PyTradier Documentation

Release 0.0

Robert Leonard

Oct 14, 2021

Getting Started

getting_started

Please see the examples for a quick start with this library: Examples!.

CHAPTER 1

Risk Disclosure

Investing and trading through any medium contains a certain level of risk of loss. PyTradier is merely a tool for interfacing with the Tradier Brokerage. By using PyTradier, you agree to the terms and conditions of both Tradier's Brokerage and Development portal and the license associated with PyTradier. The creators and collaborators of PyTradier are not responsible for any fees, damages, losses, or other results associated with using or inability to use this software. Use PyTradier at your own risk.

It is your responsibility to carefully review the laws and regulations associated with this software and the activities associated with this software before attempting to use it.

See the applicable license for PyTradier on GitHub for further information.

CHAPTER 2

Installing PyTradier

As of right now, the only way to install PyTradier is by cloning the GitHub repository into your project.

In the future, PyTradier will be hosted by pypi.com for use with `pip install`. Until then, please clone the repository.

CHAPTER 3

Authenticating Tradier

Coming soon!

CHAPTER 4

Quick Start

PyTradier first needs to have the `Tradier` class initialized, which is how the program logs into the API. This is required since all of Tradier's API is private and requires authentication before accessing. This can be done using the following example:

```
import PyTradier

tradier = Tradier(token='abc123', account_id='123456', sandbox=False)
```

The initialization for `Tradier` takes three parameters:

- `token` Your API access token provided by Tradier. *Required.*
- `account_id` The ID number associated with your Tradier Brokerage account. You will only have this if you have opened a Brokerage account with Tradier. *Optional.*
- `endpoint` Determines whether the developer sandbox, brokerage sandbox, or full API endpoint will be used. Developer sandbox has limited access, but is completely free. The full API requires a Brokerage account. Brokerage accounts also come with a sandbox account for paper trading and has the capabilites of the full API. *Optional.*

Once the `Tradier` class has been initialized, all submodules can be called like this:

```
stocks = tradier.stock('AAPL', 'MSFT')
options = tradier.option('AAPL170630P00130000')
```

CHAPTER 5

Account

5.1 Account Balance

class pytradier.account.**Balance**

account_number(**config)
Return the account number associated with the current account.

account_type(**config)
Return the type of trading account. For example, `'cash'`, `'margin'`, `'pdt'`.

cash_available(**config)
Return the amount of funds ready for trading.

close_pl(**config)
Return the Profit and Loss of the current trading day's closed positions.

current_requirement(**config)
Return the option requirement of current account positions.

day_trade_buying_power(**config)
Return the total amount of funds available for the purchase of fully marginable stock during the current trading day. A portion of these funds cannot be held overnight.

dividend_balance(**config)
Return the account's dividend balance.

equity(**config)
Return the account's equity value.

fed_call(**config)
Returns the amount that the account is in deficit for trades that have occurred but not been paid for.

long_liquid_value(**config)
Return the account's long liquid value.

long_market_value(**config)
Return the account's long market value.

maintenance_call(**config)
Return the amount that the account is under the minimum equity required in the account to support the current holdings.

market_value(**config)
Return the market value of the account's positions.

net_value(**config)
Return the net value of the account's positions.

open_pl(**config)
Return the Profit & Loss of the account's current positions.

option_buying_power(**config)
Return the amount of funds available to purchase non-marginable securities.

option_long_value(**config)
Return the value of long options held in the account.

option_requirement(**config)
Return the account's option requirement.

option_short_value(**config)
Return the value of short options held in the account.

pending_cash(**config)
Return the amount of cash that is being held for open orders. This is generally from funds that are transferred into the account or from selling profitable positions.

pending_orders_count(**config)
Returns the account's amount of open orders.

short_liquid_value(**config)
Return the short liquid value of the account.

short_market_value(**config)
Return the short market value of the account.

stock_buying_power(**config)
Return the amount of funds available to purchase fully marginable securities.

stock_long_value(**config)
Return the value of long stocks held in the account.

stock_short_value(**config)
Return the value of short stocks held in the account.

sweep(**config)
Sweep. Documentation in progress.

total_cash(**config)
Return the total amount of cash in the account.

total_equity(**config)
Return the total value of the account.

uncleared_funds(**config)
Return the amount of funds that are not currently available for trading.

unsettled_funds(**config)
Return the amount of cash that is in the account from recent stock or option sales, but has not yet settled.

Coming soon!

CHAPTER 6

User

Coming soon!

CHAPTER 7

Market Information

The market directory allows programs to access information about the market, including searching for companies and retrieving market timings.

7.1 Market Calendar

class pytradier.market.**Calendar**(*month=None*, *year=None*)
A class for obtaining dates and times for the market, handling both past and future dates.

Note: The output of each method is a dictionary with a date as the key and the particular value of the method as the value.

__init__(*month=None*, *year=None*)
Create an instance of the Calendar class.

Parameters

- **month** – Month of the calendar requested.
- **year** – Year of the calendar requested.

Either one or both of `month` and `year` can be provided. The fetched calendar will only be as specific as you request. For example:

```
calendar1 = tradier.market.calendar(month=12, year=2016)
calendar2 = tradier.market.calendar(year=2017)
```

`calendar1` will return a calendar of only December, 2016. `calendar2`, on the other hand, will return a calendar for all of 2017. Future dates are also accepted.

date(***config*)
Return the date in the format YYYY-MM-DD.

desc(***config*)
Returns a short description of the date.

month(***config*)
Retrieve the month of the calendar.

open(***config*)
Returns a dictionary of the interval the market is open. Returns `NoneType` if the market is closed (i.e. Sunday, etc.).

postmarket(***config*)
Returns a dictionary of the interval time for postmarket. Returns `NoneType` if there isn't a postmarket (i.e. Sunday, etc.).

premarket(***config*)
Returns a dictionary of the interval time for the premarket. Returns `NoneType` if there isn't a premarket (i.e. Sunday, etc.).

status(***config*)
Get the status of the market.

year(***config*)
Retrieve the year of the calendar.

