you can specify with or without prefix `X:`
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_gainers_and_losers(*direction='gainers'*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current top 20 gainers or losers of the day in cryptocurrency markets. Official docs

Parameters

- **direction** – The direction of the snapshot results to return. See *polygon.enums.SnapshotDirection* for available choices. Defaults to Gainers.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

get_level2_book(*symbol: str*, *raw_response: bool = False*) → Union[requests.models.Response, dict]

Get the current level 2 book of a single ticker. This is the combined book from all of the exchanges. Official Docs

Parameters

- **symbol** – The cryptocurrency ticker. eg: `X:BTCUSD`. You can specify with or without the prefix `` `X: ``
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_historic_trades(*from_symbol: str*, *to_symbol: str*, *date*, *offset: Optional[Union[str, int]] = None*, *limit: int = 500*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get historic trade ticks for a cryptocurrency pair - Async method. Official Docs

Parameters

- **from_symbol** – The "from" symbol of the crypto pair.

- **to_symbol** – The "to" symbol of the crypto pair.
- **date** – The date/day of the historic ticks to retrieve. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **offset** – The timestamp offset, used for pagination. This is the offset at which to start the results. Using the timestamp of the last result as the offset will give you the next page of results. I'm trying to think of a good way to implement pagination in the library for these endpoints which do not return a `next_url` attribute.
- **limit** – Limit the size of the response, max 10000. Default 500
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_last_trade(*from_symbol: str*, *to_symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the last trade tick for a cryptocurrency pair - Async method Official Docs

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_daily_open_close(*from_symbol: str*, *to_symbol: str*, *date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the open, close prices of a cryptocurrency symbol on a certain day - Async method Official Docs:

Parameters

- **from_symbol** – The "from" symbol of the pair.
- **to_symbol** – The "to" symbol of the pair.
- **date** – The date of the requested open/close. Could be `datetime`, `date` or string `YYYY-MM-DD`.
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_aggregate_bars(*symbol: str*, *from_date*, *to_date*, *multiplier: int = 1*, *timespan='day'*, *adjusted: bool = True*, *sort='asc'*, *limit: int = 5000*, *raw_response: bool = False*) → Union[httpx.Response, dict]

et aggregate bars for a cryptocurrency pair over a given date range in custom time window sizes. For example, if `timespan='minute'` and `multiplier='5'` then 5-minute bars will be returned - Async

method Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without prefix `X:`
- **from_date** – The start of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **to_date** – The end of the aggregate time window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **multiplier** – The size of the timespan multiplier
- **timespan** – The size of the time window. Defaults to day candles. see *polygon.enums.Timespan* for choices
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **sort** – Order of sorting the results. See *polygon.enums.SortOrder* for available choices. Defaults to `asc` (oldest at the top)
- **limit** – Limits the number of base aggregates queried to create the aggregate results. Max 50000 and Default 5000.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_grouped_daily_bars(*date*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the daily open, high, low, and close (OHLC) for the entire cryptocurrency market - Async method Official Docs

Parameters

- **date** – The date for the aggregate window. Could be `datetime`, `date` or string `YYYY-MM-DD`
- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

async async_get_previous_close(*symbol: str*, *adjusted: bool = True*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the previous day's open, high, low, and close (OHLC) for the specified cryptocurrency pair - Async method Official Docs

Parameters

- **symbol** – The ticker symbol of the currency pair. eg: `X:BTCUSD`. You can specify with or without the prefix `X:`

- **adjusted** – Whether or not the results are adjusted for splits. By default, results are adjusted. Set this to False to get results that are NOT adjusted for splits.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_snapshot_all(*symbols: list*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for all traded cryptocurrency symbols - Async method Official Docs

Parameters

- **symbols** – A list of tickers to get snapshots for.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_snapshot(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current minute, day, and previous day's aggregate, as well as the last trade and quote for a single traded cryptocurrency symbol - Async method Official Docs

Parameters

- **symbol** – Symbol of the currency pair. eg: X:BTCUSD. you can specify with or without prefix X:
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_gainers_and_losers(*direction='gainers'*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current top 20 gainers or losers of the day in cryptocurrency markets - Async method Official docs

Parameters

- **direction** – The direction of the snapshot results to return. See *polygon.enums.SnapshotDirection* for available choices. Defaults to Gainers.
- **raw_response** – Whether or not to return the Response Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make raw_response=True to get underlying response object

async async_get_level2_book(*symbol: str*, *raw_response: bool = False*) → Union[httpx.Response, dict]

Get the current level 2 book of a single ticker. combined book from all of the exchanges - Async method Official Docs

Parameters

- **symbol** – The cryptocurrency ticker. eg: `X:BTCUSD`. You can specify with or without the prefix `X:.
- **raw_response** – Whether or not to return the `Response` Object. Useful for when you need to say check the status code or inspect the headers. Defaults to False which returns the json decoded dictionary.

