
tda-api

May 12, 2020

Contents:

1	Getting Started	3
1.1	TD Ameritrade API Access	3
1.2	Installing <code>tda-api</code>	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	Client Wrapper	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 Chains	13
3.8	Price History	15
3.9	Current Quotes	17
3.10	Other Endpoints	17
4	Creating Order Specifications	21
4.1	Common Values	21
4.2	Equity Orders	22
5	Utilities	25
5.1	Get the Most Recent Order	25
6	Example Application	27
7	Contributing to <code>tda-api</code>	29
7.1	Setting up the Dev Environment	29
7.2	Development Guidelines	30
8	Indices and tables	31
	Python Module Index	33

This project is on [GitHub](#)

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 directory. 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*.

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 another's 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 *Client Wrapper*, ready to send API calls. It also handles token refreshing, and writes updated tokens to the token file.

```
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. 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 the specified token path. The session will perform an auth refresh as needed. It will also update the token on disk whenever appropriate.

Parameters

- **token_path** – Path to the token. Updated tokens will be written to this path.
- **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`.

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. 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, please file a ticket on our [issues](#) page.

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:

 A third-party application may be attempting to make unauthorized access to your account. For help with your account, [contact us](#). For information on securing your account, visit the [TD Ameritrade Security 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`.

CHAPTER 3

Client Wrapper

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 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.

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 [Creating Order Specifications](#), 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 Chains

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 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.](#)

Creating Order Specifications

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.

The `tda.orders` module provides an incomplete set of helpers for building these order specs. The aim is to make it impossible to build an invalid JSON object. For example, here is how you might use this module to place a market order for ten shares of Apple common stock:

```
from tda.orders import EquityOrderBuilder, Duration, Session

builder = EquityOrderBuilder('AAPL', 10)
builder.set_instruction(EquityOrderBuilder.Instruction.SELL)
builder.set_order_type(EquityOrderBuilder.OrderType.MARKET)
builder.set_duration(Duration.DAY)
builder.set_session(Session.NORMAL)

client = ... # Get a client however you see fit
account_id = 12345678

resp = client.place_order(account_id, builder.build())
assert resp.ok
```

4.1 Common Values

```
class tda.orders.Duration
    An enumeration.

    DAY = 'DAY'

    FILL_OR_KILL = 'FILL_OR_KILL'

    GOOD_TILL_CANCEL = 'GOOD_TILL_CANCEL'
```

```
class tda.orders.Session
    An enumeration.

    AM = 'AM'

    NORMAL = 'NORMAL'

    PM = 'PM'

    SEAMESS = 'SEAMLESS'

exception tda.orders.InvalidOrderException
    Raised when attempting to build an incomplete order
```

4.2 Equity Orders

```
class tda.orders.EquityOrderBuilder(symbol, quantity)
    Helper class to construct equity orders.

    __init__(symbol, quantity)
        Create an order for the given symbol and quantity. Note all unspecified parameters must be set prior to
        building the order spec.

        Parameters
            • symbol – Symbol for the order
            • quantity – Quantity of the order

    class Instruction
        Order instruction

        BUY = 'BUY'

        SELL = 'SELL'

    class OrderType
        Order type

        LIMIT = 'LIMIT'

        MARKET = 'MARKET'

    build()
        Build the order spec.

        Raises InvalidOrderException – if the order is not fully specified

    matches(order)
        Takes a real object, as might be returned from the TD Ameritrade API, and indicates whether this order
        object matches it. Returns true if the given order if the given order could have been placed by calling
        Client.place_order() with this order.

        This method may be called on incomplete orders builders (builders whose build() method would fail if
        called. In such a case, unset values are ignored and have no effect on filtering.

    set_duration(duration)
        Set the order duration

    set_instruction(instruction)
        Set the order instruction
```

set_order_type (*order_type*)

Set the order type

set_price (*price*)

Set the order price. Must be set for LIMIT orders.

set_session (*session*)

Set the order's session

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.

5.1 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 headers. 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.

For orders that were rejected or whose order responses for whatever other reason might not contain the order ID, we can do a best-effort lookup using this method:

`Utils.find_most_recent_order(*, symbol=None, quantity=None, instruction=None, order_type=None, lookback_window=datetime.timedelta(days=1))`

When placing orders, the TDA API does not always return the order ID of the newly placed order, especially when the order was rejected. This means if we want to make extra sure of its status, we have to take a guess as to which order we just placed. This method simplifies things by returning the most recently-placed order with the given order signature.

Note: This method cannot guarantee that the calling process was the one which placed an order. This means that if there are multiple sources of orders, this method may return an order which was placed by another process.

Parameters

- **symbol** – Limit search to orders for this symbol.
- **quantity** – Limit search to orders of this quantity.
- **instruction** – Limit search to orders with this instruction. See `tda.orders.EquityOrderBuilder.Instruction`
- **order_type** – Limit search to orders with this order type. See `tda.orders.EquityOrderBuilder.OrderType`
- **lookback_window** – Limit search to orders entered less than this long ago. Note the TDA API does not provide orders older than 60 days.

CHAPTER 6

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@GAMER.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()

```

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.

