tda-api

Jul 04, 2020

Contents:

1 Getting Started **3**
- 1.1 TD Ameritrade API Access . . . 3
- 1.2 Installing `tda-api` . . . 4
- 1.3 Getting Help . . . 4

2 Authentication and Client Creation **5**
- 2.1 OAuth Refresher . . . 5
- 2.2 Fetching a Token and Creating a Client . . . 6
- 2.3 Troubleshooting . . . 7

3 HTTP Client **9**
- 3.1 Calling Conventions . . . 9
- 3.2 Return Values . . . 10
- 3.3 Creating a New Client . . . 10
- 3.4 Orders . . . 10
- 3.5 Account Info . . . 12
- 3.6 Instrument Info . . . 13
- 3.7 Option Chain . . . 13
- 3.8 Price History . . . 15
- 3.9 Current Quotes . . . 17
- 3.10 Other Endpoints . . . 17

4 Streaming Client **21**
- 4.1 Use Overview . . . 22
- 4.2 Enabling Real-Time Data Access . . . 25
- 4.3 OHLCV Charts . . . 25
- 4.4 Level One Quotes . . . 27
- 4.5 Level Two Order Book . . . 38
- 4.6 Time of Sale . . . 39
- 4.7 News Headlines . . . 41
- 4.8 Account Activity . . . 42

5 Order Templates **43**
- 5.1 Using These Templates . . . 43
- 5.2 Equity Templates . . . 44
- 5.3 Options Templates . . . 44
- 5.4 Utility Methods . . . 46

5.5 What happened to `EquityOrderBuilder`? . . . 46

6 `OrderBuilder` Reference **49**
6.1 Optional: Order Specification Introduction . . . 49
6.2 `OrderBuilder` Reference . . . 52

7 Utilities **61**
7.1 Get the Most Recent Order . . . 61

8 Example Application **63**

9 Getting Help **65**
9.1 Enable Logging . . . 65
9.2 Gather Logs For Your Bug Report . . . 65
9.3 Submit Your Ticket . . . 66

10 Contributing to `tda-api` **67**
10.1 Setting up the Dev Environment . . . 67
10.2 Development Guidelines . . . 68

11 Indices and tables **69**

Python Module Index **71**

Index **73**

CHAPTER 1

Getting Started

Welcome to `tda-api`! Read this page to learn how to install and configure your first TD Ameritrade Python application.

1.1 TD Ameritrade API Access

All API calls to the TD Ameritrade API require an API key. Before we do anything with `tda-api`, you'll need to create a developer account with TD Ameritrade and register an application. By the end of this section, you'll have accomplished the three prerequisites for using `tda-api`:

1. Create an application.
2. Choose and save the callback URL (important for authenticating).
3. Receive an API key.

You can create a developer account here. The instructions from here on out assume you're logged in, so make sure you log into the developer site after you've created your account.

Next, you'll want to create an application. The app name and purpose aren't particularly important right now, but the callback URL is. In a nutshell, the OAuth login flow that TD Ameritrade uses works by opening a TD Ameritrade login page, securely collecting credentials on their domain, and then sending an HTTP request to the callback URL with the token in the URL query.

How you use to choose your callback URL depends on whether and how you plan on distributing your app. If you're writing an app for your own personal use, and plan to run entirely on your own machine, use `https://localhost`. If you plan on running on a server and having users send requests to you, use a URL you own, such as a dedicated endpoint on your domain.

Once your app is created and approved, you will receive your API key, also known as the Client ID. This will be visible in TDA's app listing page. Record this key, since it is necessary to access most API endpoints.

1.2 Installing `tda-api`

This section outlines the installation process for client users. For developers, check out *Contributing to tda-api*.

The recommended method of installing `tda-api` is using `pip` from PyPi in a virtualenv. First create a virtualenv in your project diretory. Here we assume your virtualenv is called `my-venv`:

```
pip install virtualenv
virtualenv -v my-venv
source my-venv/bin/activate
```

You are now ready to install `tda-api`:

```
pip install tda-api
```

That's it! You're done! You can verify the install succeeded by importing the package:

```
import tda
```

If this succeeded, you're ready to move on to *Authentication and Client Creation*.

Note that if you are using a virtual environment and switch to a new terminal your virtual environment will not be active in the new terminal, and you need to run the activate command again. If you want to disable the loaded virtual environment in the same terminal window, use the command:

```
deactivate
```

1.3 Getting Help

If you are ever stuck, feel free to join our Discord server to ask questions, get advice, and chat with like-minded people. If you feel you've found a bug, you can *fill out a bug report*.

CHAPTER 2

Authentication and Client Creation

By now, you should have followed the instructions in *Getting Started* and are ready to start making API calls. Read this page to learn how to get over the last remaining hurdle: OAuth authentication.

Before we begin, however, note that this guide is meant to users who want to run applications on their own machines, without distributing them to others. If you plan on distributing your app, or if you plan on running it on a server and allowing access to other users, this login flow is not for you.

2.1 OAuth Refresher

This section is purely for the curious. If you already understand OAuth (wow, congrats) or if you don't care and just want to use this package as fast as possible, feel free to skip this section. If you encounter any weird behavior, this section may help you understand that's going on.

Webapp authentication is a complex beast. The OAuth protocol was created to allow applications to access one anothers' APIs securely and with the minimum level of trust possible. A full treatise on this topic is well beyond the scope of this guide, but in order to alleviate some of the confusion and complexity that seems to surround this part of the API, let's give a quick explanation of how OAuth works in the context of TD Ameritrade's API.

The first thing to understand is that the OAuth webapp flow was created to allow client-side applications consisting of a webapp frontend and a remotely hosted backend to interact with a third party API. Unlike the backend application flow, in which the remotely hosted backend has a secret which allows it to access the API on its own behalf, the webapp flow allows either the webapp frontend or the remotely host backend to access the API *on behalf of its users*.

If you've ever installed a GitHub, Facebook, Twitter, GMail, etc. app, you've seen this flow. You click on the "install" link, a login window pops up, you enter your password, and you're presented with a page that asks whether you want to grant the app access to your account.

Here's what's happening under the hood. The window that pops up is the authentication URL, which opens a login page for the target API. The aim is to allow the user to input their username and password without the webapp frontend or the remotely hosted backend seeing it. On web browsers, this is accomplished using the browser's refusal to send credentials from one domain to another.

Once login here is successful, the API replies with a redirect to a URL that the remotely hosted backend controls. This is the callback URL. This redirect will contain a code which securely identifies the user to the API, embedded in the query of the request.

You might think that code is enough to access the API, and it would be if the API author were willing to sacrifice long-term security. The exact reasons why it doesn't work involve some deep security topics like robustness against replay attacks and session duration limitation, but we'll skip them here.

This code is useful only for fetching a token from the authentication endpoint. *This token* is what we want: a secure secret which the client can use to access API endpoints, and can be refreshed over time.

If you've gotten this far and your head isn't spinning, you haven't been paying attention. Security-sensitive protocols can be very complicated, and you should **never** build your own implementation. Fortunately there exist very robust implementations of this flow, and `tda-api`'s authentication module makes using them easy.

2.2 Fetching a Token and Creating a Client

`tda-api` provides an easy implementation of the client-side login flow in the `auth` package. It uses a selenium webdriver to open the TD Ameritrade authentication URL, take your login credentials, catch the post-login redirect, and fetch a reusable token. It returns a fully-configured *HTTP Client*, ready to send API calls. It also handles token refreshing, and writes updated tokens to the token file.

These functions are webdriver-agnostic, meaning you can use whatever webdriver-supported browser you can available on your system. You can find information about available webdriver on the Selenium documentation.

`tda.auth.`**`client_from_login_flow`**(*webdriver*, *api_key*, *redirect_url*, *token_path*, *redirect_wait_time_seconds=0.1*)

Uses the webdriver to perform an OAuth webapp login flow and creates a client wrapped around the resulting token. The client will be configured to refresh the token as necessary, writing each updated version to `token_path`.

Parameters

- **`webdriver`** – selenium webdriver which will be used to perform the login flow.
- **`api_key`** – Your TD Ameritrade application's API key, also known as the client ID.
- **`redirect_url`** – Your TD Ameritrade application's redirect URL. Note this must *exactly* match the value you've entered in your application configuration, otherwise login will fail with a security error.
- **`token_path`** – Path to which the new token will be written. If the token file already exists, it will be overwritten with a new one. Updated tokens will be written to this path as well.

Once you have a token written on disk, you can reuse it without going through the login flow again.

`tda.auth.`**`client_from_token_file`**(*token_path*, *api_key*)

Returns a session from an existing token file. The session will perform an auth refresh as needed. It will also update the token on disk whenever appropriate.

Parameters

- **`token_path`** – Path to an existing token. Updated tokens will be written to this path. If you do not yet have a token, use *`client_from_login_flow()`* or *`easy_client()`* to create one.
- **`api_key`** – Your TD Ameritrade application's API key, also known as the client ID.

The following is a convenient wrapper around these two methods, calling each when appropriate:

tda.auth.**easy_client**(*api_key*, *redirect_uri*, *token_path*, *webdriver_func=None*)

Convenient wrapper around *client_from_login_flow()* and *client_from_token_file()*. If token_path exists, loads the token from it. Otherwise open a login flow to fetch a new token. Returns a client configured to refresh the token to token_path.

Reminder: You should never create the token file yourself or modify it in any way. If token_path refers to an existing file, this method will assume that file is valid token and will attempt to parse it.

Parameters

- **api_key** – Your TD Ameritrade application's API key, also known as the client ID.
- **redirect_url** – Your TD Ameritrade application's redirect URL. Note this must *exactly* match the value you've entered in your application configuration, otherwise login will fail with a security error.
- **token_path** – Path that new token will be read from and written to. If If this file exists, this method will assume it's valid and will attempt to parse it as a token. If it does not, this method will create a new one using *client_from_login_flow()*. Updated tokens will be written to this path as well.
- **webdriver_func** – Function that returns a webdriver for use in fetching a new token. Will only be called if the token file cannot be found.

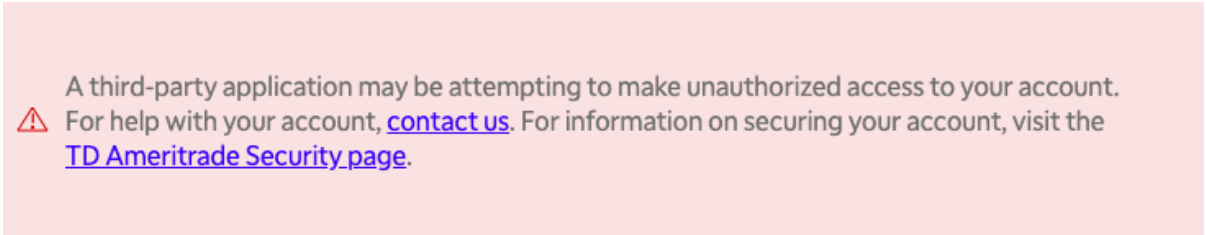
2.3 Troubleshooting

As simple as it seems, this process is complex and mistakes are easy to make. This section outlines some of the more common issues you might encounter. If you find yourself dealing with something that isn't listed here, or if you try the suggested remedies and are still seeing issues, see the *Getting Help* page. You can also join our Discord server to ask questions.

2.3.1 "A third-party application may be attempting to make unauthorized access to your account"

One attack on improperly implemented OAuth login flows involves tricking a user into submitting their credentials for a real app and then redirecting to a malicious web server (remember the GET request to the redirect URI contains all credentials required to access the user's account). This is especially pernicious because from the user's perspective, they see a real login window and probably never realize they've been sent to a malicious server, especially if the landing page is designed to resemble the target API's landing page.

TD Ameritrade correctly prevents this attack by refusing to allow a login if the redirect URI does not **exactly** match the redirect URI with which the app is configured. If you make *any* mistake in setting your API key or redirect URI, you'll see this instead of a login page:



If this happens, you almost certainly copied your API key or redirect URI incorrectly. Go back to your application list and copy-paste it again.

2.3.2 `tda-api` Hangs After Successful Login

After opening the login window, `tda-api` loops and waits until the webdriver's current URL starts with the given redirect URI:

```
callback_url = ''
while not callback_url.startswith(redirect_url):
    callback_url = webdriver.current_url
    time.sleep(redirect_wait_time_seconds)
```

Usually, it would be impossible for a successful post-login callback to not start with the callback URI, but there's one major exception: when the callback URI starts with `http`. Behavior varies by browser and app configuration, but a callback URI starting with `http` can sometimes be redirected to one starting with `https`, in which case `tda-api` will never notice the redirect.

If this is happening to you, consider changing your callback URI to use `https` instead of `http`. Not only will it make your life easier here, but it is *extremely* bad practice to send credentials like this over an unencrypted channel like that provided by `http`.

2.3.3 Token Parsing Failures

`tda-api` handles creating and refreshing tokens. Simply put, *the user should never create or modify the token file*. If you are experiencing parse errors when accessing the token file or getting exceptions when accessing it, it's probably because you created it yourself or modified it. If you're experiencing token parsing issues, remember that:

1. You should never create the token file yourself. If you don't already have a token, you should pass a nonexistent file path to *`client_from_login_flow()`* or *`easy_client()`*. If the file already exists, these methods assume it's a valid token file. If the file does not exist, they will go through the login flow to create one.
2. You should never modify the token file. The token file is automatically managed by `tda-api`, and modifying it will almost certainly break it.
3. You should never share the token file. If the token file is shared between applications, one of them will beat the other to refreshing, locking the slower one out of using `tda-api`.

If you didn't do any of this and are still seeing issues using a token file that you're confident is valid, please file a ticket. Just remember, **never share your token file, not even with `tda-api` developers**. Sharing the token file is as dangerous as sharing your TD Ameritrade username and password.

CHAPTER 3

HTTP Client

A naive, unopinionated wrapper around the TD Ameritrade HTTP API. This client provides access to all endpoints of the API in as easy and direct a way as possible. For example, here is how you can fetch the past 20 years of data for Apple stock:

```
from tda.auth import easy_client
from tda.client import Client

c = easy_client(
        api_key='APIKEY',
        redirect_uri='https://localhost',
        token_path='/tmp/token.pickle')

resp = c.get_price_history('AAPL',
        period_type=Client.PriceHistory.PeriodType.YEAR,
        period=Client.PriceHistory.Period.TWENTY_YEARS,
        frequency_type=Client.PriceHistory.FrequencyType.DAILY,
        frequency=Client.PriceHistory.Frequency.DAILY)
assert resp.ok
history = resp.json()
```

Note we we create a new client using the `auth` package as described in *Authentication and Client Creation*. Creating a client directly is possible, but not recommended.

3.1 Calling Conventions

Function parameters are categorized as either required or optional. Required parameters, such as `'AAPL'` in the example above, are passed as positional arguments. Optional parameters, like `period_type` and the rest, are passed as keyword arguments.

Parameters which have special values recognized by the API are represented by Python enums. This is because the API rejects requests which pass unrecognized values, and this enum wrapping is provided as a convenient mechanism to avoid consternation caused by accidentally passing an unrecognized value.

By default, passing values other than the required enums will raise a `ValueError`. If you believe the API accepts a value that isn't supported here, you can use `set_enforce_enums` to disable this behavior at your own risk. If you *do* find a supported value that isn't listed here, please open an issue describing it or submit a PR adding the new functionality.

3.2 Return Values

All methods return a response object generated under the hood by the requests module. For a full listing of what's possible, read that module's documentation. Most if not all users can simply use the following pattern:

```
r = client.some_endpoint()
assert r.ok, r.raise_for_status()
data = r.json()
```

The API indicates errors using the response status code, and this pattern will raise the appropriate exception if the response is not a success. The data can be fetched by calling the `.json()` method.

This data will be pure python data structures which can be directly accessed. You can also use your favorite data analysis library's dataframe format using the appropriate library. For instance you can create a pandas dataframe using its conversion method.

Note: Because the author has no relationship whatsoever with TD Ameritrade, this document makes no effort to describe the structure of the returned JSON objects. TDA might change them at any time, at which point this document will become silently out of date. Instead, each of the methods described below contains a link to the official documentation. For endpoints that return meaningful JSON objects, it includes a JSON schema which describes the return value. Please use that documentation or your own experimentation when figuring out how to use the data returned by this API.

3.3 Creating a New Client

99.9% of users should not create their own clients, and should instead follow the instructions outlined in *Authentication and Client Creation*. For those brave enough to build their own, the constructor looks like this:

`Client.__init__`(*api_key*, *session*, *, *enforce_enums=True*)

Create a new client with the given API key and session. Set *enforce_enums=False* to disable strict input type checking.

3.4 Orders

3.4.1 Placing New Orders

Placing new orders can be a complicated task. The *`Client.place_order()`* method is used to create all orders, from equities to options. The precise order type is defined by a complex order spec. TDA provides some example order specs to illustrate the process and provides a schema in the place order documentation, but beyond that we're on our own.

`tda-api` includes some helpers, described in *Order Templates*, which provide an incomplete utility for creating various order types. While it only scratches the surface of what's possible, we encourage you to use that module instead of creating your own order specs.

Client.**place_order**(*account_id*, *order_spec*)
Place an order for a specific account. If order creation was successful, the response will contain the ID of the generated order. See *tda.utils.Utils.extract_order_id()* for more details.

Official documentation.

3.4.2 Accessing Existing Orders

Client.**get_orders_by_path**(*account_id*, *, *max_results=None*, *from_entered_datetime=None*, *to_entered_datetime=None*, *status=None*, *statuses=None*)
Orders for a specific account. At most one of `status` and `statuses` may be set. Official documentation.

Parameters

- **max_results** – The maximum number of orders to retrieve.
- **from_entered_datetime** – Specifies that no orders entered before this time should be returned. Date must be within 60 days from today's date. `toEnteredTime` must also be set.
- **to_entered_datetime** – Specifies that no orders entered after this time should be returned. `fromEnteredTime` must also be set.
- **status** – Restrict query to orders with this status. See *Order.Status* for options.
- **statuses** – Restrict query to orders with any of these statuses. See *Order.Status* for options.