Returns A JSON decoded Dictionary by default. Make `raw_response=True` to get underlying response object

13.7 Callback Streamer Client (Sync)

class polygon.streaming.streaming.**StreamClient**(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *on_message=None*, *on_close=None*, *on_error=None*, *enable_connection_logs: bool = False*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

Note that this is callback based stream client which is suitable for threaded/multi-processed applications. If you need to stream using an `asyncio` based stream client, see *Async Streamer Client*.

This class implements all the websocket endpoints. Note that you should always import names from top level. eg: `from polygon import StreamClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = StreamClient('MY_API_KEY', 'other_options')`

Once you have the client, you can call its methods to subscribe/unsubscribe to streams, change handlers and process messages. All methods have sane default values and almost everything can be customized.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

Take a look at the Official documentation to get an idea of the stream, data formatting for messages and related useful stuff.

__init__(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *on_message=None*, *on_close=None*, *on_error=None*, *enable_connection_logs: bool = False*)

Initializes the callback function based stream client Official Docs

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **cluster** – Which market/cluster to connect to. See *polygon.enums.StreamCluster* for choices. NEVER connect to the same cluster again if there is an existing stream connected to it. The existing connection would be dropped and new one will be established. You can have up to 4 concurrent streams connected to 4 different clusters.
- **host** – Host url to connect to. Default is real time. See *polygon.enums.StreamHost* for choices.
- **on_message** – The function to be called when data is received. This is primary function you'll write to process the data from the stream. The function should accept one and only one `arg` (message). Default handler is *_default_on_msg()*.
- **on_close** – The function to be called when stream is closed. Function should accept two args (close_status_code, close_message). Default handler is *_default_on_close()*
- **on_error** – Function to be called when an error is encountered. Function should accept one arg (exception object). Default handler is *_default_on_error()*

- **enable_connection_logs** – Whether or not to print debug info related to the stream connection. Helpful for debugging.

_start_stream(*ping_interval: int = 21*, *ping_timeout: int = 20*, *ping_payload: str = ''*, *skip_utf8_validation: bool = True*)

Starts the Stream Event Loop. The loop is infinite and will continue to run until the stream is terminated, either manually or due to an exception. This method is for internal use only. you should always use *start_stream_thread()* to start the stream.

Parameters

- **ping_interval** – client would send a `ping` every specified number of seconds to server to keep connection alive. Set to 0 to disable pinging. Defaults to 21 seconds
- **ping_timeout** – Timeout in seconds if a `pong` (response to ping from server) is not received. The Stream is terminated as it is considered to be dead if no pong is received within the specified timeout. default: 20 seconds
- **ping_payload** – The option message to be sent with the ping. Better to leave it empty string.
- **skip_utf8_validation** – Whether to skip utf validation of messages. Defaults to True. Setting it to False may result in performance downgrade

Returns None

start_stream_thread(*ping_interval: int = 21*, *ping_timeout: int = 20*, *ping_payload: str = ''*, *skip_utf8_validation: bool = True*)

Starts the Stream. This will not block the main thread and it spawns the streamer in its own thread.

Parameters

- **ping_interval** – client would send a `ping` every specified number of seconds to server to keep connection alive. Set to 0 to disable pinging. Defaults to 21 seconds
- **ping_timeout** – Timeout in seconds if a `pong` (response to ping from server) is not received. The Stream is terminated as it is considered to be dead if no pong is received within the specified timeout. default: 20 seconds
- **ping_payload** – The option message to be sent with the ping. Better to leave it empty string.
- **skip_utf8_validation** – Whether to skip utf validation of messages. Defaults to True. Setting it to False may result in performance downgrade

Returns None

close_stream(**args*, ***kwargs*)

Close the websocket connection. Wait for thread to finish if running.

_authenticate()

Authenticates the client with the server using API key. Internal function, not meant to be called directly by users.