7.1 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/build/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.

7.2 Development Guidelines

7.2.1 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
```

7.2.2 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
```

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.

t

`tda.auth`, 4
`tda.client`, 8

Symbols

`__init__()` (*tda.client.Client* method), 10
`__init__()` (*tda.orders.EquityOrderBuilder* method), 22
`__init__()` (*tda.utils.Utils* method), 25

A

ACCEPTED (*tda.client.Client.Order.Status* attribute), 11
Account (class in *tda.client.Client*), 12
Account.Fields (class in *tda.client.Client*), 12
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
AM (*tda.orders.Session* attribute), 22
ANALYTICAL (*tda.client.Client.Options.Strategy* attribute), 14
APRIL (*tda.client.Client.Options.ExpirationMonth* attribute), 14
AUGUST (*tda.client.Client.Options.ExpirationMonth* attribute), 14
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

BOND (*tda.client.Client.Markets* attribute), 19

build() (*tda.orders.EquityOrderBuilder* method), 22
BUTTERFLY (*tda.client.Client.Options.Strategy* attribute), 15
BUY (*tda.orders.EquityOrderBuilder.Instruction* attribute), 22
BUY_ONLY (*tda.client.Client.Transactions.TransactionType* attribute), 18

C

CALENDAR (*tda.client.Client.Options.Strategy* attribute), 15
CALL (*tda.client.Client.Options.ContractType* attribute), 14
cancel_order() (*tda.client.Client* method), 12
CANCELLED (*tda.client.Client.Order.Status* attribute), 11
CASH_IN_OR_CASH_OUT (*tda.client.Client.Transactions.TransactionType* attribute), 18
CHECKING (*tda.client.Client.Transactions.TransactionType* attribute), 18
client_from_login_flow() (in module *tda.auth*), 6
client_from_token_file() (in module *tda.auth*), 6
COLLAR (*tda.client.Client.Options.Strategy* attribute), 15
CONDOR (*tda.client.Client.Options.Strategy* attribute), 15
COVERED (*tda.client.Client.Options.Strategy* attribute), 15
create_saved_order() (*tda.client.Client* method), 18
create_watchlist() (*tda.client.Client* method), 20

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.Duration* attribute), 21
 DECEMBER (*tda.client.Client.Options.ExpirationMonth* attribute), 14
 delete_saved_order() (*tda.client.Client* method), 18
 delete_watchlist() (*tda.client.Client* method), 20
 DESC_REGEX (*tda.client.Client.Instrument.Projection* attribute), 13
 DESC_SEARCH (*tda.client.Client.Instrument.Projection* attribute), 13
 DIAGONAL (*tda.client.Client.Options.Strategy* attribute), 15
 DIVIDEND (*tda.client.Client.Transactions.TransactionType* attribute), 18
 DOWN (*tda.client.Client.Movers.Direction* attribute), 19
 Duration (class in *tda.orders*), 21

E

easy_client() (in module *tda.auth*), 6
 EQUITY (*tda.client.Client.Markets* attribute), 19
 EquityOrderBuilder (class in *tda.orders*), 22
 EquityOrderBuilder.Instruction (class in *tda.orders*), 22
 EquityOrderBuilder.OrderType (class in *tda.orders*), 22
 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
 EXPIRED (*tda.client.Client.Order.Status* attribute), 11
 extract_order_id() (*tda.utils.Utils* method), 25

F

FEBRUARY (*tda.client.Client.Options.ExpirationMonth* attribute), 14
 FIFTEEN_YEARS (*tda.client.Client.PriceHistory.Period* attribute), 16
 FILL_OR_KILL (*tda.orders.Duration* attribute), 21
 FILLED (*tda.client.Client.Order.Status* attribute), 11
 find_most_recent_order() (*tda.utils.Utils* method), 26
 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
 FUNDAMENTAL (*tda.client.Client.Instrument.Projection* attribute), 13
 FUTURE (*tda.client.Client.Markets* attribute), 19

G

get_account() (*tda.client.Client* method), 12
 get_accounts() (*tda.client.Client* method), 12
 get_hours_for_multiple_markets() (*tda.client.Client* method), 18
 get_hours_for_single_market() (*tda.client.Client* method), 18
 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), 19
 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), 19
 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.Duration* attribute), 21

I

IN_THE_MONEY (*tda.client.Client.Options.StrikeRange* attribute), 15
 Instrument (class in *tda.client.Client*), 13
 Instrument.Projection (class in *tda.client.Client*), 13
 INTEREST (*tda.client.Client.Transactions.TransactionType* attribute), 18
 InvalidOrderException, 22