Client.**get_orders_by_query**(*, *max_results=None*, *from_entered_datetime=None*, *to_entered_datetime=None*, *status=None*, *statuses=None*)
Orders for all linked accounts. At most one of `status` and `statuses` may be set. Official documentation.

Parameters

- **max_results** – The maximum number of orders to retrieve.
- **from_entered_datetime** – Specifies that no orders entered before this time should be returned. Date must be within 60 days from today's date. `toEnteredTime` must also be set.
- **to_entered_datetime** – Specifies that no orders entered after this time should be returned. `fromEnteredTime` must also be set.
- **status** – Restrict query to orders with this status. See *Order.Status* for options.
- **statuses** – Restrict query to orders with any of these statuses. See *Order.Status* for options.

Client.**get_order**(*order_id*, *account_id*)
Get a specific order for a specific account by its order ID. Official documentation.

class tda.client.Client.**Order**

class Status
Order statuses passed to *get_orders_by_path()* and *get_orders_by_query()*

ACCEPTED = 'ACCEPTED'

AWAITING_CONDITION = 'AWAITING_CONDITION'

AWAITING_MANUAL_REVIEW = 'AWAITING_MANUAL_REVIEW'

AWAITING_PARENT_ORDER = 'AWAITING_PARENT_ORDER'

```
AWAITING_UR_OUR = 'AWAITING_UR_OUR'
CANCELLED = 'CANCELLED'
EXPIRED = 'EXPIRED'
FILLED = 'FILLED'
PENDING_ACTIVATION = 'PENDING_ACTIVATION'
PENDING_CANCEL = 'PENDING_CANCEL'
PENDING_REPLACE = 'PENDING_REPLACE'
QUEUED = 'QUEUED'
REJECTED = 'REJECTED'
REPLACED = 'REPLACED'
WORKING = 'WORKING'
```

3.4.3 Editing Existing Orders

Endpoints for canceling and replacing existing orders. Annoyingly, while these endpoints require an order ID, it seems that when placing new orders the API does not return any metadata about the new order. As a result, if you want to cancel or replace an order after you've created it, you must search for it using the methods described in *Accessing Existing Orders*.

`Client.`**`cancel_order`**(*order_id*, *account_id*)
Cancel a specific order for a specific account. Official documentation.

`Client.`**`replace_order`**(*account_id*, *order_id*, *order_spec*)
Replace an existing order for an account. The existing order will be replaced by the new order. Once replaced, the old order will be canceled and a new order will be created. Official documentation.

3.5 Account Info

These methods provide access to useful information about accounts. An incomplete list of the most interesting bits:

- Account balances, including available trading balance
- Positions
- Order history

See the official documentation for each method for a complete response schema.

`Client.`**`get_account`**(*account_id*, *, *fields=None*)
Account balances, positions, and orders for a specific account. Official documentation.

Parameters **`fields`** – Balances displayed by default, additional fields can be added here by adding values from *`Account.Fields`*.

`Client.`**`get_accounts`**(*, *fields=None*)
Account balances, positions, and orders for all linked accounts. Official documentation.

Parameters **`fields`** – Balances displayed by default, additional fields can be added here by adding values from *`Account.Fields`*.

`class` `tda.client.Client.`**`Account`**

class Fields

Account fields passed to *get_account()* and *get_accounts()*

ORDERS = 'orders'

POSITIONS = 'positions'

3.6 Instrument Info

Note: symbol fundamentals (P/E ratios, number of shares outstanding, dividend yield, etc.) is available using the `Instrument.Projection.FUNDAMENTAL` projection.

`Client.`**search_instruments**(*symbols*, *projection*)

Search or retrieve instrument data, including fundamental data. Official documentation.

Parameters **projection** – Query type. See *Instrument.Projection* for options.

`Client.`**get_instrument**(*cusip*)

Get an instrument by CUSIP. Official documentation.

class `tda.client.Client.`**Instrument**

class Projection

Search query type for *search_instruments()*. See the official documentation for details on the semantics of each.

DESC_REGEX = 'desc-regex'

DESC_SEARCH = 'desc-search'

FUNDAMENTAL = 'fundamental'

SYMBOL_REGEX = 'symbol-regex'

SYMBOL_SEARCH = 'symbol-search'

3.7 Option Chain

Unfortunately, option chains are well beyond the ability of your humble author. You are encouraged to read the official API documentation to learn more.

If you *are* knowledgeable enough to write something more substantive here, please follow the instructions in *Contributing to tda-api* to send in a patch.

`Client.`**get_option_chain**(*symbol*, ***, *contract_type=None*, *strike_count=None*, *include_quotes=None*, *strategy=None*, *interval=None*, *strike=None*, *strike_range=None*, *strike_from_date=None*, *strike_to_date=None*, *volatility=None*, *underlying_price=None*, *interest_rate=None*, *days_to_expiration=None*, *exp_month=None*, *option_type=None*)

Get option chain for an optionable Symbol. Official documentation.

Parameters

- **contract_type** – Type of contracts to return in the chain. See *Options.ContractType* for choices.
- **strike_count** – The number of strikes to return above and below the at-the-money price.
- **include_quotes** – Include quotes for options in the option chain?

- **strategy** – If passed, returns a Strategy Chain. See *Options.Strategy* for choices.
- **interval** – Strike interval for spread strategy chains (see `strategy` param).
- **strike** – Return options only at this strike price.
- **strike_range** – Return options for the given range. See *Options.StrikeRange* for choices.
- **strike_from_date** – Only return expirations after this date. For strategies, expiration refers to the nearest term expiration in the strategy. Accepts `datetime.date` and `datetime.datetime`.
- **strike_to_date** – Only return expirations before this date. For strategies, expiration refers to the nearest term expiration in the strategy. Accepts `datetime.date` and `datetime.datetime`.
- **volatility** – Volatility to use in calculations. Applies only to `ANALYTICAL` strategy chains.
- **underlying_price** – Underlying price to use in calculations. Applies only to `ANALYTICAL` strategy chains.
- **interest_rate** – Interest rate to use in calculations. Applies only to `ANALYTICAL` strategy chains.
- **days_to_expiration** – Days to expiration to use in calculations. Applies only to `ANALYTICAL` strategy chains
- **exp_month** – Return only options expiring in the specified month. See *Options.ExpirationMonth* for choices.
- **option_type** – Types of options to return. See *Options.Type* for choices.

class `tda.client.Client.`**Options**

class ContractType
An enumeration.

ALL = 'ALL'

CALL = 'CALL'

PUT = 'PUT'

class ExpirationMonth
An enumeration.

APRIL = 'APR'

AUGUST = 'AUG'

DECEMBER = 'DEC'

FEBRUARY = 'FEB'

JANUARY = 'JAN'

JULY = 'JUL'

JUN = 'JUN'

MARCH = 'MAR'

MAY = 'MAY'

NOVEMBER = 'NOV'

OCTOBER = 'OCT'

SEPTEMBER = 'SEP'

class Strategy
An enumeration.

ANALYTICAL = 'ANALYTICAL'

BUTTERFLY = 'BUTTERFLY'

CALENDAR = 'CALENDAR'

COLLAR = 'COLLAR'

CONDOR = 'CONDOR'

COVERED = 'COVERED'

DIAGONAL = 'DIAGONAL'

ROLL = 'ROLL'

SINGLE = 'SINGLE'

STRADDLE = 'STRADDLE'

STRANGLE = 'STRANGLE'

VERTICAL = 'VERTICAL'

class StrikeRange
An enumeration.

ALL = 'ALL'

IN_THE_MONEY = 'ITM'

NEAR_THE_MONEY = 'NTM'

OUT_OF_THE_MONEY = 'OTM'

STRIKES_ABOVE_MARKET = 'SAK'

STRIKES_BELOW_MARKET = 'SBK'

STRIKES_NEAR_MARKET = 'SNK'

class Type
An enumeration.

ALL = 'ALL'

NON_STANDARD = 'NS'

STANDARD = 'S'

3.8 Price History

Fetching price history is somewhat complicated due to the fact that only certain combinations of parameters are valid. To avoid accidentally making it impossible to send valid requests, this method performs no validation on its parameters. If you are receiving empty requests or other weird return values, see the official documentation for more details.

Client.**get_price_history**(*symbol*, ***, *period_type=None*, *period=None*, *frequency_type=None*, *frequency=None*, *start_datetime=None*, *end_datetime=None*, *need_extended_hours_data=None*)

Get price history for a symbol. Official documentation.

Parameters

- **period_type** – The type of period to show.
- **period** – The number of periods to show. Should not be provided if `start_datetime` and `end_datetime`.
- **frequency_type** – The type of frequency with which a new candle is formed.
- **frequency** – The number of the frequencyType to be included in each candle.
- **start_datetime** – End date. Default is previous trading day.
- **end_datetime** – Start date.
- **need_extended_hours_data** – If true, return extended hours data. Otherwise return regular market hours only.

class tda.client.Client.**PriceHistory**

class Frequency

An enumeration.

DAILY = 1

EVERY_FIFTEEN_MINUTES = 15

EVERY_FIVE_MINUTES = 5

EVERY_MINUTE = 1

EVERY_TEN_MINUTES = 10

EVERY_THIRTY_MINUTES = 30

MONTHLY = 1

WEEKLY = 1

class FrequencyType

An enumeration.

DAILY = 'daily'

MINUTE = 'minute'

MONTHLY = 'monthly'

WEEKLY = 'weekly'

class Period

An enumeration.

FIFTEEN_YEARS = 15

FIVE_DAYS = 5

FIVE_YEARS = 5

FOUR_DAYS = 4

ONE_DAY = 1

```
ONE_MONTH = 1
ONE_YEAR = 1
SIX_MONTHS = 6
TEN_DAYS = 10
TEN_YEARS = 10
THREE_DAYS = 3
THREE_MONTHS = 3
THREE_YEARS = 3
TWENTY_YEARS = 20
TWO_DAYS = 2
TWO_MONTHS = 2
TWO_YEARS = 2
YEAR_TO_DATE = 1
```

class PeriodType
An enumeration.

```
DAY = 'day'
MONTH = 'month'
YEAR = 'year'
YEAR_TO_DATE = 'ytd'
```

3.9 Current Quotes

Client.**get_quote**(*symbol*)
Get quote for a symbol. Note due to limitations in URL encoding, this method is not recommended for instruments with symbols symbols containing non-alphanumeric characters, for example as futures like `/ES`. To get quotes for those symbols, use *Client.get_quotes()*.

Official documentation.

Client.**get_quotes**(*symbols*)
Get quote for a symbol. This method supports all symbols, including those containing non-alphanumeric characters like `/ES`. Official documentation.

3.10 Other Endpoints

Note If your account limited to delayed quotes, these quotes will also be delayed.

3.10.1 Transaction History

Client.**get_transaction**(*account_id*, *transaction_id*)
Transaction for a specific account. Official documentation.

Client.**get_transactions**(*account_id*, *, *transaction_type=None*, *symbol=None*, *start_date=None*, *end_date=None*)

Transaction for a specific account. Official documentation.

Parameters

- **transaction_type** – Only transactions with the specified type will be returned.
- **symbol** – Only transactions with the specified symbol will be returned.
- **start_date** – Only transactions after this date will be returned. Note the maximum date range is one year. Accepts `datetime.date` and `datetime.datetime`.
- **end_date** – Only transactions before this date will be returned Note the maximum date range is one year. Accepts `datetime.date` and `datetime.datetime`.

class tda.client.Client.**Transactions**

class TransactionType

An enumeration.

ADVISORY_FEES = 'ADVISORY_FEES'

ALL = 'ALL'

BUY_ONLY = 'BUY_ONLY'

CASH_IN_OR_CASH_OUT = 'CASH_IN_OR_CASH_OUT'

CHECKING = 'CHECKING'

DIVIDEND = 'DIVIDEND'

INTEREST = 'INTEREST'

OTHER = 'OTHER'

SELL_ONLY = 'SELL_ONLY'

TRADE = 'TRADE'

3.10.2 Saved Orders

Client.**create_saved_order**(*account_id*, *order_spec*)

Save an order for a specific account. Official documentation.

Client.**delete_saved_order**(*account_id*, *order_id*)

Delete a specific saved order for a specific account. Official documentation.

Client.**get_saved_order**(*account_id*, *order_id*)

Specific saved order by its ID, for a specific account. Official documentation.

Client.**get_saved_orders_by_path**(*account_id*)

Saved orders for a specific account. Official documentation.

Client.**replace_saved_order**(*account_id*, *order_id*, *order_spec*)

Replace an existing saved order for an account. The existing saved order will be replaced by the new order. Official documentation.

3.10.3 Market Hours

Client.**get_hours_for_multiple_markets**(*markets*, *date*)
Retrieve market hours for specified markets. Official documentation.

Parameters

- **markets** – Market to return hours for. Iterable of *Markets*.
- **date** – The date for which market hours information is requested. Accepts `datetime.date` and `datetime.datetime`.

Client.**get_hours_for_single_market**(*market*, *date*)
Retrieve market hours for specified single market. Official documentation.

Parameters

- **markets** – Market to return hours for. Instance of *Markets*.
- **date** – The date for which market hours information is requested. Accepts `datetime.date` and `datetime.datetime`.

class tda.client.Client.**Markets**
Values for *get_hours_for_multiple_markets()* and *get_hours_for_single_market()*.

BOND = 'BOND'

EQUITY = 'EQUITY'

FOREX = 'FOREX'

FUTURE = 'FUTURE'

OPTION = 'OPTION'

3.10.4 Movers

Client.**get_movers**(*index*, *direction*, *change*)
Top 10 (up or down) movers by value or percent for a particular market. Official documentation.

Parameters

- **direction** – See *Movers.Direction*
- **change** – See *Movers.Change*

class tda.client.Client.**Movers**

class Change
Values for *get_movers()*

PERCENT = 'percent'

VALUE = 'value'

class Direction
Values for *get_movers()*

DOWN = 'down'

UP = 'up'

3.10.5 User Info and Preferences

`Client.`**`get_preferences`**(*account_id*)
Preferences for a specific account. Official documentation.

`Client.`**`get_user_principals`**(*fields=None*)
User Principal details. Official documentation.

`Client.`**`update_preferences`**(*account_id*, *preferences*)
Update preferences for a specific account.

Please note that the directOptionsRouting and directEquityRouting values cannot be modified via this operation. Official documentation.

class `tda.client.Client.`**`UserPrincipals`**

class Fields
An enumeration.

PREFERENCES = 'preferences'

STREAMER_CONNECTION_INFO = 'streamerConnectionInfo'

STREAMER_SUBSCRIPTION_KEYS = 'streamerSubscriptionKeys'

SURROGATE_IDS = 'surrogateIds'

3.10.6 Watchlists

`Client.`**`create_watchlist`**(*account_id*, *watchlist_spec*)
'Create watchlist for specific account.This method does not verify that the symbol or asset type are valid. Official documentation.

`Client.`**`delete_watchlist`**(*account_id*, *watchlist_id*)
Delete watchlist for a specific account. Official documentation.

`Client.`**`get_watchlist`**(*account_id*, *watchlist_id*)
Specific watchlist for a specific account. Official documentation.

`Client.`**`get_watchlists_for_multiple_accounts`**()
All watchlists for all of the user's linked accounts. Official documentation.

`Client.`**`get_watchlists_for_single_account`**(*account_id*)
All watchlists of an account. Official documentation.

`Client.`**`replace_watchlist`**(*account_id*, *watchlist_id*, *watchlist_spec*)
Replace watchlist for a specific account. This method does not verify that the symbol or asset type are valid. Official documentation.

`Client.`**`update_watchlist`**(*account_id*, *watchlist_id*, *watchlist_spec*)
Partially update watchlist for a specific account: change watchlist name, add to the beginning/end of a watchlist, update or delete items in a watchlist. This method does not verify that the symbol or asset type are valid. Official documentation.

CHAPTER 4

Streaming Client

A wapper around the TD Ameritrade Streaming API. This API is a websockets-based streaming API that provides to up-to-the-second data on market activity. Most impressively, it provides realtime data, including Level Two and time of sale data for major equities, options, and futures exchanges.

Here's an example of how you can receive book snapshots of `GOOG` (note if you run this outside regular trading hours you may not see anything):

```
from tda.auth import easy_client
from tda.client import Client
from tda.streaming import StreamClient

import asyncio
import json

client = easy_client(
        api_key='APIKEY',
        redirect_uri='https://localhost',
        token_path='/tmp/token.pickle')
stream_client = StreamClient(client, account_id=1234567890)

async def read_stream():
    await stream_client.login()
    await stream_client.quality_of_service(StreamClient.QOSLevel.EXPRESS)
    await stream_client.nasdaq_book_subs(['GOOG'])
    stream_client.add_timesale_options_handler(
            lambda msg: print(json.dumps(msg, indent=4)))

    while True:
        await stream_client.handle_message()

asyncio.get_event_loop().run_until_complete(read_stream())
```

This API uses Python coroutines to simplify implementation and preserve performance. As a result, it requires Python 3.8 or higher to use. `tda.stream` will not be available on older versions of Python.

4.1 Use Overview

The example above demonstrates the end-to-end workflow for using `tda.stream`. There's more in there than meets the eye, so let's dive into the details.

4.1.1 Logging In

Before we can perform any stream operations, the client must be logged in to the stream. Unlike the HTTP client, in which every request is authenticated using a token, this client sends unauthenticated requests and instead authenticates the entire stream. As a result, this login process is distinct from the token generation step that's used in the HTTP client.

Stream login is accomplished simply by calling *`StreamClient.login()`*. Once this happens successfully, all stream operations can be performed. Attemping to perform operations that require login before this function is called raises an exception.