Returns None

_modify_sub(*symbols=None*, *action='subscribe'*, *_prefix='T.'*)

Internal Function to send subscribe or unsubscribe requests to websocket. You should prefer using the corresponding methods to subscribe or unsubscribe to streams.

Parameters

- **symbols** – The list of symbols to apply the actions to.

- **action** – Defaults to subscribe which subscribes to requested stream. Change to unsubscribe to remove an existing subscription.
- **_prefix** – prefix of the stream service. See *polygon.enums.StreamServicePrefix* for choices.

Returns None

subscribe_stock_trades(*symbols: Optional[list] = None*)
Stream real-time trades for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_trades(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_stock_quotes(*symbols: Optional[list] = None*)
Stream real-time Quotes for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_quotes(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_stock_minute_aggregates(*symbols: Optional[list] = None*)
Stream real-time minute aggregates for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_stock_second_aggregates(*symbols: Optional[list] = None*)
Stream real-time second aggregates for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_second_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_stock_limit_up_limit_down(*symbols: Optional[list] = None*)
Stream real-time LULD events for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_limit_up_limit_down(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_stock_imbalances(*symbols: Optional[list] = None*)
Stream real-time Imbalance Events for given stock ticker symbol(s).

Parameters **symbols** – A list of tickers. Default is * which subscribes to ALL tickers in the market

Returns None

unsubscribe_stock_imbalances(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

subscribe_option_trades(*symbols: Optional[list] = None*)
Stream real-time Options Trades for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

unsubscribe_option_trades(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Trades for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

subscribe_option_minute_aggregates(*symbols: Optional[list] = None*)
Stream real-time Options Minute Aggregates for given Options contract(s).

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL tickers in the market. you can pass **with or without** the prefix `O:`

Returns None

unsubscribe_option_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Minute aggregates for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

subscribe_option_second_aggregates(*symbols: Optional[list] = None*)
Stream real-time Options Second Aggregates for given Options contract(s).

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL tickers in the market. you can pass **with or without** the prefix `O:`

Returns None

unsubscribe_option_second_aggregates(*symbols: Optional[list] = None*)
Unsubscribe real-time Options Second Aggregates for given Options contract.

Parameters **symbols** – A list of symbols. Default is * which subscribes to ALL symbols in the market. you can pass **with or without** the prefix `O:`

Returns None

subscribe_forex_quotes(*symbols: Optional[list] = None*)
Stream real-time forex quotes for given forex pair(s).

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

unsubscribe_forex_quotes(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

subscribe_forex_minute_aggregates(*symbols: Optional[list] = None*)
Stream real-time forex Minute Aggregates for given forex pair(s).

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

unsubscribe_forex_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream service for the symbols specified. Defaults to all symbols.

Parameters **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

subscribe_crypto_trades(*symbols: Optional[list] = None*)
Stream real-time Trades for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

unsubscribe_crypto_trades(*symbols: Optional[list] = None*)
Unsubscribe real-time trades for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

subscribe_crypto_quotes(*symbols: Optional[list] = None*)
Stream real-time Quotes for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

unsubscribe_crypto_quotes(*symbols: Optional[list] = None*)
Unsubscribe real-time quotes for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

subscribe_crypto_minute_aggregates(*symbols: Optional[list] = None*)
Stream real-time Minute Aggregates for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

unsubscribe_crypto_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe real-time minute aggregates for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

subscribe_crypto_level2_book(*symbols: Optional[list] = None*)
Stream real-time level 2 book data for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

unsubscribe_crypto_level2_book(*symbols: Optional[list] = None*)
Unsubscribe real-time level 2 book data for given cryptocurrency pair(s).

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

static _default_on_msg(*_ws: websocket._app.WebSocketApp*, *msg*)
Default handler for message processing

Parameters **msg** – The message as received from the server

Returns None

static _default_on_close(*_ws: websocket._app.WebSocketApp*, *close_code*, *msg*)
THe default function to be called when stream is closed.

Parameters

- **close_code** – The close code as received from server
- **msg** – The close message as received from server

Returns

static _default_on_error(*_ws: websocket._app.WebSocketApp*, *error*, **args*)
Default function to be called when an error is encountered.

Parameters **error** – The exception object as supplied by the handler

Returns None

_default_on_open(*_ws: websocket._app.WebSocketApp*, **args*)
Default function to be called when stream client is initialized. Takes care of the authentication.