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

L

LIMIT (*tda.orders.EquityOrderBuilder.OrderType attribute*), 22

M

MARCH (*tda.client.Client.Options.ExpirationMonth attribute*), 14

MARKET (*tda.orders.EquityOrderBuilder.OrderType attribute*), 22

Markets (*class in tda.client.Client*), 19

matches() (*tda.orders.EquityOrderBuilder method*), 22

MAY (*tda.client.Client.Options.ExpirationMonth attribute*), 14

MINUTE (*tda.client.Client.PriceHistory.FrequencyType attribute*), 16

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

N

NEAR_THE_MONEY (*tda.client.Client.Options.StrikeRange attribute*), 15

NON_STANDARD (*tda.client.Client.Options.Type attribute*), 15

NORMAL (*tda.orders.Session attribute*), 22

NOVEMBER (*tda.client.Client.Options.ExpirationMonth attribute*), 14

O

OCTOBER (*tda.client.Client.Options.ExpirationMonth attribute*), 14

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*), 16

OPTION (*tda.client.Client.Markets attribute*), 19

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*), 14

Options.StrikeRange (*class in tda.client.Client*), 15

Options.Type (*class in tda.client.Client*), 15

Order (*class in tda.client.Client*), 11

Order.Status (*class in tda.client.Client*), 11

ORDERS (*tda.client.Client.Account.Fields attribute*), 12

OTHER (*tda.client.Client.Transactions.TransactionType attribute*), 18

OUT_OF_THE_MONEY (*tda.client.Client.Options.StrikeRange attribute*), 15

P

PENDING_ACTIVATION (*tda.client.Client.Order.Status attribute*), 11

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

place_order() (*tda.client.Client method*), 10

PM (*tda.orders.Session attribute*), 22

POSITIONS (*tda.client.Client.Account.Fields attribute*), 12

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

PUT (*tda.client.Client.Options.ContractType attribute*), 14

Q

QUEUED (*tda.client.Client.Order.Status attribute*), 12

R

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
`ROLL` (*tda.client.Client.Options.Strategy* attribute), 15

S

`SEAMESS` (*tda.orders.Session* attribute), 22
`search_instruments()` (*tda.client.Client* method), 13
`SELL` (*tda.orders.EquityOrderBuilder.Instruction* attribute), 22
`SELL_ONLY` (*tda.client.Client.Transactions.TransactionType* attribute), 18
`SEPTEMBER` (*tda.client.Client.Options.ExpirationMonth* attribute), 14
`Session` (class in *tda.orders*), 21
`set_account_id()` (*tda.utils.Utils* method), 25
`set_duration()` (*tda.orders.EquityOrderBuilder* method), 22
`set_instruction()` (*tda.orders.EquityOrderBuilder* method), 22
`set_order_type()` (*tda.orders.EquityOrderBuilder* method), 22
`set_price()` (*tda.orders.EquityOrderBuilder* method), 23
`set_session()` (*tda.orders.EquityOrderBuilder* method), 23
`SINGLE` (*tda.client.Client.Options.Strategy* attribute), 15
`SIX_MONTHS` (*tda.client.Client.PriceHistory.Period* attribute), 16
`STANDARD` (*tda.client.Client.Options.Type* attribute), 15
`STRADDLE` (*tda.client.Client.Options.Strategy* attribute), 15
`STRANGLE` (*tda.client.Client.Options.Strategy* attribute), 15
`STREAMER_CONNECTION_INFO` (*tda.client.Client.UserPrincipals.Fields* attribute), 20
`STREAMER_SUBSCRIPTION_KEYS` (*tda.client.Client.UserPrincipals.Fields* attribute), 20
`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
`SURROGATE_IDS` (*tda.client.Client.UserPrincipals.Fields* attribute), 20

`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
`TEN_DAYS` (*tda.client.Client.PriceHistory.Period* attribute), 16
`TEN_YEARS` (*tda.client.Client.PriceHistory.Period* attribute), 16
`THREE_DAYS` (*tda.client.Client.PriceHistory.Period* attribute), 16
`THREE_MONTHS` (*tda.client.Client.PriceHistory.Period* attribute), 17
`THREE_YEARS` (*tda.client.Client.PriceHistory.Period* attribute), 17
`TRADE` (*tda.client.Client.Transactions.TransactionType* attribute), 18
`Transactions` (class in *tda.client.Client*), 18
`Transactions.TransactionType` (class in *tda.client.Client*), 18
`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

`UP` (*tda.client.Client.Movers.Direction* attribute), 19
`update_preferences()` (*tda.client.Client* method), 19
`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*), 25

V

`VALUE` (*tda.client.Client.Movers.Change* attribute), 19
`VERTICAL` (*tda.client.Client.Options.Strategy* attribute), 15

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](#)