`StreamClient.`**`login`**`()`
Official Documentation

Performs initial stream setup:

- Fetches streaming information from the HTTP client's *`get_user_principals()`* method
- Initializes the socket
- Builds and sends and authentication request
- Waits for response indicating login success

All stream operations are available after this method completes.

4.1.2 Setting Quality of Service

By default, the stream's update frequency is set to 1000ms. The frequency can be increased by calling the `quality_of_service` function and passing an appropriate `QOSLevel` value.

`StreamClient.`**`quality_of_service`**`(`*qos_level*`)`
Official Documentation

Specifies the frequency with which updated data should be sent to the client. If not called, the frequency will default to every second.

Parameters **`qos_level`** – Quality of service level to request. See *`QOSLevel`* for options.

`class` `StreamClient.`**`QOSLevel`**
Quality of service levels

`EXPRESS = '0'`
500ms between updates. Fastest available

`REAL_TIME = '1'`
750ms between updates

`FAST = '2'`
1000ms between updates. Default value.

`MODERATE = '3'`
1500ms between updates

SLOW = '4'
3000ms between updates

DELAYED = '5'
5000ms between updates

4.1.3 Subscribing to Streams

These functions have names that follow the pattern `SERVICE_NAME_subs`. These functions send a request to enable streaming data for a particular data stream. They are *not* thread safe, so they should only be called in series.

When subscriptions are called multiple times on the same stream, the results vary. What's more, these results aren't documented in the official documentation. As a result, it's recommended not to call a subscription function more than once for any given stream.

Some services, notably *Equity Charts* and *Futures Charts*, offer `SERVICE_NAME_add` functions which can be used to add symbols to the stream after the subscription has been created. For others, calling the subscription methods again seems to clear the old subscription and create a new one. Note this behavior is not officially documented, so this interpretation may be incorrect.

4.1.4 Registering Handlers

By themselves, the subscription functions outlined above do nothing except cause messages to be sent to the client. The `add_SERVICE_NAME_handler` functions register functions that will receive these messages when they arrive. When messages arrive, these handlers will be called serially. There is no limit to the number of handlers that can be registered to a service.

4.1.5 Handling Messages

Once the stream client is properly logged in, subscribed to streams, and has handlers registered, we can start handling messages. This is done simply by awaiting on the `handle_message()` function. This function reads a single message and dispatches it to the appropriate handler or handlers.

If a message is received for which no handler is registered, that message is ignored.

Handlers should take a single argument representing the stream message received:

```
import json

def sample_handler(msg):
    print(json.dumps(msg, indent=4))
```

4.1.6 Data Field Relabeling

Under the hood, this API returns JSON objects with numerical key representing labels:

```
{
    "service": "CHART_EQUITY",
    "timestamp": 1590597641293,
    "command": "SUBS",
    "content": [
        {
            "seq": 985,
```

(continues on next page)

(continued from previous page)

```
            "key": "MSFT",
            "1": 179.445,
            "2": 179.57,
            "3": 179.4299,
            "4": 179.52,
            "5": 53742.0,
            "6": 339,
            "7": 1590597540000,
            "8": 18409
        },
    ]
}
```

These labels are tricky to decode, and require a knowledge of the documentation to decode properly. `tda-api` makes your life easier by doing this decoding for you, replacing numerical labels with strings pulled from the documentation. For instance, the message above would be relabeled as:

```
{
    "service": "CHART_EQUITY",
    "timestamp": 1590597641293,
    "command": "SUBS",
    "content": [
        {
            "seq": 985,
            "key": "MSFT",
            "OPEN_PRICE": 179.445,
            "HIGH_PRICE": 179.57,
            "LOW_PRICE": 179.4299,
            "CLOSE_PRICE": 179.52,
            "VOLUME": 53742.0,
            "SEQUENCE": 339,
            "CHART_TIME": 1590597540000,
            "CHART_DAY": 18409
        },
    ]
}
```

This documentation describes the various fields and their numerical values. You can find them by investigating the various enum classes ending in `***Fields`.

Some streams, such as the ones described in *Level One Quotes*, allow you to specify a subset of fields to be returned. Subscription handlers for these services take a list of the appropriate field enums the extra `fields` parameter. If nothing is passed to this parameter, all supported fields are requested.

4.1.7 Interpreting Sequence Numbers

Many endpoints include a `seq` parameter in their data contents. The official documentation is unclear on the interpretation of this value: the time of sale documentation states that messages containing already-observed values of `seq` can be ignored, but other streams contain this field both in their metadata and in their content, and yet their documentation doesn't mention ignoring any `seq` values.

This presents a design choice: should `tda-api` ignore duplicate `seq` values on users' behalf? Given the ambiguity of the documentation, it was decided to not ignore them and instead pass them to all handlers. Clients are encouraged to use their judgment in handling these values.

4.1.8 Unimplemented Streams

This document lists the streams supported by `tda-api`. Eagle-eyed readers may notice that some streams are described in the documentation but were not implemented. This is due to complexity or anticipated lack of interest. If you feel you'd like a stream added, please file an issue here or see the contributing guidelines to learn how to add the functionality yourself.

4.2 Enabling Real-Time Data Access

By default, TD Ameritrade delivers delayed quotes. However, as of this writing, real time streaming is available for all streams, including quotes and level two depth of book data. It is also available for free, which in the author's opinion is an impressive feature for a retail brokerage. For most users it's enough to sign the relevant exchange agreements, although your mileage may vary.

Please remember that your use of this API is subject to agreeing to TDAmeritrade's terms of service. Please don't reach out to us asking for help enabling real-time data. Answers to most questions are a Google search away.

4.3 OHLCV Charts

These streams summarize trading activity on a minute-by-minute basis for equities and futures, providing OHLCV (Open/High/Low/Close/Volume) data.

4.3.1 Equity Charts

Minute-by-minute OHLCV data for equities.

`StreamClient.`**`chart_equity_subs`**(*symbols*)
Official documentation

Subscribe to equity charts. Behavior is undefined if called multiple times.

Parameters **`symbols`** – Equity symbols to subscribe to.

`StreamClient.`**`chart_equity_add`**(*symbols*)
Official documentation

Add a symbol to the equity charts subscription. Behavior is undefined if called before *`chart_equity_subs()`*.

Parameters **`symbols`** – Equity symbols to add to the subscription.

`StreamClient.`**`add_chart_equity_handler`**(*handler*)
Adds a handler to the equity chart subscription. See *Handling Messages* for details.

class `StreamClient.`**`ChartEquityFields`**
Official documentation

Data fields for equity OHLCV data. Primarily an implementation detail and not used in client code. Provided here as documentation for key values stored returned in the stream messages.

`SYMBOL = 0`
Ticker symbol in upper case. Represented in the stream as the `key` field.

`OPEN_PRICE = 1`
Opening price for the minute

HIGH_PRICE = 2
Highest price for the minute

LOW_PRICE = 3
Chart's lowest price for the minute

CLOSE_PRICE = 4
Closing price for the minute

VOLUME = 5
Total volume for the minute

SEQUENCE = 6
Identifies the candle minute. Explicitly labeled "not useful" in the official documentation.

CHART_TIME = 7
Milliseconds since Epoch

CHART_DAY = 8
Documented as not useful, included for completeness

4.3.2 Futures Charts

Minute-by-minute OHLCV data for futures.

StreamClient.**chart_futures_subs**(*symbols*)
Official documentation

Subscribe to futures charts. Behavior is undefined if called multiple times.

Parameters **symbols** – Futures symbols to subscribe to.

StreamClient.**chart_futures_add**(*symbols*)
Official documentation

Add a symbol to the futures chart subscription. Behavior is undefined if called before *chart_futures_subs()*.

Parameters **symbols** – Futures symbols to add to the subscription.

StreamClient.**add_chart_futures_handler**(*handler*)
Adds a handler to the futures chart subscription. See *Handling Messages* for details.

class StreamClient.**ChartFuturesFields**
Official documentation

Data fields for equity OHLCV data. Primarily an implementation detail and not used in client code. Provided here as documentation for key values stored returned in the stream messages.

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

CHART_TIME = 1
Milliseconds since Epoch

OPEN_PRICE = 2
Opening price for the minute

HIGH_PRICE = 3
Highest price for the minute

LOW_PRICE = 4
Chart's lowest price for the minute

CLOSE_PRICE = 5
Closing price for the minute

VOLUME = 6
Total volume for the minute

4.4 Level One Quotes

Level one quotes provide an up-to-date view of bid/ask/volume data. In particular they list the best available bid and ask prices, together with the requested volume of each. They are updated live as market conditions change.

4.4.1 Equities Quotes

Level one quotes for equities traded on NYSE, AMEX, and PACIFIC.

`StreamClient.`**`level_one_equity_subs`**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to level one equity quote data.

Parameters

- **symbols** – Equity symbols to receive quotes for
- **fields** – Iterable of *LevelOneEquityFields* representing the fields to return in streaming entries. If unset, all fields will be requested.

`StreamClient.`**`add_level_one_equity_handler`**(*handler*)
Register a function to handle level one equity quotes as they are sent. See *Handling Messages* for details.

class `StreamClient.`**`LevelOneEquityFields`**
Official documentation

Fields for equity quotes.

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

BID_PRICE = 1
Current Best Bid Price

ASK_PRICE = 2
Current Best Ask Price

LAST_PRICE = 3
Price at which the last trade was matched

BID_SIZE = 4
Number of shares for bid

ASK_SIZE = 5
Number of shares for ask

ASK_ID = 6
Exchange with the best ask

BID_ID = 7
Exchange with the best bid

TOTAL_VOLUME = 8
Aggregated shares traded throughout the day, including pre/post market hours. Note volume is set to zero at 7:28am ET.

LAST_SIZE = 9
Number of shares traded with last trade, in 100's

TRADE_TIME = 10
Trade time of the last trade, in seconds since midnight EST

QUOTE_TIME = 11
Trade time of the last quote, in seconds since midnight EST

HIGH_PRICE = 12
Day's high trade price. Notes:

- According to industry standard, only regular session trades set the High and Low.
- If a stock does not trade in the AM session, high and low will be zero.
- High/low reset to 0 at 7:28am ET

LOW_PRICE = 13
Day's low trade price. Same notes as `HIGH_PRICE`.

BID_TICK = 14
Indicates Up or Downtick (NASDAQ NMS & Small Cap). Updates whenever bid updates.

CLOSE_PRICE = 15
Previous day's closing price. Notes:

- Closing prices are updated from the DB when Pre-Market tasks are run by TD Ameritrade at 7:29AM ET.
- As long as the symbol is valid, this data is always present.
- This field is updated every time the closing prices are loaded from DB

EXCHANGE_ID = 16
Primary "listing" Exchange.

MARGINABLE = 17
Stock approved by the Federal Reserve and an investor's broker as being suitable for providing collateral for margin debt?

SHORTABLE = 18
Stock can be sold short?

ISLAND_BID_DEPRECATED = 19
Deprecated, documented for completeness.

ISLAND_ASK_DEPRECATED = 20
Deprecated, documented for completeness.

ISLAND_VOLUME_DEPRECATED = 21
Deprecated, documented for completeness.

QUOTE_DAY = 22
Day of the quote

TRADE_DAY = 23
Day of the trade

VOLATILITY = 24
Option Risk/Volatility Measurement. Notes:

- Volatility is reset to 0 when Pre-Market tasks are run at 7:28 AM ET
- Once per day descriptions are loaded from the database when Pre-Market tasks are run at 7:29:50 AM ET.

DESCRIPTION = 25
A company, index or fund name

LAST_ID = 26
Exchange where last trade was executed

DIGITS = 27
Valid decimal points. 4 digits for AMEX, NASDAQ, OTCBB, and PINKS, 2 for others.

OPEN_PRICE = 28
Day's Open Price. Notes:

- Open is set to ZERO when Pre-Market tasks are run at 7:28.
- If a stock doesn't trade the whole day, then the open price is 0.
- In the AM session, Open is blank because the AM session trades do not set the open.

NET_CHANGE = 29
Current Last-Prev Close

HIGH_52_WEEK = 30
Highest price traded in the past 12 months, or 52 weeks

LOW_52_WEEK = 31
Lowest price traded in the past 12 months, or 52 weeks

PE_RATIO = 32
Price to earnings ratio

DIVIDEND_AMOUNT = 33
Dividen earnings Per Share

DIVIDEND_YIELD = 34
Dividend Yield

ISLAND_BID_SIZE_DEPRECATED = 35
Deprecated, documented for completeness.

ISLAND_ASK_SIZE_DEPRECATED = 36
Deprecated, documented for completeness.

NAV = 37
Mutual Fund Net Asset Value

FUND_PRICE = 38
Mutual fund price

EXCHANGE_NAME = 39
Display name of exchange

DIVIDEND_DATE = 40
Dividend date

IS_REGULAR_MARKET_QUOTE = 41
Is last quote a regular quote

IS_REGULAR_MARKET_TRADE = 42
Is last trade a regular trade

REGULAR_MARKET_LAST_PRICE = 43
Last price, only used when `IS_REGULAR_MARKET_TRADE` is `True`

REGULAR_MARKET_LAST_SIZE = 44
Last trade size, only used when `IS_REGULAR_MARKET_TRADE` is `True`

REGULAR_MARKET_TRADE_TIME = 45
Last trade time, only used when `IS_REGULAR_MARKET_TRADE` is `True`

REGULAR_MARKET_TRADE_DAY = 46
Last trade date, only used when `IS_REGULAR_MARKET_TRADE` is `True`

REGULAR_MARKET_NET_CHANGE = 47
`REGULAR_MARKET_LAST_PRICE` minus `CLOSE_PRICE`

SECURITY_STATUS = 48
Indicates a symbols current trading status, Normal, Halted, Closed

MARK = 49
Mark Price

QUOTE_TIME_IN_LONG = 50
Last quote time in milliseconds since Epoch

TRADE_TIME_IN_LONG = 51
Last trade time in milliseconds since Epoch

REGULAR_MARKET_TRADE_TIME_IN_LONG = 52
Regular market trade time in milliseconds since Epoch

4.4.2 Options Quotes

Level one quotes for options. Note you can use *`Client.get_option_chain()`* to fetch available option symbols.

`StreamClient.`**`level_one_option_subs`**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to level one option quote data.

Parameters

- **symbols** – Option symbols to receive quotes for
- **fields** – Iterable of *`LevelOneOptionFields`* representing the fields to return in streaming entries. If unset, all fields will be requested.

`StreamClient.`**`add_level_one_option_handler`**(*handler*)
Register a function to handle level one options quotes as they are sent. See *Handling Messages* for details.

class `StreamClient.`**`LevelOneOptionFields`**
Official documentation

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

DESCRIPTION = 1
A company, index or fund name

BID_PRICE = 2
Current Best Bid Price

ASK_PRICE = 3
Current Best Ask Price

LAST_PRICE = 4
Price at which the last trade was matched

HIGH_PRICE = 5
Day's high trade price. Notes:

- According to industry standard, only regular session trades set the High and Low.
- If an option does not trade in the AM session, high and low will be zero.
- High/low reset to 0 at 7:28am ET.

LOW_PRICE = 6
Day's low trade price. Same notes as `HIGH_PRICE`.

CLOSE_PRICE = 7
Previous day's closing price. Closing prices are updated from the DB when Pre-Market tasks are run at 7:29AM ET.

TOTAL_VOLUME = 8
Aggregated shares traded throughout the day, including pre/post market hours. Reset to zero at 7:28am ET.

OPEN_INTEREST = 9
Open interest

VOLATILITY = 10
Option Risk/Volatility Measurement. Volatility is reset to 0 when Pre-Market tasks are run at 7:28 AM ET.

QUOTE_TIME = 11
Trade time of the last quote in seconds since midnight EST

TRADE_TIME = 12
Trade time of the last quote in seconds since midnight EST

MONEY_INTRINSIC_VALUE = 13
Money intrinsic value

QUOTE_DAY = 14
Day of the quote

TRADE_DAY = 15
Day of the trade

EXPIRATION_YEAR = 16
Option expiration year

MULTIPLIER = 17
Option multiplier

DIGITS = 18
Valid decimal points. 4 digits for AMEX, NASDAQ, OTCBB, and PINKS, 2 for others.

OPEN_PRICE = 19
Day's Open Price. Notes:

- Open is set to ZERO when Pre-Market tasks are run at 7:28.
- If a stock doesn't trade the whole day, then the open price is 0.
- In the AM session, Open is blank because the AM session trades do not set the open.

BID_SIZE = 20
Number of shares for bid

ASK_SIZE = 21
Number of shares for ask

LAST_SIZE = 22
Number of shares traded with last trade, in 100's

NET_CHANGE = 23
Current Last-Prev Close

STRIKE_PRICE = 24

CONTRACT_TYPE = 25

UNDERLYING = 26

EXPIRATION_MONTH = 27

DELIVERABLES = 28

TIME_VALUE = 29

EXPIRATION_DAY = 30

DAYS_TO_EXPIRATION = 31

DELTA = 32

GAMMA = 33

THETA = 34

VEGA = 35

RHO = 36

SECURITY_STATUS = 37
Indicates a symbols current trading status, Normal, Halted, Closed

THEORETICAL_OPTION_VALUE = 38

UNDERLYING_PRICE = 39

UV_EXPIRATION_TYPE = 40

MARK = 41
Mark Price

4.4.3 Futures Quotes

Level one quotes for futures.

`StreamClient.`**`level_one_futures_subs`**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to level one futures quote data.

Parameters

- **`symbols`** – Futures symbols to receive quotes for
- **`fields`** – Iterable of *`LevelOneFuturesFields`* representing the fields to return in streaming entries. If unset, all fields will be requested.