Parameters **args** – Any args supplied by the handler

Returns None

static _change_enum(*val*, *allowed_type=<class 'str'>*)

13.8 Async Streamer Client

class polygon.streaming.async_streaming.**AsyncStreamClient**(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *ping_interval: int = 20*, *ping_timeout: bool = 19*, *max_message_size: int = 1048576*, *max_memory_queue: int = 32*, *read_limit: int = 65536*, *write_limit: int = 65536*)

These docs are not meant for general users. These are library API references. The actual docs will be available on the index page when they are prepared.

Note that this is asyncio based stream client which is suitable for async applications. If you need to stream using an `callback` based stream client, see *Callback Streamer Client (Sync)*.

This class implements all the websocket endpoints. Note that you should always import names from top level. eg: `from polygon import AsyncStreamClient` or `import polygon` (which allows you to access all names easily)

Creating the client is as simple as: `client = AsyncStreamClient('MY_API_KEY', 'other_options')`

Once you have the client, you can call its methods to subscribe/unsubscribe to streams, change handlers and process messages. All methods have sane default values and almost everything can be customized.

Type Hinting tells you what data type a parameter is supposed to be. You should always use `enums` for most parameters to avoid supplying error prone values.

Take a look at the Official documentation to get an idea of the stream, data formatting for messages and related useful stuff.

__init__(*api_key: str*, *cluster*, *host='socket.polygon.io'*, *ping_interval: int = 20*, *ping_timeout: bool = 19*, *max_message_size: int = 1048576*, *max_memory_queue: int = 32*, *read_limit: int = 65536*, *write_limit: int = 65536*)

Initializes the stream client for async streaming Official Docs

Parameters

- **api_key** – Your API Key. Visit your dashboard to get yours.
- **cluster** – Which market/cluster to connect to. See *polygon.enums.StreamCluster* for choices. NEVER connect to the same cluster again if there is an existing stream connected to it. The existing connection would be dropped and new one will be established. You can have up to 4 concurrent streams connected to 4 different clusters.
- **host** – Host url to connect to. Default is real time. See *polygon.enums.StreamHost* for choices
- **ping_interval** – Send a `ping` to server every specified number of seconds to keep the connection alive. Defaults to 20 seconds. Setting to 0 disables pinging.
- **ping_timeout** – The number of seconds to wait after sending a ping for the response (`pong`). If no response is received from the server in those many seconds, stream is considered dead and exits with code `1011`. Defaults to 19 seconds.
- **max_message_size** – The max_size parameter enforces the maximum size for incoming messages in bytes. The default value is `1 MiB` (not MB). None disables the limit. If a message larger than the maximum size is received, `recv()` will raise `ConnectionClosedError` and the connection will be closed with code `1009`

- **max_memory_queue** – sets the maximum length of the queue that holds incoming messages. The default value is `32`. None disables the limit. Messages are added to an in-memory queue when they're received; then `recv()` pops from that queue
- **read_limit** – sets the high-water limit of the buffer for incoming bytes. The low-water limit is half the high-water limit. The default value is `64 KiB`, half of asyncio's default. Don't change if you are unsure of what it implies.
- **write_limit** – The write_limit argument sets the high-water limit of the buffer for outgoing bytes. The low-water limit is a quarter of the high-water limit. The default value is `64 KiB`, equal to asyncio's default. Don't change if you're unsure what it implies.

async login(*key: Optional[str] = None*)
Creates Websocket Socket client using the configuration and Logs to the stream with credentials. Primarily meant for internal uses. You shouldn't need to call this method manually as the streamer does it automatically behind the scenes

Returns None

async _send(*data: str*)
Internal function to send data to websocket server endpoint

Parameters **data** – The formatted data string to be sent.

Returns None

async _recv()
Internal function to receive messages from websocket server endpoint.

Returns The JSON decoded message data dictionary.

async handle_messages(*reconnect: bool = False*, *max_reconnection_attempts=5*, *reconnection_delay=5*)
The primary method to start the stream. Connects & Logs in by itself. Allows Reconnecting by simply altering a parameter (subscriptions are persisted across reconnected streams)

Parameters

- **reconnect** – If this is `False` (default), it simply awaits the next message and calls the appropriate handler. Uses the *_default_process_message()* if no handler was specified. You should use the statement inside a while loop in that case. Setting it to True creates an inner loop which traps disconnection errors except login failed due to invalid Key, and reconnects to the stream with the same subscriptions it had earlier before getting disconnected.
- **max_reconnection_attempts** – Determines how many times should the program attempt to reconnect in case of failed attempts. The Counter is reset as soon as a successful connection is re-established. Setting it to False disables the limit which is NOT recommended unless you know you got a situation. This value is ignored if `reconnect` is False (The default). Defaults to 5.
- **reconnection_delay** – Number of seconds to wait before attempting to reconnect after a failed reconnection attempt or a disconnection. This value is ignored if `reconnect` is False (the default). Defaults to 5.