StreamClient.**add_level_one_futures_handler**(*handler*)
Register a function to handle level one futures quotes as they are sent. See *Handling Messages* for details.

class StreamClient.**LevelOneFuturesFields**
Official documentation

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

BID_PRICE = 1
Current Best Bid Price

ASK_PRICE = 2
Current Best Ask Price

LAST_PRICE = 3
Price at which the last trade was matched

BID_SIZE = 4
Number of shares for bid

ASK_SIZE = 5
Number of shares for ask

ASK_ID = 6
Exchange with the best ask

BID_ID = 7
Exchange with the best bid

TOTAL_VOLUME = 8
Aggregated shares traded throughout the day, including pre/post market hours

LAST_SIZE = 9
Number of shares traded with last trade

QUOTE_TIME = 10
Trade time of the last quote in milliseconds since epoch

TRADE_TIME = 11
Trade time of the last trade in milliseconds since epoch

HIGH_PRICE = 12
Day's high trade price

LOW_PRICE = 13
Day's low trade price

CLOSE_PRICE = 14
Previous day's closing price

EXCHANGE_ID = 15
Primary "listing" Exchange. Notes: * I → ICE * E → CME * L → LIFFEUS

DESCRIPTION = 16
Description of the product

LAST_ID = 17
Exchange where last trade was executed

OPEN_PRICE = 18
Day's Open Price

NET_CHANGE = 19
Current Last-Prev Close

FUTURE_PERCENT_CHANGE = 20
Current percent change

EXCHANGE_NAME = 21
Name of exchange

SECURITY_STATUS = 22
Trading status of the symbol. Indicates a symbol's current trading status, Normal, Halted, Closed.

OPEN_INTEREST = 23
The total number of futures ontracts that are not closed or delivered on a particular day

MARK = 24
Mark-to-Market value is calculated daily using current prices to determine profit/loss

TICK = 25
Minimum price movement

TICK_AMOUNT = 26
Minimum amount that the price of the market can change

PRODUCT = 27
Futures product

FUTURE_PRICE_FORMAT = 28
Display in fraction or decimal format.

FUTURE_TRADING_HOURS = 29
Trading hours. Notes:

- days: 0 = monday-friday, 1 = sunday.
- 7 = Saturday
- 0 = [-2000,1700] ==> open, close
- 1 = [-1530,-1630,-1700,1515] ==> open, close, open, close
- 0 = [-1800,1700,d,-1700,1900] ==> open, close, DST-flag, open, close
- If the DST-flag is present, the following hours are for DST days: http://www.cmegroup.com/trading_hours/

FUTURE_IS_TRADEABLE = 30
Flag to indicate if this future contract is tradable

FUTURE_MULTIPLIER = 31
Point value

FUTURE_IS_ACTIVE = 32
Indicates if this contract is active

FUTURE_SETTLEMENT_PRICE = 33
Closing price

FUTURE_ACTIVE_SYMBOL = 34
Symbol of the active contract

FUTURE_EXPIRATION_DATE = 35
Expiration date of this contract in milliseconds since epoch

4.4.4 Forex Quotes

Level one quotes for foreign exchange pairs.

`StreamClient.`**`level_one_forex_subs`**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to level one forex quote data.

Parameters

- **`symbols`** – Forex symbols to receive quotes for
- **`fields`** – Iterable of *`LevelOneForexFields`* representing the fields to return in streaming entries. If unset, all fields will be requested.

`StreamClient.`**`add_level_one_forex_handler`**(*handler*)
Register a function to handle level one forex quotes as they are sent. See *Handling Messages* for details.

`class` `StreamClient.`**`LevelOneForexFields`**
Official documentation

`SYMBOL = 0`
Ticker symbol in upper case. Represented in the stream as the `key` field.

`BID_PRICE = 1`
Current Best Bid Price

`ASK_PRICE = 2`
Current Best Ask Price

`LAST_PRICE = 3`
Price at which the last trade was matched

`BID_SIZE = 4`
Number of shares for bid

`ASK_SIZE = 5`
Number of shares for ask

`TOTAL_VOLUME = 6`
Aggregated shares traded throughout the day, including pre/post market hours

`LAST_SIZE = 7`
Number of shares traded with last trade

`QUOTE_TIME = 8`
Trade time of the last quote in milliseconds since epoch

`TRADE_TIME = 9`
Trade time of the last trade in milliseconds since epoch

`HIGH_PRICE = 10`
Day's high trade price

`LOW_PRICE = 11`
Day's low trade price

`CLOSE_PRICE = 12`
Previous day's closing price

`EXCHANGE_ID = 13`
Primary "listing" Exchange

DESCRIPTION = 14
Description of the product

OPEN_PRICE = 15
Day's Open Price

NET_CHANGE = 16
Current Last-Prev Close

EXCHANGE_NAME = 18
Name of exchange

DIGITS = 19
Valid decimal points

SECURITY_STATUS = 20
Trading status of the symbol. Indicates a symbols current trading status, Normal, Halted, Closed.

TICK = 21
Minimum price movement

TICK_AMOUNT = 22
Minimum amount that the price of the market can change

PRODUCT = 23
Product name

TRADING_HOURS = 24
Trading hours

IS_TRADABLE = 25
Flag to indicate if this forex is tradable

MARKET_MAKER = 26

HIGH_52_WEEK = 27
Higest price traded in the past 12 months, or 52 weeks

LOW_52_WEEK = 28
Lowest price traded in the past 12 months, or 52 weeks

MARK = 29
Mark-to-Market value is calculated daily using current prices to determine profit/loss

4.4.5 Futures Options Quotes

Level one quotes for futures options.

`StreamClient.`**`level_one_futures_options_subs`**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to level one futures options quote data.

Parameters

- **`symbols`** – Futures options symbols to receive quotes for
- **`fields`** – Iterable of *`LevelOneFuturesOptionsFields`* representing the fields to return in streaming entries. If unset, all fields will be requested.

`StreamClient.`**`add_level_one_futures_options_handler`**(*handler*)
Register a function to handle level one futures options quotes as they are sent. See *Handling Messages* for details.

class StreamClient.**LevelOneFuturesOptionsFields**

Official documentation

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

BID_PRICE = 1
Current Best Bid Price

ASK_PRICE = 2
Current Best Ask Price

LAST_PRICE = 3
Price at which the last trade was matched

BID_SIZE = 4
Number of shares for bid

ASK_SIZE = 5
Number of shares for ask

ASK_ID = 6
Exchange with the best ask

BID_ID = 7
Exchange with the best bid

TOTAL_VOLUME = 8
Aggregated shares traded throughout the day, including pre/post market hours

LAST_SIZE = 9
Number of shares traded with last trade

QUOTE_TIME = 10
Trade time of the last quote in milliseconds since epoch

TRADE_TIME = 11
Trade time of the last trade in milliseconds since epoch

HIGH_PRICE = 12
Day's high trade price

LOW_PRICE = 13
Day's low trade price

CLOSE_PRICE = 14
Previous day's closing price

EXCHANGE_ID = 15
Primary "listing" Exchange. Notes: * I → ICE * E → CME * L → LIFFEUS

DESCRIPTION = 16
Description of the product

LAST_ID = 17
Exchange where last trade was executed

OPEN_PRICE = 18
Day's Open Price

NET_CHANGE = 19
Current Last-Prev Close

FUTURE_PERCENT_CHANGE = 20
Current percent change

EXCHANGE_NAME = 21
Name of exchange

SECURITY_STATUS = 22
Trading status of the symbol. Indicates a symbols current trading status, Normal, Halted, Closed.

OPEN_INTEREST = 23
The total number of futures ontracts that are not closed or delivered on a particular day

MARK = 24
Mark-to-Market value is calculated daily using current prices to determine profit/loss

TICK = 25
Minimum price movement

TICK_AMOUNT = 26
Minimum amount that the price of the market can change

PRODUCT = 27
Futures product

FUTURE_PRICE_FORMAT = 28
Display in fraction or decimal format

FUTURE_TRADING_HOURS = 29
Trading hours

FUTURE_IS_TRADEABLE = 30
Flag to indicate if this future contract is tradable

FUTURE_MULTIPLIER = 31
Point value

FUTURE_IS_ACTIVE = 32
Indicates if this contract is active

FUTURE_SETTLEMENT_PRICE = 33
Closing price

FUTURE_ACTIVE_SYMBOL = 34
Symbol of the active contract

FUTURE_EXPIRATION_DATE = 35
Expiration date of this contract, in milliseconds since epoch

4.5 Level Two Order Book

Level two streams provide a view on continuous order books of various securities. The level two order book describes the current the current bids and asks on the market, and these streams provide snapshots of that state.

Due to the lack of official documentation, these streams are largely reverse engineered. While the labeled data represents a best effort attempt to interpret stream fields, it's possible that something is wrong or incorrectly labeled.

The documentation lists more book types than are implemented here. In particular, it also lists `FOREX_BOOK`, `FUTURES_BOOK`, and `FUTURES_OPTIONS_BOOK` as accessible streams. All experimentation has resulted in these streams refusing to connect, typically returning errors about unavailable services. Due to this behavior and the lack of official documentation for book streams generally, `tda-api` assumes these streams are not actually implemented, and so excludes them. If you have any insight into using them, please let us know.

4.5.1 Equities Order Books: NYSE and NASDAQ

`tda-api` supports level two data for NYSE and NASDAQ, which are the two major exchanges dealing in equities, ETFs, etc. Stocks are typically listed on one or the other, and it is useful to learn about the differences between them:

- "The NYSE and NASDAQ: How They Work" on Investopedia
- "Here's the difference between the NASDAQ and NYSE" on Business Insider
- "Can Stocks Be Traded on More Than One Exchange?" on Investopedia

You can identify on which exchange a symbol is listed by using *`Client.search_instruments()`*:

```
r = c.search_instruments(['GOOG'], projection=c.Instrument.Projection.FUNDAMENTAL)
assert r.ok, r.raise_for_status()
print(r.json()['GOOG']['exchange'])  # Outputs NASDAQ
```

However, many symbols have order books available on these streams even though this API call returns neither NYSE nor NASDAQ. The only sure-fire way to find out whether the order book is available is to attempt to subscribe and see what happens.

Note to preserve equivalence with what little documentation there is, the NYSE book is called "listed." Testing indicates this stream corresponds to the NYSE book, but if you find any behavior that suggests otherwise please let us know.

`StreamClient.`**`listed_book_subs`**(*symbols*)
: Subscribe to the NYSE level two order book. Note this stream has no official documentation.

`StreamClient.`**`add_listed_book_handler`**(*handler*)
: Register a function to handle level two NYSE book data as it is updated See *Handling Messages* for details.

`StreamClient.`**`nasdaq_book_subs`**(*symbols*)
: Subscribe to the NASDAQ level two order book. Note this stream has no official documentation.

`StreamClient.`**`add_nasdaq_book_handler`**(*handler*)
: Register a function to handle level two NASDAQ book data as it is updated See *Handling Messages* for details.

4.5.2 Options Order Book

This stream provides the order book for options. It's not entirely clear what exchange it aggregates from, but it's been tested to work and deliver data. The leading hypothesis is that it is bethe order book for the Chicago Board of Exchange options exchanges, although this is an admittedly an uneducated guess.

`StreamClient.`**`options_book_subs`**(*symbols*)
: Subscribe to the level two order book for options. Note this stream has no official documentation, and it's not entirely clear what exchange it corresponds to. Use at your own risk.

`StreamClient.`**`add_options_book_handler`**(*handler*)
: Register a function to handle level two options book data as it is updated See *Handling Messages* for details.

4.6 Time of Sale

The data in *Level Two Order Book* describes the bids and asks for various instruments, but by itself is insufficient to determine when trades actually take place. The time of sale streams notify on trades as they happen. Together with the level two data, they provide a fairly complete picture of what is happening on an exchange.

All time of sale streams uss a common set of fields:

class StreamClient.**TimesaleFields**
Official documentation

SYMBOL = 0
Ticker symbol in upper case. Represented in the stream as the `key` field.

TRADE_TIME = 1
Trade time of the last trade in milliseconds since epoch

LAST_PRICE = 2
Price at which the last trade was matched

LAST_SIZE = 3
Number of shares traded with last trade

LAST_SEQUENCE = 4
Number of shares for bid

4.6.1 Equity Trades

StreamClient.**timesale_equity_subs**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to time of sale notifications for equities.

Parameters **symbols** – Equity symbols to subscribe to

StreamClient.**add_timesale_equity_handler**(*handler*)
Register a function to handle equity trade notifications as they happen See *Handling Messages* for details.

4.6.2 Futures Trades

StreamClient.**timesale_futures_subs**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to time of sale notifications for futures.

Parameters **symbols** – Futures symbols to subscribe to

StreamClient.**add_timesale_futures_handler**(*handler*)
Register a function to handle futures trade notifications as they happen See *Handling Messages* for details.

4.6.3 Options Trades

StreamClient.**timesale_options_subs**(*symbols*, *, *fields=None*)
Official documentation

Subscribe to time of sale notifications for options.

Parameters **symbols** – Options symbols to subscribe to

StreamClient.**add_timesale_options_handler**(*handler*)
Register a function to handle options trade notifications as they happen See *Handling Messages* for details.

4.7 News Headlines

TD Ameritrade supposedly supports streaming news headlines. However, we have yet to receive any reports of successful access to this stream. Attempts to read this stream result in messages like the following, followed by TDA-initiated stream closure:

```
{
    "notify": [
        {
            "service": "NEWS_HEADLINE",
            "timestamp": 1591500923797,
            "content": {
                "code": 17,
                "msg": "Not authorized for all quotes."
            }
        }
    ]
}
```

The current hypothesis is that this stream requires some permissions or paid access that so far no one has had.If you manage to get this stream working, or even if you manage to get it to fail with a different message than the one above, please report it. In the meantime, `tda-api` provides the following methods for attempting to access this stream.

`StreamClient.`**`news_headline_subs`**(*symbols*)
Official documentation

Subscribe to news headlines related to the given symbols.

`StreamClient.`**`add_news_headline_handler`**(*handler*)
Register a function to handle news headlines as they are provided. See *Handling Messages* for details.

`class` `StreamClient.`**`NewsHeadlineFields`**
Official documentation

`SYMBOL = 0`
Ticker symbol in upper case. Represented in the stream as the `key` field.

`ERROR_CODE = 1`
Specifies if there is any error

`STORY_DATETIME = 2`
Headline's datetime in milliseconds since epoch

`HEADLINE_ID = 3`
Unique ID for the headline

`STATUS = 4`

`HEADLINE = 5`
News headline

`STORY_ID = 6`

`COUNT_FOR_KEYWORD = 7`

`KEYWORD_ARRAY = 8`

`IS_HOT = 9`

`STORY_SOURCE = 10`

4.8 Account Activity

This stream allows you to monitor your account activity, including order execution/cancellation/expiration/etc. `tda-api` provide utilities for setting up and reading the stream, but leaves the task of parsing the response XML object to the user.

`StreamClient.`**`account_activity_sub`**`()`
Official documentation

Subscribe to account activity for the account id associated with this streaming client. See *`AccountActivityFields`* for more info.

`StreamClient.`**`add_account_activity_handler`**`(`*`handler`*`)`
Adds a handler to the account activity subscription. See *Handling Messages* for details.

`class` `StreamClient.`**`AccountActivityFields`**
Official documentation

Data fields for equity account activity. Primarily an implementation detail and not used in client code. Provided here as documentation for key values stored returned in the stream messages.

`SUBSCRIPTION_KEY = 0`
Subscription key. Represented in the stream as the `key` field.

`ACCOUNT = 1`
Account # subscribed

`MESSAGE_TYPE = 2`
Refer to the message type table in the official documentation

`MESSAGE_DATA = 3`
The core data for the message. Either XML Message data describing the update, `NULL` in some cases, or plain text in case of `ERROR`.

CHAPTER 5

Order Templates

`tda-api` strives to be easy to use. This means making it easy to do simple things, while making it possible to do complicated things. Order construction is a major challenge to this mission: both simple and complicated orders use the same format, meaning simple orders require a surprising amount of sophistication to place.

We get around this by providing templates that make it easy to place common orders, while allowing advanced users to modify the orders returned from the templates to create more complex ones. Very advanced users can even create their own orders from scratch. This page describes the simple templates, while the *OrderBuilder Reference* page documents the order builder in all its complexity.

5.1 Using These Templates

These templates serve two purposes. First, they are designed to choose defaults so you can immediately *place them*. These defaults are:

- All orders execute during the current normal trading session. If placed outside of trading hours, the execute during the next normal trading session.
- Time-in-force is set to `DAY`.
- All other fields (such as requested destination, etc.) are left unset, meaning they receive default treatment from TD Ameritrade. Note this treatment depends on TDA's implementation, and may change without warning.

Secondly, they serve as starting points for building more complex order types. All templates return a pre-populated `OrderBuilder` object, meaning complex functionality can be specified by modifying the returned object. For example, here is how you would place an order to buy `GOOG` for no more than $1250 at any time in the next six months:

```
from tda.orders.equities import equity_buy_limit
from tda.orders.common import Duration, Session

client = ... # See "Authentication and Client Creation"

client.place_order(
```

(continues on next page)

(continued from previous page)

```
    equity_buy_limit('GOOG', 1, 1250.0)
        .set_duration(Duration.GOOD_TILL_CANCEL)
        .set_session(Session.SEAMLESS)
        .build())
```

You can find a full reference for all supported fields in *OrderBuilder Reference*.

5.2 Equity Templates

5.2.1 Buy orders

`tda.orders.equities.`**`equity_buy_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for an equity buy market order.

`tda.orders.equities.`**`equity_buy_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for an equity buy limit order.

5.2.2 Sell orders

`tda.orders.equities.`**`equity_sell_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for an equity sell market order.

`tda.orders.equities.`**`equity_sell_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for an equity sell limit order.

5.2.3 Sell short orders

`tda.orders.equities.`**`equity_sell_short_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for an equity short sell market order.

`tda.orders.equities.`**`equity_sell_short_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for an equity short sell limit order.

5.2.4 Buy to cover orders

`tda.orders.equities.`**`equity_buy_to_cover_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for an equity buy-to-cover market order.

`tda.orders.equities.`**`equity_buy_to_cover_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for an equity buy-to-cover limit order.