Returns None

async reconnect() → tuple
Reconnects the stream. Existing subscriptions (ones before disconnections) are persisted and automatically re-subscribed when reconnection succeeds. All the handlers are also automatically restored. Returns a tuple based on success status. While this instance method is supposed to be used internally, it is possible to utilize this in your your custom attempts of reconnection implementation. Feel free to share your implementations with the community if you find success :)

Returns `(True, message)` if reconnection succeeds else `(False, message)`

async _default_process_message(*update*)
The default Handler for Message Streams which were NOT initialized with a handler function

Parameters **update** – The update message as received after decoding the message.

Returns None

_default_handlers_and_apis()
Assign default handler value to all stream setups. ONLY meant for internal use

async _modify_sub(*symbols: Optional[Union[str, list]]*, *action: str = 'subscribe'*, *_prefix: str = 'T.'*)
Internal Function to send subscribe or unsubscribe requests to websocket. You should prefer using the corresponding methods to subscribe or unsubscribe to streams.

Parameters

- **symbols** – The list of symbols to apply the actions to.
- **action** – Defaults to subscribe which subscribes to requested stream. Change to unsubscribe to remove an existing subscription.
- **_prefix** – prefix of the stream service. See *polygon.enums.StreamServicePrefix* for choices.

Returns None

async subscribe_stock_trades(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time trades for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL tickers.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_trades(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_stock_quotes(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time quotes for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL tickers.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_quotes(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_stock_minute_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_stock_second_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Seconds Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_second_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_stock_limit_up_limit_down(*symbols: Optional[list] = None*,
handler_function=None)
Get Real time LULD Events for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_limit_up_limit_down(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_stock_imbalances(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Imbalance Events for provided symbol(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker.
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_stock_imbalances(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied ticker symbols.

Parameters **symbols** – A list of tickers to unsubscribe from. Defaults to ALL tickers.

Returns None

async subscribe_option_trades(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time options trades for provided ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_option_trades(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

async subscribe_option_minute_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time options minute aggregates for given ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_option_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

async subscribe_option_second_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time options second aggregates for given ticker(s)

Parameters

- **symbols** – A list of tickers to subscribe to. Defaults to ALL ticker. You can specify with or without the prefix `O:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_option_second_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied option symbols.

Parameters **symbols** – A list of symbols to unsubscribe from. Defaults to ALL tickers. You can specify with or without the prefix `O:`

Returns None

async subscribe_forex_quotes(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Forex Quotes for provided symbol(s)

Parameters

- **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_forex_quotes(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied forex symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

async subscribe_forex_minute_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Forex Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of forex tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_forex_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied forex symbols.

Parameters **symbols** – A list of forex tickers. Default is * which unsubscribes to ALL tickers in the market. each Ticker must be in format: `from/to`. For example: `USD/CNH`. you can pass **with or without** the prefix `C:`

Returns None

async subscribe_crypto_trades(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Crypto Trades for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_crypto_trades(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

async subscribe_crypto_quotes(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Crypto Quotes for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_crypto_quotes(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

async subscribe_crypto_minute_aggregates(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Crypto Minute Aggregates for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_crypto_minute_aggregates(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

async subscribe_crypto_level2_book(*symbols: Optional[list] = None*, *handler_function=None*)
Get Real time Crypto Level 2 Book Data for provided symbol(s)

Parameters

- **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`
- **handler_function** – The function which you'd want to call to process messages received from this subscription. Defaults to None which uses the default process message function.

Returns None

async unsubscribe_crypto_level2_book(*symbols: Optional[list] = None*)
Unsubscribe from the stream for the supplied crypto symbols.

Parameters **symbols** – A list of Crypto tickers. Default is * which subscribes to ALL tickers in the market. each Ticker must be in format: `from-to`. For example: `BTC-USD`. you can pass symbols with or without the prefix `X:`

Returns None

async change_handler(*service_prefix*, *handler_function*)
Change your handler function for a service. Can be used to update handlers dynamically while stream is running.

Parameters

- **service_prefix** – The Prefix of the service you want to change handler for. see *polygon.enums.StreamServicePrefix* for choices.
- **handler_function** – The new handler function to assign for this service

Returns None

13.9 Enums Interface

class polygon.enums.**TickerMarketType**(*value*)
Market Types for method: `ReferenceClient.get_tickers()`

STOCKS = 'stocks'

OPTIONS = 'options'

FOREX = 'fx'

CRYPTO = 'crypto'

class polygon.enums.**TickerType**(*value*)
Ticker types for method: `ReferenceClient.get_tickers()`

CS = 'CS'

COMMON_STOCKS = 'CS'

ADRC = 'ADRC'

ADRP = 'ADRP'

ADRR = 'ADRR'

UNIT = 'UNIT'

RIGHT = 'RIGHT'

PFD = 'PFD'

FUND = 'FUND'

SP = 'SP'

WARRANT = 'WARRANT'

INDEX = 'INDEX'

ETF = 'ETF'

ETN = 'ETN'

class polygon.enums.TickerSortType(*value*)
Sort key for method: ReferenceClient.get_tickers()

TICKER = 'ticker'

NAME = 'name'

MARKET = 'market'

LOCALE = 'locale'

PRIMARY_EXCHANGE = 'primary_exchange'

TYPE = 'type'

ACTIVE = 'active'

CURRENCY_SYMBOL = 'currency_symbol'

CURRENCY_NAME = 'currency_name'

BASE_CURRENCY_SYMBOL = 'base_currency_symbol'

BASE_CURRENCY_NAME = 'base_currency_name'

CIK = 'cik'

COMPOSITE_FIGI = 'composite_figi'

SHARE_CLASS_FIGI = 'share_class_figi'

LAST_UPDATED_UTC = 'last_updated_utc'

DELISTED_UTC = 'delisted_utc'

class polygon.enums.SortOrder(*value*)
Order of sort. Ascending usually means oldest at the top. Descending usually means newest at the top. It is recommended to ensure the behavior in the corresponding function's docs. This enum can be used by any method accepting Sort order values.

ASCENDING = 'asc'

ASC = 'asc'

DESCENDING = 'desc'

DESC = 'desc'

class polygon.enums.TickerTypeAssetClass(*value*)
Asset Class for method: ReferenceClient.get_ticker_types_v3()

STOCKS = 'stocks'

OPTIONS = 'options'

FOREX = 'fx'

CRYPTO = 'crypto'

class polygon.enums.**TickerNewsSort**(*value*)
Sort key for method: `ReferenceClient.get_ticker_news()`

PUBLISHED_UTC = 'published_utc'

ALL = None

class polygon.enums.**StockReportType**(*value*)
Type of report for method: `ReferenceClient.get_stock_financials()`

YEAR = 'Y'

Y = 'Y'

YA = 'YA'

YEAR_ANNUALIZED = 'YA'

Q = 'Q'

QUARTER = 'Q'

QA = 'QA'

QUARTER_ANNUALIZED = 'QA'

T = 'T'

TRAILING_TWELVE_MONTHS = 'T'

TA = 'TA'

TRAILING_TWELVE_MONTHS_ANNUALIZED = 'TA'

class polygon.enums.**StockFinancialsSortType**(*value*)
Direction to use for sorting report for method: `ReferenceClient.get_stock_financials()`

REPORT_PERIOD = 'reportPeriod'

REVERSE_REPORT_PERIOD = '-reportPeriod'

CALENDAR_DATE = 'calendarDate'

REVERSE_CALENDAR_DATE = '-calendarDate'

class polygon.enums.**StockFinancialsTimeframe**(*value*)
Query by timeframe. Annual financials originate from 10-K filings, and quarterly financials originate from 10-Q filings. Note: Most companies do not file quarterly reports for Q4 and instead include those financials in their annual report, so some companies my not return quarterly financials for Q4 for method: `ReferenceClient.get_stock_financials_vx()`

ANNUAL = 'annual'

QUARTERLY = 'quarterly'

class polygon.enums.**StockFinancialsSortKey**(*value*)
Sort field for method: `ReferenceClient.get_stock_financials_vx()`

FILLING_DATE = 'filling_date'