5.3 Options Templates

TD Ameritrade supports over a dozen options strategies, each of which involve a precise structure in the order builder. `tda-api` is slowly gaining support for these strategies, and they are documented here as they become ready for use. As time goes on, more templates will be added here.

In the meantime, you can construct all supported options orders using the *OrderBuilder*, although you will have to construct them yourself.

Note orders placed using these templates may be rejected, depending on the user's options trading authorization.

5.3.1 Where Do I Find Options Symbols?

All templates require option symbols, which are somewhat more involved than equity symbols. They encode the underlying, the expiration date, option type (put or call) and the strike price. They are especially tricky to extract because both the TD Ameritrade UI and the thinkorswim UI don't reveal the symbol in the option chain view.

Symbols can be generated manually by plugging into the `[Underlying_[Expiration][P/C][Strike]]` pattern. They can also be found by requesting the *Option Chain*.

5.3.2 Single Options

Buy and sell single options.

`tda.orders.options.`**`option_buy_to_open_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for a buy-to-open market order.

`tda.orders.options.`**`option_buy_to_open_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for a buy-to-open limit order.

`tda.orders.options.`**`option_sell_to_open_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for a sell-to-open market order.

`tda.orders.options.`**`option_sell_to_open_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for a sell-to-open limit order.

`tda.orders.options.`**`option_buy_to_close_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for a buy-to-close market order.

`tda.orders.options.`**`option_buy_to_close_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for a buy-to-close limit order.

`tda.orders.options.`**`option_sell_to_close_market`**(*symbol*, *quantity*)
Returns a pre-filled `OrderBuilder` for a sell-to-close market order.

`tda.orders.options.`**`option_sell_to_close_limit`**(*symbol*, *quantity*, *price*)
Returns a pre-filled `OrderBuilder` for a sell-to-close limit order.

5.3.3 Vertical Spreads

Vertical spreads are a complex option strategy that provides both limited upside and limited downside. They are constructed using by buying an option at one strike while simultaneously selling another option with the same underlying and expiration date, except with a different strike, and they can be constructed using either puts or call. You can find more information about this strategy on Investopedia

`tda-api` provides utilities for opening and closing vertical spreads in various ways. It follows the standard `(bull/bear) (put/call)` naming convention, where the name specifies the market attitude and the option type used in construction.

For consistency's sake, the option with the smaller strike price is always passed first, followed by the higher strike option. You can find the option symbols by consulting the return value of the *Option Chain* client call.

Call Verticals

tda.orders.options.**bull_call_vertical_open**(*long_call_symbol*, *short_call_symbol*, *quantity*, *net_debit*)
Returns a pre-filled OrderBuilder that opens a bull call vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bull_call_vertical_close**(*long_call_symbol*, *short_call_symbol*, *quantity*, *net_credit*)
Returns a pre-filled OrderBuilder that closes a bull call vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bear_call_vertical_open**(*short_call_symbol*, *long_call_symbol*, *quantity*, *net_credit*)
Returns a pre-filled OrderBuilder that opens a bear call vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bear_call_vertical_close**(*short_call_symbol*, *long_call_symbol*, *quantity*, *net_debit*)
Returns a pre-filled OrderBuilder that closes a bear call vertical position. See *Vertical Spreads* for details.

Put Verticals

tda.orders.options.**bull_put_vertical_open**(*long_put_symbol*, *short_put_symbol*, *quantity*, *net_credit*)
Returns a pre-filled OrderBuilder that opens a bull put vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bull_put_vertical_close**(*long_put_symbol*, *short_put_symbol*, *quantity*, *net_debit*)
Returns a pre-filled OrderBuilder that closes a bull put vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bear_put_vertical_open**(*short_put_symbol*, *long_put_symbol*, *quantity*, *net_debit*)
Returns a pre-filled OrderBuilder that opens a bear put vertical position. See *Vertical Spreads* for details.

tda.orders.options.**bear_put_vertical_close**(*short_put_symbol*, *long_put_symbol*, *quantity*, *net_credit*)
Returns a pre-filled OrderBuilder that closes a bear put vertical position. See *Vertical Spreads* for details.

5.4 Utility Methods

These methods return orders that represent complex multi-order strategies, namely "one cancels other" and "first triggers second" strategies. Note they expect all their parameters to be of type OrderBuilder. You can construct these orders using the templates above or by *creating them from scratch*.

tda.orders.common.**one_cancels_other**(*order1*, *order2*)
If one of the orders is executed, immediately cancel the other.

tda.orders.common.**first_triggers_second**(*first_order*, *second_order*)
If first_order is executed, immediately place second_order.

5.5 What happened to EquityOrderBuilder?

Long-time users may notice that this documentation no longer mentions the EquityOrderBuilder class. This class used to be used to create equities orders, and offered a subset of the functionality offered by the *OrderBuilder*. This class has been deprecated in favor of the order builder and the above templates, and will be removed from a future release.

In the meantime, you can continue using this order builder, although you really should migrate to the new one soon. You can find documentation for this class in the older versions of `tda-api`'s documentation.

CHAPTER 6

`OrderBuilder` Reference

The *`Client.place_order()`* method expects a rather complex JSON object that describes the desired order. TDA provides some example order specs to illustrate the process and provides a schema in the place order documentation, but beyond that we're on our own. `tda-api` aims to be useful to everyone, from users who want to easily place common equities and options trades, to advanced users who want to place complex multi-leg, multi-asset type trades.

For users interested in simple trades, `tda-api` supports pre-built *Order Templates* that allow fast construction of many common trades. Advanced users can modify these trades however they like, and can even build trades from scratch.

This page describes the features of the complete order schema in all their complexity. It is aimed at advanced users who want to create complex orders. Less advanced users can use the *order templates* to create orders. If they find themselves wanting to go beyond those templates, they can return to this page to learn how.

6.1 Optional: Order Specification Introduction

Before we dive in to creating order specs, let's briefly introduce their structure. This section is optional, although users wanting to use more advanced featured like stop prices and complex options orders will likely want to read it.

Here is an example of a spec that places a limit order to buy 13 shares of `MSFT` for no more than $190. This is exactly the order that would be returned by *`tda.orders.equities.equity_buy_limit()`*:

```
{
    "session": "NORMAL",
    "duration": "DAY",
    "orderType": "LIMIT",
    "price": "190.90",
    "orderLegCollection": [
        {
            "instruction": "BUY",
            "instrument": {
                "assetType": "EQUITY",
                "symbol": "MSFT"
```

(continues on next page)

(continued from previous page)

```
            },
            "quantity": 1
        }
    ],
    "orderStrategyType": "SINGLE"
}
```

Some key points are:

- The `LIMIT` order type notifies TD that you'd like to place a limit order.
- The order strategy type is `SINGLE`, meaning this order is not a composite order.
- The order leg collection contains a single leg to purchase the equity.
- The price is specified *outside* the order leg. This may seem counterintuitive, but it's important when placing composite options orders.

If this seems like a lot of detail to specify a rather simple order, it is. The thing about the order spec object is that it can express *every* order that can be made through the TD Ameritrade API. For an advanced example, here is a order spec for a standing order to enter a long position in `GOOG` at $1310 or less that triggers a one-cancels-other order that exits the position if the price rises to $1400 or falls below $1250:

```
{
    "session": "NORMAL",
    "duration": "GOOD_TILL_CANCEL",
    "orderType": "LIMIT",
    "price": "1310.00",
    "orderLegCollection": [
        {
            "instruction": "BUY",
            "instrument": {
                "assetType": "EQUITY",
                "symbol": "GOOG"
            },
            "quantity": 1
        }
    ],
    "orderStrategyType": "TRIGGER",
    "childOrderStrategies": [
        {
            "orderStrategyType": "OCO",
            "childOrderStrategies": [
                {
                    "session": "NORMAL",
                    "duration": "GOOD_TILL_CANCEL",
                    "orderType": "LIMIT",
                    "price": "1400.00",
                    "orderLegCollection": [
                        {
                            "instruction": "SELL",
                            "instrument": {
                                "assetType": "EQUITY",
                                "symbol": "GOOG"
                            },
                            "quantity": 1
                        }
                    ]
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "session": "NORMAL",
                    "duration": "GOOD_TILL_CANCEL",
                    "orderType": "STOP_LIMIT",
                    "stopPrice": "1250.00",
                    "orderLegCollection": [
                        {
                            "instruction": "SELL",
                            "instrument": {
                                "assetType": "EQUITY",
                                "symbol": "GOOG"
                            },
                            "quantity": 1
                        }
                    ]
                }
            ]
        }
    ]
}
```

While this looks complex, it can be broken down into the same components as the simpler buy order:

- This time, the `LIMIT` order type applies to the top-level order.
- The order strategy type is `TRIGGER`, which tells TD Ameritrade to hold off placing the the second order until the first one completes.
- The order leg collection still contains a single leg, and the price is still defined outside the order leg. This is typical for equities orders.

There are also a few things that aren't there in the simple buy order:

- The `childOrderStrategies` contains the `OCO` order that is triggered when the first `LIMIT` order is executed.
- If you look carefully, you'll notice that the inner `OCO` is a fully-featured suborder in itself.

This order is large and complex, and it takes a lot of reading to understand what's going on here. Fortunately for you, you don't have to; `tda-api` cuts down on this complexity by providing templates and helpers to make building orders easy:

```
from tda.orders.common import OrderType
from tda.orders.generic import OrderBuilder

one_triggers_other(
    equity_buy_limit('GOOG', 1, 1310),
    one_cancels_other(
        equity_sell_limit('GOOG', 1, 1400),
        equity_sell_limit('GOOG', 1, 1240)
            .set_order_type(OrderType.STOP_LIMIT)
            .clear_price()
            .set_stop_price(1250)
    )
```

You can find the full listing of order templates and utility functions *here*.

Now that you have some background on how orders are structured, let's dive into the order builder itself.

6.2 `OrderBuilder` Reference

This section provides a detailed reference of the generic order builder. You can use it to help build your own custom orders, or you can modify the pre-built orders generated by `tda-api`'s order templates.

Unfortunately, this reference is largely reverse-engineered. It was initially generated from the schema provided in the official API documents, but many of the finer points, such as which fields should be populated for which order types, etc. are best guesses. If you find something is inaccurate or missing, please let us know.

That being said, experienced traders who understand how various order types and complex strategies work should find this builder easy to use, at least for the order types with which they are familiar. Here are some resources you can use to learn more, courtesy of the Securites and Exchange Commission:

- Trading Basics: Understanding the Different Ways to Buy and Sell Stock
- Trade Execution: What Every Investor Should Know
- Investor Bulletin: An Introduction to Options

You can also find TD Ameritrade's official documentation on orders here, although it doesn't actually cover all functionality that `tda-api` supports.

6.2.1 Order Types

Here are the order types that can be used:

class `tda.orders.common.`**OrderType**
Type of equity or option order to place.

MARKET = 'MARKET'
Execute the order immediately at the best-available price. More Info.

LIMIT = 'LIMIT'
Execute the order at your price or better. More info.

STOP = 'STOP'
Wait until the price reaches the stop price, and then immediately place a market order. More Info.

STOP_LIMIT = 'STOP_LIMIT'
Wait until the price reaches the stop price, and then immediately place a limit order at the specified price. More Info.

TRAILING_STOP = 'TRAILING_STOP'
Similar to `STOP`, except if the price moves in your favor, the stop price is adjusted in that direction. Places a market order if the stop condition is met. More info.

TRAILING_STOP_LIMIT = 'TRAILING_STOP_LIMIT'
Similar to `STOP_LIMIT`, except if the price moves in your favor, the stop price is adjusted in that direction. Places a limit order at the specified price if the stop condition is met. More info.

MARKET_ON_CLOSE = 'MARKET_ON_CLOSE'
Place the order at the closing price immediately upon market close. More info

EXERCISE = 'EXERCISE'
Exercise an option.

NET_DEBIT = 'NET_DEBIT'
Place an order for an options spread resulting in a net debit. More info

NET_CREDIT = 'NET_CREDIT'
Place an order for an options spread resulting in a net credit. More info

NET_ZERO = 'NET_ZERO'
Place an order for an options spread resulting in neither a credit nor a debit. More info

OrderBuilder.**set_order_type**(*order_type*)
Set the order type. See *OrderType* for details.

OrderBuilder.**clear_order_type**()
Clear the order type.

6.2.2 Session and Duration

Together, these fields control when the order will be placed and how long it will remain active. Note tda-api's *templates* place orders that are active for the duration of the current normal trading session. If you want to modify the default session and duration, you can use these methods to do so.

class tda.orders.common.**Session**
The market session during which the order trade should be executed.

NORMAL = 'NORMAL'
Normal market hours, from 9:30am to 4:00pm Eastern.

AM = 'AM'
Premarket session, from 8:00am to 9:30am Eastern.

PM = 'PM'
After-market session, from 4:00pm to 8:00pm Eastern.

SEAMESS = 'SEAMLESS'
Unclear what this means. If you know, please tell us.

class tda.orders.common.**Duration**
Length of time over which the trade will be active.

DAY = 'DAY'
Cancel the trade at the end of the trading day. Note if the order cannot be filled all at once, you may see partial executions throughout the day.

GOOD_TILL_CANCEL = 'GOOD_TILL_CANCEL'
Keep the trade open for six months, or until the end of the cancel date, whichever is shorter. Note if the order cannot be filled all at once, you may see partial executions over the lifetime of the order.

FILL_OR_KILL = 'FILL_OR_KILL'
Either execute the order immediately at the specified price, or cancel it immediately.

OrderBuilder.**set_duration**(*duration*)
Set the order duration. See *Duration* for details.

OrderBuilder.**clear_duration**()
Clear the order duration.

OrderBuilder.**set_session**(*session*)
Set the order session. See *Session* for details.

OrderBuilder.**clear_session**()
Clear the order session.

6.2.3 Price

Price is the amount you'd like to pay for each unit of the position you're taking:

- For equities and simple options limit orders, this is the price which you'd like to pay/receive.
- For complex options limit orders (net debit/net credit), this is the total credit or debit you'd like to receive.

In other words, the price is the sum of the prices of the *Order Legs*. This is particularly powerful for complex multi-leg options orders, which support complex top and/or limit orders that trigger when the price of a position reaches certain levels. In those cases, the price of an order can drop below the specified price as a result of movements in multiple legs of the trade.

`OrderBuilder.`**`set_price`**(*price*)
Set the order price.

`OrderBuilder.`**`clear_price`**()
Clear the order price

6.2.4 Order Legs

Order legs are where the actual assets being bought or sold are specified. For simple equity or single-options orders, there is just one leg. However, for complex multi-leg options trades, there can be more than one leg.

Note that order legs often do not execute all at once. Order legs can be executed over the specified *`Duration`* of the order. What's more, if order legs request a large number of shares, legs themselves can be partially filled. You can control this setting using the *`SpecialInstruction`* value `ALL_OR_NONE`.

With all that out of the way, order legs are relatively simple to specify. `tda-api` currently supports equity and option order legs:

`OrderBuilder.`**`add_equity_leg`**(*instruction*, *symbol*, *quantity*)
Add an equity order leg.

Parameters

- **`instruction`** – Instruction for the leg. See *`EquityInstruction`* for valid options.
- **`symbol`** – Equity symbol
- **`quantity`** – Number of shares for the order

`class` `tda.orders.common.`**`EquityInstruction`**
Instructions for opening and closing equity positions.

`BUY = 'BUY'`
Open a long equity position

`SELL = 'SELL'`
Close a long equity position

`SELL_SHORT = 'SELL_SHORT'`
Open a short equity position

`BUY_TO_COVER = 'BUY_TO_COVER'`
Close a short equity position

`OrderBuilder.`**`add_option_leg`**(*instruction*, *symbol*, *quantity*)
Add an option order leg.

Parameters

- **`instruction`** – Instruction for the leg. See *`OptionInstruction`* for valid options.
- **`symbol`** – Option symbol
- **`quantity`** – Number of contracts for the order

class tda.orders.common.**OptionInstruction**
Instructions for opening and closing options positions.

BUY_TO_OPEN = 'BUY_TO_OPEN'
Enter a new long option position

SELL_TO_CLOSE = 'SELL_TO_CLOSE'
Exit an existing long option position

SELL_TO_OPEN = 'SELL_TO_OPEN'
Enter a short position in an option

BUY_TO_CLOSE = 'BUY_TO_CLOSE'
Exit an existing short position in an option

OrderBuilder.**clear_order_legs**()
Clear all order legs.

6.2.5 Requested Destination

By default, TD Ameritrade sends trades to whichever exchange provides the best price. This field allows you to request a destination exchange for your trade, although whether your order is actually executed there is up to TDA.

class tda.orders.common.**Destination**
Destinations for when you want to request a specific destination for your order.

INET = 'INET'

ECN_ARCA = 'ECN_ARCA'

CBOE = 'CBOE'

AMEX = 'AMEX'

PHLX = 'PHLX'

ISE = 'ISE'

BOX = 'BOX'

NYSE = 'NYSE'

NASDAQ = 'NASDAQ'

BATS = 'BATS'

C2 = 'C2'

AUTO = 'AUTO'

OrderBuilder.**set_requested_destination**(*requested_destination*)
Set the requested destination. See *Destination* for details.

OrderBuilder.**clear_requested_destination**()
Clear the requested destination.

6.2.6 Special Instructions

Trades can contain special instructions which handle some edge cases:

class tda.orders.common.**SpecialInstruction**
Special instruction for trades.

ALL_OR_NONE = 'ALL_OR_NONE'
Disallow partial order execution. More info.

DO_NOT_REDUCE = 'DO_NOT_REDUCE'
Do not reduce order size in response to cash dividends. More info.

ALL_OR_NONE_DO_NOT_REDUCE = 'ALL_OR_NONE_DO_NOT_REDUCE'
Combination of ALL_OR_NONE and DO_NOT_REDUCE.

OrderBuilder.**set_special_instruction**(*special_instruction*)
Set the special instruction. See *SpecialInstruction* for details.

OrderBuilder.**clear_special_instruction**()
Clear the special instruction.

6.2.7 Complex Options Strategies

TD Ameritrade supports a number of complex options strategies. These strategies are complex affairs, with each leg of the trade specified in the order legs. TD performs additional validation on these strategies, so they are somewhat complicated to place. However, the benefit is more flexibility, as trades like trailing stop orders based on net debit/credit can be specified.

Unfortunately, due to the complexity of these orders and the lack of any real documentation, we cannot offer definitively say how to structure these orders. A few things have been observed, however:

- The legs of the order can be placed by adding them as option order legs using *add_option_leg()*.
- For spreads resulting in a new debit/credit, the price represents the overall debit or credit desired.

If you successfully use these strategies, we want to know about it. Please let us know by joining our Discord server to chat about it, or by creating a feature request.

class tda.orders.common.**ComplexOrderStrategyType**
Explicit order strategies for executing multi-leg options orders.

NONE = 'NONE'
No complex order strategy. This is the default.

COVERED = 'COVERED'
Covered call

VERTICAL = 'VERTICAL'
Vertical spread

BACK_RATIO = 'BACK_RATIO'
Ratio backspread

CALENDAR = 'CALENDAR'
Calendar spread

DIAGONAL = 'DIAGONAL'
Diagonal spread

STRADDLE = 'STRADDLE'
Straddle spread

STRANGLE = 'STRANGLE'
Strandle spread

COLLAR_SYNTHETIC = 'COLLAR_SYNTHETIC'

BUTTERFLY = 'BUTTERFLY'
Butterfly spread

CONDOR = 'CONDOR'
Condor spread

IRON_CONDOR = 'IRON_CONDOR'
Iron condor spread

VERTICAL_ROLL = 'VERTICAL_ROLL'
Roll a vertical spread

COLLAR_WITH_STOCK = 'COLLAR_WITH_STOCK'
Collar strategy

DOUBLE_DIAGONAL = 'DOUBLE_DIAGONAL'
Double diagonal spread

UNBALANCED_BUTTERFLY = 'UNBALANCED_BUTTERFLY'
Unbalanced butterfy spread

UNBALANCED_CONDOR = 'UNBALANCED_CONDOR'

UNBALANCED_IRON_CONDOR = 'UNBALANCED_IRON_CONDOR'

UNBALANCED_VERTICAL_ROLL = 'UNBALANCED_VERTICAL_ROLL'

CUSTOM = 'CUSTOM'
A custom multi-leg order strategy.

OrderBuilder.**set_complex_order_strategy_type**(*complex_order_strategy_type*)
Set the complex order strategy type. See *ComplexOrderStrategyType* for details.

OrderBuilder.**clear_complex_order_strategy_type**()
Clear the complex order strategy type.

6.2.8 Composite Orders

tda-api supports composite order strategies, in which execution of one order has an effect on another:

- OCO, or "one cancels other" orders, consist of a pair of orders where execution of one immediately cancels the other.
- TRIGGER orders consist of a pair of orders where execution of one immediately results in placement of the other.

tda-api provides helpers to specify these easily: *one_cancels_other()* and *first_triggers_second()*. This is almost certainly easier than specifying these orders manually. However, if you still want to create them yourself, you can specify these composite order strategies like so:

class tda.orders.common.**OrderStrategyType**
Rules for composite orders.

SINGLE = 'SINGLE'
No chaining, only a single order is submitted

OCO = 'OCO'
Execution of one order cancels the other

TRIGGER = 'TRIGGER'
Execution of one order triggers placement of the other

OrderBuilder.**set_order_strategy_type**(*order_strategy_type*)
Set the order strategy type. See *OrderStrategyType* for more details.

`OrderBuilder.`**`clear_order_strategy_type`**`()`
Clear the order strategy type.

6.2.9 Undocumented Fields

Unfortunately, your humble author is not an expert in all things trading. The order spec schema describes some things that are outside my ability to document, so rather than make stuff up, I'm putting them here in the hopes that someone will come along and shed some light on them. You can make suggestions by filing an issue on our GitHub issues page, or by joining our Discord server.

Quantity

This one seems obvious: doesn't the quantity mean the number of stock I want to buy? The trouble is that the order legs also have a `quantity` field, which suggests this field means something else. The leading hypothesis is that is outlines the number of copies of the order to place, although we have yet to verify that.

`OrderBuilder.`**`set_quantity`**`(`*quantity*`)`
Exact semantics unknown. See *Quantity* for a discussion.

`OrderBuilder.`**`clear_quantity`**`()`
Clear the order-level quantity. Note this does not affect order legs.

Stop Order Configuration

Stop orders and their variants (stop limit, trailing stop, trailing stop limit) support some rather complex configuration. Both stops prices and the limit prices of the resulting order can be configured to follow the market in a dynamic fashion. The market dimensions that they follow can also be configured differently, and it appears that which dimensions are supported varies by order type.

We have unfortunately not yet done a thorough analysis of what's supported, nor have we made the effort to make it simple and easy. While we're *pretty* sure we understand how these fields work, they've been temporarily placed into the "undocumented" section, pending a followup. Users are invited to experiment with these fields at their own risk.

`OrderBuilder.`**`set_stop_price`**`(`*stop_price*`)`
Set the stop price.

`OrderBuilder.`**`clear_stop_price`**`()`
Clear the stop price.

`class` `tda.orders.common.`**`StopPriceLinkBasis`**
An enumeration.

`MANUAL = 'MANUAL'`

`BASE = 'BASE'`

`TRIGGER = 'TRIGGER'`

`LAST = 'LAST'`

`BID = 'BID'`

`ASK = 'ASK'`

`ASK_BID = 'ASK_BID'`

`MARK = 'MARK'`

`AVERAGE = 'AVERAGE'`

OrderBuilder.**set_stop_price_link_basis**(*stop_price_link_basis*)
Set the stop price link basis. See *StopPriceLinkBasis* for details.

OrderBuilder.**clear_stop_price_link_basis**()
Clear the stop price link basis.

class tda.orders.common.**StopPriceLinkType**
An enumeration.

VALUE = 'VALUE'

PERCENT = 'PERCENT'

TICK = 'TICK'

OrderBuilder.**set_stop_price_link_type**(*stop_price_link_type*)
Set the stop price link type. See *StopPriceLinkType* for details.

OrderBuilder.**clear_stop_price_link_type**()
Clear the stop price link type.

OrderBuilder.**set_stop_price_offset**(*stop_price_offset*)
Set the stop price offset.

OrderBuilder.**clear_stop_price_offset**()
Clear the stop price offset.

class tda.orders.common.**StopType**
An enumeration.

STANDARD = 'STANDARD'

BID = 'BID'

ASK = 'ASK'

LAST = 'LAST'

MARK = 'MARK'

OrderBuilder.**set_stop_type**(*stop_type*)
Set the stop type. See *StopType* for more details.

OrderBuilder.**clear_stop_type**()
Clear the stop type.

class tda.orders.common.**PriceLinkBasis**
An enumeration.

MANUAL = 'MANUAL'

BASE = 'BASE'

TRIGGER = 'TRIGGER'

LAST = 'LAST'

BID = 'BID'

ASK = 'ASK'

ASK_BID = 'ASK_BID'

MARK = 'MARK'

AVERAGE = 'AVERAGE'

OrderBuilder.**set_price_link_basis**(*price_link_basis*)
Set the price link basis. See *PriceLinkBasis* for details.

OrderBuilder.**clear_price_link_basis**()
Clear the price link basis.

class tda.orders.common.**PriceLinkType**
An enumeration.

VALUE = 'VALUE'

PERCENT = 'PERCENT'

TICK = 'TICK'

OrderBuilder.**set_price_link_type**(*price_link_type*)
Set the price link type. See *PriceLinkType* for more details.

OrderBuilder.**clear_price_link_type**()
Clear the price link basis.

OrderBuilder.**set_activation_price**(*activation_price*)
Set the activation price.

OrderBuilder.**clear_activation_price**()
Clear the activation price.

CHAPTER 7

Utilities

This section describes miscellaneous utility methods provided by `tda-api`. All utilities are presented under the `Utils` class:

class tda.utils.**Utils**(*client*, *account_id*)

Helper for placing orders on equities. Provides easy-to-use implementations for common tasks such as market and limit orders.

__init__(*client*, *account_id*)

Creates a new `Utils` instance. For convenience, this object assumes the user wants to work with a single account ID at a time.

set_account_id(*account_id*)

Set the account ID used by this `Utils` instance.

7.1 Get the Most Recent Order

For successfully placed orders, *tda.client.Client.place_order()* returns the ID of the newly created order, encoded in the `r.headers['Location']` header. This method inspects the response and extracts the order ID from the contents, if it's there. This order ID can then be used to monitor or modify the order as described in the *Client documentation*. Example usage:

```
# Assume client and order already exist and are valid
account_id = 123456
r = client.place_order(account_id, order)
assert r.ok, raise_for_status()
order_id = Utils(account_id, client).extract_order_id(r)
assert order_id is not None
```

Utils.**extract_order_id**(*place_order_response*)

Attempts to extract the order ID from a response object returned by *Client.place_order()*. Return `None` if the order location is not contained in the response.

Parameters **place_order_response** – Order response as returned by *Client.place_order()*. Note this method requires that the order was successful.

Raises **ValueError** – if the order was not succesful or if the order's account ID is not equal to the account ID set in this `Utils` object.

CHAPTER 8

Example Application

To illustrate some of the functionality of `tda-api`, here is an example application that finds stocks that pay a dividend during the month of your birthday and purchases one of each.

```
from urllib.request import urlopen

import atexit
import datetime
import dateutil
import sys
import tda

API_KEY = 'YOUR_API_KEY@AMER.OAUTHAP'
REDIRECT_URI = 'YOUR_REDIRECT_URI'
TOKEN_PATH = '/YOUR/TOKEN/PATH'
YOUR_BIRTHDAY = datetime.datetime(year=1969, month=4, day=20)


def make_webdriver():
    # Import selenium here because it's slow to import
    from selenium import webdriver

    driver = webdriver.Chrome()
    atexit.register(lambda: driver.quit())
    return driver


# Create a new client
client = tda.auth.easy_client(
    API_KEY,
    REDIRECT_URI,
    TOKEN_PATH,
    make_webdriver)

# Load S&P 500 composition from documentation
```

(continues on next page)

(continued from previous page)

```
sp500 = urlopen(
    'https://tda-api.readthedocs.io/en/latest/_static/sp500.txt').read().decode().
↪split()

# Fetch fundamentals for all symbols and filter out the ones with ex-dividend
# dates in the future and dividend payment dates on your birth month. Note we
# perform the fetch in two calls because the API places an upper limit on the
# number of symbols you can fetch at once.
today = datetime.datetime.today()
birth_month_dividends = []
for s in (sp500[:250], sp500[250:]):
    r = client.search_instruments(
        s, tda.client.Client.Instrument.Projection.FUNDAMENTAL)
    assert r.ok, r.raise_for_status()

    for symbol, f in r.json().items():

        # Parse ex-dividend date
        ex_div_string = f['fundamental']['dividendDate']
        if not ex_div_string.strip():
            continue
        ex_dividend_date = dateutil.parser.parse(ex_div_string)

        # Parse payment date
        pay_date_string = f['fundamental']['dividendPayDate']
        if not pay_date_string.strip():
            continue
        pay_date = dateutil.parser.parse(pay_date_string)

        # Check dates
        if (ex_dividend_date > today
                and pay_date.month == YOUR_BIRTHDAY.month):
            birth_month_dividends.append(symbol)

if not birth_month_dividends:
    print('Sorry, no stocks are paying out in your birth month yet. This is ',
          'most likely because the dividends haven\'t been announced yet. ',
          'Try again closer to your birthday.')
    sys.exit(1)

# Purchase one share of each the stocks that pay in your birthday month.
account_id = int(input(
    'Input your TDA account number to place orders (<Ctrl-C> to quit): '))
for symbol in birth_month_dividends:
    print('Buying one share of', symbol)

    # Build the order spec and place the order
    builder = tda.orders.EquityOrderBuilder(symbol, 1)
    builder.set_instruction(builder.Instruction.BUY)
    builder.set_order_type(builder.OrderType.MARKET)
    builder.set_duration(tda.orders.Duration.DAY)
    builder.set_session(tda.orders.Session.NORMAL)
    order = builder.build()

    r = client.place_order(account_id, order)
    assert r.ok, r.raise_for_status()
```

CHAPTER 9

Getting Help

`tda-api` is not perfect. Features are missing, documentation may be out of date, and it almost certainly contains bugs. If you think of a way in which `tda-api` can be improved, we're more than happy to hear it.

This page outlines the process for getting help if you found a bug. If you need general help using `tda-api`, or just want to chat with other people interested in developing trading strategies, you can join our discord.

If you still want to submit an issue, we ask that you follow a few guidelines to make both our lives easier:

9.1 Enable Logging

Behind the scenes, `tda-api` performs diagnostic logging of its activity using Python's logging module. You can enable this debug information by telling the root logger to print these messages:

```
import logging
logging.getLogger('').addHandler(logging.StreamHandler())
```

Sometimes, this additional logging is enough to help you debug. Before you ask for help, carefully read through your logs to see if there's anything there that helps you.

9.2 Gather Logs For Your Bug Report

If you still can't figure out what's going wrong, `tda-api` has special functionality for gathering and preparing logs for filing issues. It works by capturing `tda-api`'s logs, anonymizing them, and then dumping them to the console when the program exits. You can enable this by calling this method **before doing anything else in your application**:

```
tda.debug.enable_bug_report_logging()
```

This method will redact the logs to scrub them of common secrets, like account IDs, tokens, access keys, etc. However, this redaction is not guaranteed to be perfect, and it is your responsibility to make sure they are clean before you ask for help.

When filing a issue, please upload the logs along with your description. **If you do not include logs with your issue, your issue may be closed**.

For completeness, here is this method's documentation:

debug.**enable_bug_report_logging**()

Turns on bug report logging. Will collect all logged output, redact out anything that should be kept secret, and emit the result at program exit.

Notes:

- This method does a best effort redaction. Never share its output without verifying that all secret information is properly redacted.
- Because this function records all logged output, it has a performance penalty. It should not be called in production code.

9.3 Submit Your Ticket

You are now ready to write your bug. Before you do, be warned that your issue may be be closed if:

- It does not include code. The first thing we do when we receive your issue is we try to reproduce your failure. We can't do that if you don't show us your code.
- It does not include logs. It's very difficult to debug problems without logs.
- Logs are not adequately redacted. This is for your own protection.
- Logs are copy-pasted into the issue message field. Please write them to a file and attach them to your issue.
- You do not follow the issue template. We're not *super* strict about this one, but you should at least include all the information it asks for.

You can file an issue on our GitHub page.

CHAPTER 10

Contributing to `tda-api`

Fixing a bug? Adding a feature? Just cleaning up for the sake of cleaning up? Great! No improvement is too small for me, and I'm always happy to take pull requests. Read this guide to learn how to set up your environment so you can contribute.

10.1 Setting up the Dev Environment

Dependencies are listed in the *requirements.txt* file. These development requirements are distinct from the requirements listed in *setup.py* and include some additional packages around testing, documentation generation, etc.

Before you install anything, I highly recommend setting up a *virtualenv* so you don't pollute your system installation directories:

```
pip install virtualenv
virtualenv -v virtualenv
source virtualenv/bin/activate
```

Next, install project requirements:

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

Finally, verify everything works by running tests:

```
make test
```

At this point you can make your changes.

Note that if you are using a virtual environment and switch to a new terminal your virtual environment will not be active in the new terminal, and you need to run the activate command again. If you want to disable the loaded virtual environment in the same terminal window, use the command:

```
deactivate
```

10.2 Development Guidelines

10.2.1 Test your changes

This project aims for high test coverage. All changes must be properly tested, and we will accept no PRs that lack appropriate unit testing. We also expect existing tests to pass. You can run your tests using:

```
make test
```

10.2.2 Document your code

Documentation is how users learn to use your code, and no feature is complete without a full description of how to use it. If your PR changes external-facing interfaces, or if it alters semantics, the changes must be thoroughly described in the docstrings of the affected components. If your change adds a substantial new module, a new section in the documentation may be justified.