PERIOD_OF_REPORT_DATE = 'period_of_report_date'

class polygon.enums.**ConditionMappingTickType**(*value*)
Tick Type for method: `ReferenceClient.get_condition_mappings()`

TRADES = 'trades'

QUOTES = 'quotes'

class polygon.enums.**ConditionsDataType**(*value*)
Type of data for method: `ReferenceClient.get_conditions()`

TRADE = 'trade'

BBO = 'bbo'

NBBO = 'nbbo'

class polygon.enums.**ConditionsSIP**(*value*)
SIP for method: `ReferenceClient.get_conditions()`

CTA = 'CTA'

UTP = 'UTP'

OPRA = 'OPRA'

class polygon.enums.**ConditionsSortKey**(*value*)
Sort key for method: `ReferenceClient.get_conditions()`

ASSET_CLASS = 'asset_class'

ID = 'id'

TYPE = 'type'

NAME = 'name'

DATA_TYPES = 'data_types'

LEGACY = 'legacy'

class polygon.enums.**AssetClass**(*value*)
Asset Class for methods: `ReferenceClient.get_exchanges_v3()` and `ReferenceClient.get_conditions()` and wherever needed.

STOCKS = 'stocks'

OPTIONS = 'options'

FOREX = 'fx'

CRYPTO = 'crypto'

class polygon.enums.**Locale**(*value*)
Locale name``

US = 'us'

GLOBAL = 'global'

class polygon.enums.**SnapshotDirection**
Direction to be supplied to the SnapShot - Gainers and Losers APIs on Stocks, Forex and Crypto endpoints

GAINERS = 'gainers'

GAIN = 'gainers'

LOSERS = 'losers'

LOSE = 'losers'

class polygon.enums.**StreamCluster**(*value*)
The cluster to connect to. To be used for both callback and async stream client. NEVER connect to the same cluster again if there is an existing stream connected to it. The existing connection would be dropped and new one will be established. You can have up to 4 concurrent streams connected to 4 different clusters.

STOCKS = 'stocks'

OPTIONS = 'options'

FOREX = 'forex'

CRYPTO = 'crypto'

class polygon.enums.OptionsContractType(*value*)
Contract Type for method: ReferenceClient.get_options_contracts()

CALL = ('call',)

PUT = 'put'

OTHER = 'other'

class polygon.enums.OptionsContractsSortType(*value*)
Sort field used for ordering for method: ReferenceClient.get_options_contracts()

TICKER = 'ticker'

UNDERLYING_TICKER = 'underlying_ticker'

EXPIRATION_DATE = 'expiration_date'

STRIKE_PRICE = 'strike_price'

class polygon.enums.OptionTradesSort(*value*)
Sort field used for ordering option trades. Used for method: OptionsClient.get_trades

TIMESTAMP = 'timestamp'

class polygon.enums.StreamHost(*value*)
Host to be used for stream connections. WHY on earth would you use delayed if you're paying for polygon??

REAL_TIME = 'socket.polygon.io'

DELAYED = 'delayed.polygon.io'

class polygon.enums.StreamServicePrefix(*value*)
Service Prefix for Stream endpoints. To be used for method: AsyncStreamClient.async change_handler()

STOCK_TRADES = 'T'

STOCK_QUOTES = 'Q'

STOCK_MINUTE_AGGREGATES = 'AM'

STOCK_SECOND_AGGREGATES = 'A'

STOCK_LULD = 'LULD'

STOCK_IMBALANCES = 'NOI'

FOREX_QUOTES = 'C'

FOREX_MINUTE_AGGREGATES = 'CA'

CRYPTO_TRADES = 'XT'

CRYPTO_QUOTES = 'XQ'

CRYPTO_LEVEL2 = 'XL2'

CRYPTO_MINUTE_AGGREGATES = 'XA'

STATUS = 'status'

OPTION_TRADES = 'T'

OPTION_MINUTE_AGGREGATES = 'AM'

OPTION_SECOND_AGGREGATES = 'A'

class polygon.enums.**Timespan**(*value*)

The timespan values. Usually meant for aggregates endpoints. It is best to consult the relevant docs before using any value on an endpoint.

MINUTE = 'minute'

HOUR = 'hour'

DAY = 'day'

WEEK = 'week'

MONTH = 'month'

QUARTER = 'quarter'

YEAR = 'year'

CHAPTER

FOURTEEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

Symbols

A

H

I

L

M

N

O

P

Q

R

S

T

U

W

Y