Documentation is built using Sphinx. You can build the documentation using the *Makefile.sphinx* makefile. For example you can build the HTML documentation like so:

```
make -f Makefile.sphinx
```

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Disclaimer: *tda-api is an unofficial API wrapper. It is in no way endorsed by or affiliated with TD Ameritrade or any associated organization. Make sure to read and understand the terms of service of the underlying API before using this package. This authors accept no responsibility for any damage that might stem from use of this package. See the LICENSE file for more details.*

Python Module Index

t

tda.auth, 4
tda.client, 8
tda.debug, 64
tda.orders, 42
tda.orders.generic, 47
tda.streaming, 20

Index

Symbols

__init__() (*tda.client.Client method*), 10
__init__() (*tda.utils.Utils method*), 61

A

ACCEPTED (*tda.client.Client.Order.Status attribute*), 11
Account (*class in tda.client.Client*), 12
ACCOUNT (*tda.streaming.StreamClient.AccountActivityFields attribute*), 42
Account.Fields (*class in tda.client.Client*), 12
account_activity_sub() (*tda.streaming.StreamClient method*), 42
add_account_activity_handler() (*tda.streaming.StreamClient method*), 42
add_chart_equity_handler() (*tda.streaming.StreamClient method*), 25
add_chart_futures_handler() (*tda.streaming.StreamClient method*), 26
add_equity_leg() (*tda.orders.generic.OrderBuilder method*), 54
add_level_one_equity_handler() (*tda.streaming.StreamClient method*), 27
add_level_one_forex_handler() (*tda.streaming.StreamClient method*), 35
add_level_one_futures_handler() (*tda.streaming.StreamClient method*), 32
add_level_one_futures_options_handler() (*tda.streaming.StreamClient method*), 36
add_level_one_option_handler() (*tda.streaming.StreamClient method*), 30
add_listed_book_handler() (*tda.streaming.StreamClient method*), 39
add_nasdaq_book_handler() (*tda.streaming.StreamClient method*), 39
add_news_headline_handler() (*tda.streaming.StreamClient method*), 41
add_option_leg() (*tda.orders.generic.OrderBuilder method*), 54
add_options_book_handler() (*tda.streaming.StreamClient method*), 39
add_timesale_equity_handler() (*tda.streaming.StreamClient method*), 40
add_timesale_futures_handler() (*tda.streaming.StreamClient method*), 40
add_timesale_options_handler() (*tda.streaming.StreamClient method*), 40
ADVISORY_FEES (*tda.client.Client.Transactions.TransactionType attribute*), 18
ALL (*tda.client.Client.Options.ContractType attribute*), 14
ALL (*tda.client.Client.Options.StrikeRange attribute*), 15
ALL (*tda.client.Client.Options.Type attribute*), 15
ALL (*tda.client.Client.Transactions.TransactionType attribute*), 18
ALL_OR_NONE (*tda.orders.common.SpecialInstruction attribute*), 55
ALL_OR_NONE_DO_NOT_REDUCE (*tda.orders.common.SpecialInstruction attribute*), 56
AM (*tda.orders.common.Session attribute*), 53
AMEX (*tda.orders.common.Destination attribute*), 55
ANALYTICAL (*tda.client.Client.Options.Strategy attribute*), 15
APRIL (*tda.client.Client.Options.ExpirationMonth attribute*), 14
ASK (*tda.orders.common.PriceLinkBasis attribute*), 59
ASK (*tda.orders.common.StopPriceLinkBasis attribute*), 58
ASK (*tda.orders.common.StopType attribute*), 59
ASK_BID (*tda.orders.common.PriceLinkBasis attribute*), 59
ASK_BID (*tda.orders.common.StopPriceLinkBasis attribute*), 58
ASK_ID (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
ASK_ID (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
ASK_ID (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37

ASK_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
ASK_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
ASK_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
ASK_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
ASK_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 30
ASK_SIZE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
ASK_SIZE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
ASK_SIZE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
ASK_SIZE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
ASK_SIZE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
AUGUST (*tda.client.Client.Options.ExpirationMonth attribute*), 14
AUTO (*tda.orders.common.Destination attribute*), 55
AVERAGE (*tda.orders.common.PriceLinkBasis attribute*), 59
AVERAGE (*tda.orders.common.StopPriceLinkBasis attribute*), 58
AWAITING_CONDITION (*tda.client.Client.Order.Status attribute*), 11
AWAITING_MANUAL_REVIEW (*tda.client.Client.Order.Status attribute*), 11
AWAITING_PARENT_ORDER (*tda.client.Client.Order.Status attribute*), 11
AWAITING_UR_OUR (*tda.client.Client.Order.Status attribute*), 11

B

BACK_RATIO (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
BASE (*tda.orders.common.PriceLinkBasis attribute*), 59
BASE (*tda.orders.common.StopPriceLinkBasis attribute*), 58
BATS (*tda.orders.common.Destination attribute*), 55
bear_call_vertical_close() (*in module tda.orders.options*), 46
bear_call_vertical_open() (*in module tda.orders.options*), 46
bear_put_vertical_close() (*in module tda.orders.options*), 46
bear_put_vertical_open() (*in module tda.orders.options*), 46
BID (*tda.orders.common.PriceLinkBasis attribute*), 59
BID (*tda.orders.common.StopPriceLinkBasis attribute*), 58
BID (*tda.orders.common.StopType attribute*), 59
BID_ID (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
BID_ID (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
BID_ID (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
BID_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
BID_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
BID_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
BID_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
BID_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 30
BID_SIZE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
BID_SIZE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
BID_SIZE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
BID_SIZE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
BID_SIZE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
BID_TICK (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
BOND (*tda.client.Client.Markets attribute*), 19
BOX (*tda.orders.common.Destination attribute*), 55
bull_call_vertical_close() (*in module tda.orders.options*), 46
bull_call_vertical_open() (*in module tda.orders.options*), 46
bull_put_vertical_close() (*in module tda.orders.options*), 46
bull_put_vertical_open() (*in module tda.orders.options*), 46
BUTTERFLY (*tda.client.Client.Options.Strategy attribute*), 15
BUTTERFLY (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
BUY (*tda.orders.common.EquityInstruction attribute*), 54
BUY_ONLY (*tda.client.Client.Transactions.TransactionType attribute*), 18
BUY_TO_CLOSE (*tda.orders.common.OptionInstruction attribute*), 55
BUY_TO_COVER (*tda.orders.common.EquityInstruction attribute*), 54
BUY_TO_OPEN (*tda.orders.common.OptionInstruction*

attribute), 55

C

C2 (*tda.orders.common.Destination attribute*), 55
CALENDAR (*tda.client.Client.Options.Strategy attribute*), 15
CALENDAR (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
CALL (*tda.client.Client.Options.ContractType attribute*), 14
cancel_order() (*tda.client.Client method*), 12
CANCELLED (*tda.client.Client.Order.Status attribute*), 12
CASH_IN_OR_CASH_OUT (*tda.client.Client.Transactions.TransactionType attribute*), 18
CBOE (*tda.orders.common.Destination attribute*), 55
CHART_DAY (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
chart_equity_add() (*tda.streaming.StreamClient method*), 25
chart_equity_subs() (*tda.streaming.StreamClient method*), 25
chart_futures_add() (*tda.streaming.StreamClient method*), 26
chart_futures_subs() (*tda.streaming.StreamClient method*), 26
CHART_TIME (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
CHART_TIME (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
CHECKING (*tda.client.Client.Transactions.TransactionType attribute*), 18
clear_activation_price() (*tda.orders.generic.OrderBuilder method*), 60
clear_complex_order_strategy_type() (*tda.orders.generic.OrderBuilder method*), 57
clear_duration() (*tda.orders.generic.OrderBuilder method*), 53
clear_order_legs() (*tda.orders.generic.OrderBuilder method*), 55
clear_order_strategy_type() (*tda.orders.generic.OrderBuilder method*), 57
clear_order_type() (*tda.orders.generic.OrderBuilder method*), 53
clear_price() (*tda.orders.generic.OrderBuilder method*), 54
clear_price_link_basis() (*tda.orders.generic.OrderBuilder method*), 60
clear_price_link_type() (*tda.orders.generic.OrderBuilder method*), 60
clear_quantity() (*tda.orders.generic.OrderBuilder method*), 58
clear_requested_destination() (*tda.orders.generic.OrderBuilder method*), 55
clear_session() (*tda.orders.generic.OrderBuilder method*), 53
clear_special_instruction() (*tda.orders.generic.OrderBuilder method*), 56
clear_stop_price() (*tda.orders.generic.OrderBuilder method*), 58
clear_stop_price_link_basis() (*tda.orders.generic.OrderBuilder method*), 59
clear_stop_price_link_type() (*tda.orders.generic.OrderBuilder method*), 59
clear_stop_price_offset() (*tda.orders.generic.OrderBuilder method*), 59
clear_stop_type() (*tda.orders.generic.OrderBuilder method*), 59
client_from_login_flow() (*in module tda.auth*), 6
client_from_token_file() (*in module tda.auth*), 6
CLOSE_PRICE (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
CLOSE_PRICE (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
CLOSE_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
CLOSE_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
CLOSE_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
CLOSE_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFiel attribute*), 37
CLOSE_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
COLLAR (*tda.client.Client.Options.Strategy attribute*), 15
COLLAR_SYNTHETIC (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
COLLAR_WITH_STOCK (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
ComplexOrderStrategyType (*class in tda.orders.common*), 56

CONDOR (*tda.client.Client.Options.Strategy attribute*), 15
CONDOR (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
CONTRACT_TYPE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
COUNT_FOR_KEYWORD (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
COVERED (*tda.client.Client.Options.Strategy attribute*), 15
COVERED (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
create_saved_order() (*tda.client.Client method*), 18
create_watchlist() (*tda.client.Client method*), 20
CUSTOM (*tda.orders.common.ComplexOrderStrategyType attribute*), 57

D

DAILY (*tda.client.Client.PriceHistory.Frequency attribute*), 16
DAILY (*tda.client.Client.PriceHistory.FrequencyType attribute*), 16
DAY (*tda.client.Client.PriceHistory.PeriodType attribute*), 17
DAY (*tda.orders.common.Duration attribute*), 53
DAYS_TO_EXPIRATION (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
DECEMBER (*tda.client.Client.Options.ExpirationMonth attribute*), 14
DELAYED (*tda.streaming.StreamClient.QOSLevel attribute*), 23
delete_saved_order() (*tda.client.Client method*), 18
delete_watchlist() (*tda.client.Client method*), 20
DELIVERABLES (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
DELTA (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
DESC_REGEX (*tda.client.Client.Instrument.Projection attribute*), 13
DESC_SEARCH (*tda.client.Client.Instrument.Projection attribute*), 13
DESCRIPTION (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
DESCRIPTION (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
DESCRIPTION (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
DESCRIPTION (*tda.streaming.StreamClient.LevelOneFuturesOptionFields attribute*), 37
DESCRIPTION (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 30
Destination (*class in tda.orders.common*), 55
DIAGONAL (*tda.client.Client.Options.Strategy attribute*), 15
DIAGONAL (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
DIGITS (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
DIGITS (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
DIGITS (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
DIVIDEND (*tda.client.Client.Transactions.TransactionType attribute*), 18
DIVIDEND_AMOUNT (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
DIVIDEND_DATE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
DIVIDEND_YIELD (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
DO_NOT_REDUCE (*tda.orders.common.SpecialInstruction attribute*), 56
DOUBLE_DIAGONAL (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
DOWN (*tda.client.Client.Movers.Direction attribute*), 19
Duration (*class in tda.orders.common*), 53

E

easy_client() (*in module tda.auth*), 6
ECN_ARCA (*tda.orders.common.Destination attribute*), 55
enable_bug_report_logging() (*tda.debug method*), 66
EQUITY (*tda.client.Client.Markets attribute*), 19
equity_buy_limit() (*in module tda.orders.equities*), 44
equity_buy_market() (*in module tda.orders.equities*), 44
equity_buy_to_cover_limit() (*in module tda.orders.equities*), 44
equity_buy_to_cover_market() (*in module tda.orders.equities*), 44
equity_sell_limit() (*in module tda.orders.equities*), 44
equity_sell_market() (*in module tda.orders.equities*), 44
equity_sell_short_limit() (*in module tda.orders.equities*), 44
equity_sell_short_market() (*in module tda.orders.equities*), 44
EquityInstruction (*class in tda.orders.common*), 54
ERROR_CODE (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41

EVERY_FIFTEEN_MINUTES (*tda.client.Client.PriceHistory.Frequency attribute*), 16
EVERY_FIVE_MINUTES (*tda.client.Client.PriceHistory.Frequency attribute*), 16
EVERY_MINUTE (*tda.client.Client.PriceHistory.Frequency attribute*), 16
EVERY_TEN_MINUTES (*tda.client.Client.PriceHistory.Frequency attribute*), 16
EVERY_THIRTY_MINUTES (*tda.client.Client.PriceHistory.Frequency attribute*), 16
EXCHANGE_ID (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
EXCHANGE_ID (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
EXCHANGE_ID (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
EXCHANGE_ID (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
EXCHANGE_NAME (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
EXCHANGE_NAME (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
EXCHANGE_NAME (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
EXCHANGE_NAME (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
EXERCISE (*tda.orders.common.OrderType attribute*), 52
EXPIRATION_DAY (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
EXPIRATION_MONTH (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
EXPIRATION_YEAR (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
EXPIRED (*tda.client.Client.Order.Status attribute*), 12
EXPRESS (*tda.streaming.StreamClient.QOSLevel attribute*), 22
extract_order_id() (*tda.utils.Utils method*), 61

F

FAST (*tda.streaming.StreamClient.QOSLevel attribute*), 22
FEBRUARY (*tda.client.Client.Options.ExpirationMonth attribute*), 14
FIFTEEN_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 16
FILL_OR_KILL (*tda.orders.common.Duration attribute*), 53
FILLED (*tda.client.Client.Order.Status attribute*), 12
first_triggers_second() (*in module tda.orders.common*), 46
FIVE_DAYS (*tda.client.Client.PriceHistory.Period attribute*), 16
FIVE_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 16
FOREX (*tda.client.Client.Markets attribute*), 19
FOUR_DAYS (*tda.client.Client.PriceHistory.Period attribute*), 16
FUND_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
FUNDAMENTAL (*tda.client.Client.Instrument.Projection attribute*), 13
FUTURE (*tda.client.Client.Markets attribute*), 19
FUTURE_ACTIVE_SYMBOL (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_ACTIVE_SYMBOL (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_EXPIRATION_DATE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_EXPIRATION_DATE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_IS_ACTIVE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_IS_ACTIVE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_IS_TRADEABLE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_IS_TRADEABLE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_MULTIPLIER (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_MULTIPLIER (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_PERCENT_CHANGE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_PERCENT_CHANGE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
FUTURE_PRICE_FORMAT (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_PRICE_FORMAT (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_SETTLEMENT_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields*

attribute), 34
FUTURE_SETTLEMENT_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
FUTURE_TRADING_HOURS (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
FUTURE_TRADING_HOURS (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38

G

GAMMA (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
get_account() (*tda.client.Client method*), 12
get_accounts() (*tda.client.Client method*), 12
get_hours_for_multiple_markets() (*tda.client.Client method*), 19
get_hours_for_single_market() (*tda.client.Client method*), 19
get_instrument() (*tda.client.Client method*), 13
get_movers() (*tda.client.Client method*), 19
get_option_chain() (*tda.client.Client method*), 13
get_order() (*tda.client.Client method*), 11
get_orders_by_path() (*tda.client.Client method*), 11
get_orders_by_query() (*tda.client.Client method*), 11
get_preferences() (*tda.client.Client method*), 20
get_price_history() (*tda.client.Client method*), 15
get_quote() (*tda.client.Client method*), 17
get_quotes() (*tda.client.Client method*), 17
get_saved_order() (*tda.client.Client method*), 18
get_saved_orders_by_path() (*tda.client.Client method*), 18
get_transaction() (*tda.client.Client method*), 17
get_transactions() (*tda.client.Client method*), 17
get_user_principals() (*tda.client.Client method*), 20
get_watchlist() (*tda.client.Client method*), 20
get_watchlists_for_multiple_accounts() (*tda.client.Client method*), 20
get_watchlists_for_single_account() (*tda.client.Client method*), 20
GOOD_TILL_CANCEL (*tda.orders.common.Duration attribute*), 53

H

HEADLINE (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
HEADLINE_ID (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
HIGH_52_WEEK (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
HIGH_52_WEEK (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
HIGH_PRICE (*tda.streaming.StreamClient.ChartEquityFields attribute*), 25
HIGH_PRICE (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
HIGH_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
HIGH_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
HIGH_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
HIGH_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
HIGH_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31

I

IN_THE_MONEY (*tda.client.Client.Options.StrikeRange attribute*), 15
INET (*tda.orders.common.Destination attribute*), 55
Instrument (*class in tda.client.Client*), 13
Instrument.Projection (*class in tda.client.Client*), 13
INTEREST (*tda.client.Client.Transactions.TransactionType attribute*), 18
IRON_CONDOR (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
IS_HOT (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
IS_REGULAR_MARKET_QUOTE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
IS_REGULAR_MARKET_TRADE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
IS_TRADABLE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
ISE (*tda.orders.common.Destination attribute*), 55
ISLAND_ASK_DEPRECATED (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
ISLAND_ASK_SIZE_DEPRECATED (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
ISLAND_BID_DEPRECATED (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
ISLAND_BID_SIZE_DEPRECATED (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
ISLAND_VOLUME_DEPRECATED

(*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28

J

JANUARY (*tda.client.Client.Options.ExpirationMonth attribute*), 14
JULY (*tda.client.Client.Options.ExpirationMonth attribute*), 14
JUN (*tda.client.Client.Options.ExpirationMonth attribute*), 14

K

KEYWORD_ARRAY (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41

L

LAST (*tda.orders.common.PriceLinkBasis attribute*), 59
LAST (*tda.orders.common.StopPriceLinkBasis attribute*), 58
LAST (*tda.orders.common.StopType attribute*), 59
LAST_ID (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
LAST_ID (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
LAST_ID (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
LAST_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
LAST_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
LAST_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
LAST_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
LAST_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
LAST_PRICE (*tda.streaming.StreamClient.TimesaleFields attribute*), 40
LAST_SEQUENCE (*tda.streaming.StreamClient.TimesaleFields attribute*), 40
LAST_SIZE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
LAST_SIZE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
LAST_SIZE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
LAST_SIZE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
LAST_SIZE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
LAST_SIZE (*tda.streaming.StreamClient.TimesaleFields attribute*), 40
level_one_equity_subs() (*tda.streaming.StreamClient method*), 27
level_one_forex_subs() (*tda.streaming.StreamClient method*), 35
level_one_futures_options_subs() (*tda.streaming.StreamClient method*), 36
level_one_futures_subs() (*tda.streaming.StreamClient method*), 32
level_one_option_subs() (*tda.streaming.StreamClient method*), 30
LIMIT (*tda.orders.common.OrderType attribute*), 52
listed_book_subs() (*tda.streaming.StreamClient method*), 39
login() (*tda.streaming.StreamClient method*), 22
LOW_52_WEEK (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
LOW_52_WEEK (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
LOW_PRICE (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
LOW_PRICE (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
LOW_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
LOW_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
LOW_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
LOW_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
LOW_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31

M

MANUAL (*tda.orders.common.PriceLinkBasis attribute*), 59
MANUAL (*tda.orders.common.StopPriceLinkBasis attribute*), 58
MARCH (*tda.client.Client.Options.ExpirationMonth attribute*), 14
MARGINABLE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
MARK (*tda.orders.common.PriceLinkBasis attribute*), 59
MARK (*tda.orders.common.StopPriceLinkBasis attribute*), 58
MARK (*tda.orders.common.StopType attribute*), 59
MARK (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
MARK (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
MARK (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
MARK (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
MARK (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32

MARKET (*tda.orders.common.OrderType attribute*), 52
MARKET_MAKER (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
MARKET_ON_CLOSE (*tda.orders.common.OrderType attribute*), 52
Markets (*class in tda.client.Client*), 19
MAY (*tda.client.Client.Options.ExpirationMonth attribute*), 14
MESSAGE_DATA (*tda.streaming.StreamClient.AccountActivityFields attribute*), 42
MESSAGE_TYPE (*tda.streaming.StreamClient.AccountActivityFields attribute*), 42
MINUTE (*tda.client.Client.PriceHistory.FrequencyType attribute*), 16
MODERATE (*tda.streaming.StreamClient.QOSLevel attribute*), 22
MONEY_INTRINSIC_VALUE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
MONTH (*tda.client.Client.PriceHistory.PeriodType attribute*), 17
MONTHLY (*tda.client.Client.PriceHistory.Frequency attribute*), 16
MONTHLY (*tda.client.Client.PriceHistory.FrequencyType attribute*), 16
Movers (*class in tda.client.Client*), 19
Movers.Change (*class in tda.client.Client*), 19
Movers.Direction (*class in tda.client.Client*), 19
MULTIPLIER (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31

N

NASDAQ (*tda.orders.common.Destination attribute*), 55
nasdaq_book_subs() (*tda.streaming.StreamClient method*), 39
NAV (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
NEAR_THE_MONEY (*tda.client.Client.Options.StrikeRange attribute*), 15
NET_CHANGE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
NET_CHANGE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
NET_CHANGE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
NET_CHANGE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
NET_CHANGE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
NET_CREDIT (*tda.orders.common.OrderType attribute*), 52
NET_DEBIT (*tda.orders.common.OrderType attribute*), 52
NET_ZERO (*tda.orders.common.OrderType attribute*), 52
news_headline_subs() (*tda.streaming.StreamClient method*), 41
NON_STANDARD (*tda.client.Client.Options.Type attribute*), 15
NONE (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
NORMAL (*tda.orders.common.Session attribute*), 53
NOVEMBER (*tda.client.Client.Options.ExpirationMonth attribute*), 14
NYSE (*tda.orders.common.Destination attribute*), 55

O

OCO (*tda.orders.common.OrderStrategyType attribute*), 57
OCTOBER (*tda.client.Client.Options.ExpirationMonth attribute*), 15
one_cancels_other() (*in module tda.orders.common*), 46
ONE_DAY (*tda.client.Client.PriceHistory.Period attribute*), 16
ONE_MONTH (*tda.client.Client.PriceHistory.Period attribute*), 16
ONE_YEAR (*tda.client.Client.PriceHistory.Period attribute*), 17
OPEN_INTEREST (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
OPEN_INTEREST (*tda.streaming.StreamClient.LevelOneFuturesOptionsF attribute*), 38
OPEN_INTEREST (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
OPEN_PRICE (*tda.streaming.StreamClient.ChartEquityFields attribute*), 25
OPEN_PRICE (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
OPEN_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
OPEN_PRICE (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
OPEN_PRICE (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
OPEN_PRICE (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
OPEN_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
OPTION (*tda.client.Client.Markets attribute*), 19
option_buy_to_close_limit() (*in module tda.orders.options*), 45
option_buy_to_close_market() (*in module tda.orders.options*), 45
option_buy_to_open_limit() (*in module tda.orders.options*), 45
option_buy_to_open_market() (*in module tda.orders.options*), 45

option_sell_to_close_limit() (*in module tda.orders.options*), 45
option_sell_to_close_market() (*in module tda.orders.options*), 45
option_sell_to_open_limit() (*in module tda.orders.options*), 45
option_sell_to_open_market() (*in module tda.orders.options*), 45
OptionInstruction (*class in tda.orders.common*), 54
Options (*class in tda.client.Client*), 14
Options.ContractType (*class in tda.client.Client*), 14
Options.ExpirationMonth (*class in tda.client.Client*), 14
Options.Strategy (*class in tda.client.Client*), 15
Options.StrikeRange (*class in tda.client.Client*), 15
Options.Type (*class in tda.client.Client*), 15
options_book_subs() (*tda.streaming.StreamClient method*), 39
Order (*class in tda.client.Client*), 11
Order.Status (*class in tda.client.Client*), 11
ORDERS (*tda.client.Client.Account.Fields attribute*), 13
OrderStrategyType (*class in tda.orders.common*), 57
OrderType (*class in tda.orders.common*), 52
OTHER (*tda.client.Client.Transactions.TransactionType attribute*), 18
OUT_OF_THE_MONEY (*tda.client.Client.Options.StrikeRange attribute*), 15

P

PE_RATIO (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
PENDING_ACTIVATION (*tda.client.Client.Order.Status attribute*), 12
PENDING_CANCEL (*tda.client.Client.Order.Status attribute*), 12
PENDING_REPLACE (*tda.client.Client.Order.Status attribute*), 12
PERCENT (*tda.client.Client.Movers.Change attribute*), 19
PERCENT (*tda.orders.common.PriceLinkType attribute*), 60
PERCENT (*tda.orders.common.StopPriceLinkType attribute*), 59
PHLX (*tda.orders.common.Destination attribute*), 55
place_order() (*tda.client.Client method*), 10
PM (*tda.orders.common.Session attribute*), 53
POSITIONS (*tda.client.Client.Account.Fields attribute*), 13
PREFERENCES (*tda.client.Client.UserPrincipals.Fields attribute*), 20
PriceHistory (*class in tda.client.Client*), 16
PriceHistory.Frequency (*class in tda.client.Client*), 16
PriceHistory.FrequencyType (*class in tda.client.Client*), 16
PriceHistory.Period (*class in tda.client.Client*), 16
PriceHistory.PeriodType (*class in tda.client.Client*), 17
PriceLinkBasis (*class in tda.orders.common*), 59
PriceLinkType (*class in tda.orders.common*), 60
PRODUCT (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
PRODUCT (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
PRODUCT (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
PUT (*tda.client.Client.Options.ContractType attribute*), 14

Q

quality_of_service() (*tda.streaming.StreamClient method*), 22
QUEUED (*tda.client.Client.Order.Status attribute*), 12
QUOTE_DAY (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
QUOTE_DAY (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
QUOTE_TIME (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
QUOTE_TIME (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
QUOTE_TIME (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
QUOTE_TIME (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
QUOTE_TIME (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
QUOTE_TIME_IN_LONG (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30

R

REAL_TIME (*tda.streaming.StreamClient.QOSLevel attribute*), 22
REGULAR_MARKET_LAST_PRICE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 29
REGULAR_MARKET_LAST_SIZE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30

REGULAR_MARKET_NET_CHANGE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
REGULAR_MARKET_TRADE_DAY (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
REGULAR_MARKET_TRADE_TIME (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
REGULAR_MARKET_TRADE_TIME_IN_LONG (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
REJECTED (*tda.client.Client.Order.Status attribute*), 12
replace_order() (*tda.client.Client method*), 12
replace_saved_order() (*tda.client.Client method*), 18
replace_watchlist() (*tda.client.Client method*), 20
REPLACED (*tda.client.Client.Order.Status attribute*), 12
RHO (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
ROLL (*tda.client.Client.Options.Strategy attribute*), 15

S

SEAMESS (*tda.orders.common.Session attribute*), 53
search_instruments() (*tda.client.Client method*), 13
SECURITY_STATUS (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
SECURITY_STATUS (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
SECURITY_STATUS (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
SECURITY_STATUS (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
SECURITY_STATUS (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
SELL (*tda.orders.common.EquityInstruction attribute*), 54
SELL_ONLY (*tda.client.Client.Transactions.TransactionType attribute*), 18
SELL_SHORT (*tda.orders.common.EquityInstruction attribute*), 54
SELL_TO_CLOSE (*tda.orders.common.OptionInstruction attribute*), 55
SELL_TO_OPEN (*tda.orders.common.OptionInstruction attribute*), 55
SEPTEMBER (*tda.client.Client.Options.ExpirationMonth attribute*), 15
SEQUENCE (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
Session (*class in tda.orders.common*), 53
set_account_id() (*tda.utils.Utils method*), 61
set_activation_price() (*tda.orders.generic.OrderBuilder method*), 60
set_complex_order_strategy_type() (*tda.orders.generic.OrderBuilder method*), 57
set_duration() (*tda.orders.generic.OrderBuilder method*), 53
set_order_strategy_type() (*tda.orders.generic.OrderBuilder method*), 57
set_order_type() (*tda.orders.generic.OrderBuilder method*), 53
set_price() (*tda.orders.generic.OrderBuilder method*), 54
set_price_link_basis() (*tda.orders.generic.OrderBuilder method*), 59
set_price_link_type() (*tda.orders.generic.OrderBuilder method*), 60
set_quantity() (*tda.orders.generic.OrderBuilder method*), 58
set_requested_destination() (*tda.orders.generic.OrderBuilder method*), 55
set_session() (*tda.orders.generic.OrderBuilder method*), 53
set_special_instruction() (*tda.orders.generic.OrderBuilder method*), 56
set_stop_price() (*tda.orders.generic.OrderBuilder method*), 58
set_stop_price_link_basis() (*tda.orders.generic.OrderBuilder method*), 58
set_stop_price_link_type() (*tda.orders.generic.OrderBuilder method*), 59
set_stop_price_offset() (*tda.orders.generic.OrderBuilder method*), 59
set_stop_type() (*tda.orders.generic.OrderBuilder method*), 59
SHORTABLE (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
SINGLE (*tda.client.Client.Options.Strategy attribute*), 15
SINGLE (*tda.orders.common.OrderStrategyType attribute*), 57
SIX_MONTHS (*tda.client.Client.PriceHistory.Period attribute*), 17
SLOW (*tda.streaming.StreamClient.QOSLevel attribute*), 22
SpecialInstruction (*class in tda.orders.common*),

55
STANDARD (*tda.client.Client.Options.Type attribute*), 15
STANDARD (*tda.orders.common.StopType attribute*), 59
STATUS (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
STOP (*tda.orders.common.OrderType attribute*), 52
STOP_LIMIT (*tda.orders.common.OrderType attribute*), 52
StopPriceLinkBasis (*class in tda.orders.common*), 58
StopPriceLinkType (*class in tda.orders.common*), 59
StopType (*class in tda.orders.common*), 59
STORY_DATETIME (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
STORY_ID (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
STORY_SOURCE (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
STRADDLE (*tda.client.Client.Options.Strategy attribute*), 15
STRADDLE (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
STRANGLE (*tda.client.Client.Options.Strategy attribute*), 15
STRANGLE (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
StreamClient.AccountActivityFields (*class in tda.streaming*), 42
StreamClient.ChartEquityFields (*class in tda.streaming*), 25
StreamClient.ChartFuturesFields (*class in tda.streaming*), 26
StreamClient.LevelOneEquityFields (*class in tda.streaming*), 27
StreamClient.LevelOneForexFields (*class in tda.streaming*), 35
StreamClient.LevelOneFuturesFields (*class in tda.streaming*), 33
StreamClient.LevelOneFuturesOptionsFields (*class in tda.streaming*), 36
StreamClient.LevelOneOptionFields (*class in tda.streaming*), 30
StreamClient.NewsHeadlineFields (*class in tda.streaming*), 41
StreamClient.QOSLevel (*class in tda.streaming*), 22
StreamClient.TimesaleFields (*class in tda.streaming*), 39
STREAMER_CONNECTION_INFO (*tda.client.Client.UserPrincipals.Fields attribute*), 20
STREAMER_SUBSCRIPTION_KEYS (*tda.client.Client.UserPrincipals.Fields attribute*), 20
STRIKE_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
STRIKES_ABOVE_MARKET (*tda.client.Client.Options.StrikeRange attribute*), 15
STRIKES_BELOW_MARKET (*tda.client.Client.Options.StrikeRange attribute*), 15
STRIKES_NEAR_MARKET (*tda.client.Client.Options.StrikeRange attribute*), 15
SUBSCRIPTION_KEY (*tda.streaming.StreamClient.AccountActivityFields attribute*), 42
SURROGATE_IDS (*tda.client.Client.UserPrincipals.Fields attribute*), 20
SYMBOL (*tda.streaming.StreamClient.ChartEquityFields attribute*), 25
SYMBOL (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 26
SYMBOL (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
SYMBOL (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
SYMBOL (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
SYMBOL (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
SYMBOL (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 30
SYMBOL (*tda.streaming.StreamClient.NewsHeadlineFields attribute*), 41
SYMBOL (*tda.streaming.StreamClient.TimesaleFields attribute*), 40
SYMBOL_REGEX (*tda.client.Client.Instrument.Projection attribute*), 13
SYMBOL_SEARCH (*tda.client.Client.Instrument.Projection attribute*), 13

T

tda.auth (*module*), 4
tda.client (*module*), 8
tda.debug (*module*), 64
tda.orders (*module*), 42
tda.orders.generic (*module*), 47
tda.streaming (*module*), 20
TEN_DAYS (*tda.client.Client.PriceHistory.Period attribute*), 17
TEN_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 17
THEORETICAL_OPTION_VALUE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32

THETA (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
THREE_DAYS (*tda.client.Client.PriceHistory.Period attribute*), 17
THREE_MONTHS (*tda.client.Client.PriceHistory.Period attribute*), 17
THREE_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 17
TICK (*tda.orders.common.PriceLinkType attribute*), 60
TICK (*tda.orders.common.StopPriceLinkType attribute*), 59
TICK (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
TICK (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
TICK (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
TICK_AMOUNT (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
TICK_AMOUNT (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 34
TICK_AMOUNT (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 38
TIME_VALUE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
timesale_equity_subs() (*tda.streaming.StreamClient method*), 40
timesale_futures_subs() (*tda.streaming.StreamClient method*), 40
timesale_options_subs() (*tda.streaming.StreamClient method*), 40
TOTAL_VOLUME (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 27
TOTAL_VOLUME (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
TOTAL_VOLUME (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
TOTAL_VOLUME (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
TOTAL_VOLUME (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
TRADE (*tda.client.Client.Transactions.TransactionType attribute*), 18
TRADE_DAY (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
TRADE_DAY (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
TRADE_TIME (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
TRADE_TIME (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 35
TRADE_TIME (*tda.streaming.StreamClient.LevelOneFuturesFields attribute*), 33
TRADE_TIME (*tda.streaming.StreamClient.LevelOneFuturesOptionsFields attribute*), 37
TRADE_TIME (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
TRADE_TIME (*tda.streaming.StreamClient.TimesaleFields attribute*), 40
TRADE_TIME_IN_LONG (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 30
TRADING_HOURS (*tda.streaming.StreamClient.LevelOneForexFields attribute*), 36
TRAILING_STOP (*tda.orders.common.OrderType attribute*), 52
TRAILING_STOP_LIMIT (*tda.orders.common.OrderType attribute*), 52
Transactions (*class in tda.client.Client*), 18
Transactions.TransactionType (*class in tda.client.Client*), 18
TRIGGER (*tda.orders.common.OrderStrategyType attribute*), 57
TRIGGER (*tda.orders.common.PriceLinkBasis attribute*), 59
TRIGGER (*tda.orders.common.StopPriceLinkBasis attribute*), 58
TWENTY_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 17
TWO_DAYS (*tda.client.Client.PriceHistory.Period attribute*), 17
TWO_MONTHS (*tda.client.Client.PriceHistory.Period attribute*), 17
TWO_YEARS (*tda.client.Client.PriceHistory.Period attribute*), 17

U

UNBALANCED_BUTTERFLY (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
UNBALANCED_CONDOR (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
UNBALANCED_IRON_CONDOR (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
UNBALANCED_VERTICAL_ROLL (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
UNDERLYING (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
UNDERLYING_PRICE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
UP (*tda.client.Client.Movers.Direction attribute*), 19
update_preferences() (*tda.client.Client method*), 20
update_watchlist() (*tda.client.Client method*), 20

UserPrincipals (*class in tda.client.Client*), 20
UserPrincipals.Fields (*class in tda.client.Client*), 20
Utils (*class in tda.utils*), 61
UV_EXPIRATION_TYPE (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32

V

VALUE (*tda.client.Client.Movers.Change attribute*), 19
VALUE (*tda.orders.common.PriceLinkType attribute*), 60
VALUE (*tda.orders.common.StopPriceLinkType attribute*), 59
VEGA (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 32
VERTICAL (*tda.client.Client.Options.Strategy attribute*), 15
VERTICAL (*tda.orders.common.ComplexOrderStrategyType attribute*), 56
VERTICAL_ROLL (*tda.orders.common.ComplexOrderStrategyType attribute*), 57
VOLATILITY (*tda.streaming.StreamClient.LevelOneEquityFields attribute*), 28
VOLATILITY (*tda.streaming.StreamClient.LevelOneOptionFields attribute*), 31
VOLUME (*tda.streaming.StreamClient.ChartEquityFields attribute*), 26
VOLUME (*tda.streaming.StreamClient.ChartFuturesFields attribute*), 27

W

WEEKLY (*tda.client.Client.PriceHistory.Frequency attribute*), 16
WEEKLY (*tda.client.Client.PriceHistory.FrequencyType attribute*), 16
WORKING (*tda.client.Client.Order.Status attribute*), 12

Y

YEAR (*tda.client.Client.PriceHistory.PeriodType attribute*), 17
YEAR_TO_DATE (*tda.client.Client.PriceHistory.Period attribute*), 17
YEAR_TO_DATE (*tda.client.Client.PriceHistory.PeriodType attribute*), 17