7.1.1 Examples

```
# Retrieve a dictionary of the market open intervals
calendar = tradier.market.calendar()
print calendar.open()
```

The output is a dictionary with dates as keys and the status of the market on that particular date. Below is a shortened example output:

```
{u'2017-07-05': {u'start': u'09:30', u'end': u'16:00'}, u'2017-07-04': None, u'2017-
↪07-03': {u'start': u'09:30', u'end': u'13:00'}}
```

The `open()` function (as well as `premarket()` and `postmarket()`) returns a dictionary for each date in the main dictionary. This is useful since the information can be searched by its date (the key), or the entire dictionary can be looped through to get the values of each key. For example, to programmatically retrieve the market open interval from each day, use a `for` loop:

```
for key in calendar.open():
   print calendar.open()[key]  # loop through each nested dictionary
```

The (shortened) output of this is as follows:

```
None
{u'start': u'09:30', u'end': u'16:00'}
{u'start': u'09:30', u'end': u'16:00'}
```

7.2 Symbol Lookup

class pytradier.market.**Lookup**(***query*)
A class for looking up symbols or partial symbols. The results are listed by highest volume. The results are a dictionary with the symbol as the key and the information from each method as the value.

__init__(***query*)
Create an instance for the symbol lookup.

Parameters

- **symbol** – The requested search symbol. It can be a full or partial symbol.
- **type** – The type of symbol requested: `stock`, `etf`
- **exchange** – The exchange of the symbol. As of current, only Tradier exchange symbols are accepted.

Any combination of these three parameters can be used to create a search. The results of the search are stored in the instance of the class, which allows for the local storage of results rather than having to call to the API for each piece of information.

This gives you a wide range of abilities. For example, to retrieve *every* stock on the NYSE, use the argument `exchange='A'` ('A' is the Tradier exchange code for NYSE), and ignore the other parameters:

```
nyse = tradier.market.lookup(exchange='A')
print nyse.symbol
```

And the result is a long dictionary, sorted by largest volume:

```
{BAC: BAC, TWTR: TWTR, RAD: RAD, ... }
```

desc(***config*)
Return a short description of the symbol. | For example:

```
tradier.market.lookup(symbol='AAPL').desc()
```

exchange(***config*)
Return the exchange of the symbol. | For example:

```
tradier.market.lookup(symbol='AAPL').exchange()
```

Note: Exchanges are returned as symbols according to Tradier's naming system of exchanges.

symbol(***config*)
Return the symbol from the search. | For example:

```
tradier.market.lookup(symbol='AAPL').symbol()
```

type(***config*)
Return the type of symbol (`stock`, `etf`, `index`). | For example:

```
tradier.market.lookup(symbol='AAPL').type()
```

7.3 Company Search

class pytradier.market.**Search**(***query*)
A class for searching for a company.

desc(***config*)
Return a short description of the company.

exchange(**config)
Return the exchange where the symbol is located.

> **Note:** This method currently outputs Tradier's exchange codes.

symbol(**config)
Return the symbol of the company.

type(**config)
Return the type of symbol (`etf`, `stock`, `index`).

7.4 Market Status

class pytradier.market.**Status**
A class for the current market status.

date(**config)
An ISO representation of the date in YYYY-MM-DD.

desc(**config)
A short description of the market status.

next_change(**config)
Returns the time of next state change.

next_state(**config)
Returns the next state of the market (i.e. premarket, postmarket, etc.)

state(**config)
Returns the current state of the market.

timestamp(**config)
Returns the timestamp of the status update.

The default style is Unix Epoch time, though using `style='pretty'` returns the time in `YYYY-MM-DD H:M:S`

CHAPTER 8

Company Information

8.1 Options Chain

class pytradier.company.**Chain**(*symbol*, *expiration*, *greeks=<type 'bool'>*)

__init__(*symbol*, *expiration*, *greeks=<type 'bool'>*)

Create an instance of the `Chain` class.

Parameters

- **symbol** – The symbol for the options chain. By default, this is passed from the parent `Company` class.
- **expiration** – The desired expiration for the options in the format `YYY-MM-DD`. *Required.*
- **greeks** – API responce will include a dictionary object containing the greeks for each contract. "Bool"

  greeks.delta Delta greeks.gamma Gamma greeks.theta Theta greeks.vega Vega greeks.rho Rho greeks.phi Phi greeks.bid_iv Bid implied volatility greeks.mid_iv Mid implied volatility greeks.ask_iv Ask implied volatility greeks.smv_vol ORATS final implied volatility greeks.updated_at Date volatility data was last updated

---

Note: All options contracts expire on a Friday. If the requested expiration date is not a Friday, no data will be returned!

Like many of the other functions PyTradier offers, `Chain` returns the data in the form of a dictionary with the options symbol as the key and the requested data as the value.

ask(***config*)

Return the latest ask price for each contract.

bid(***config*)
Return the latest bid price for each contract.

change(***config*)
Return the latest change in price for each contract.

greeks(***config*)
Returns a dictionay of the greek attributes of each contract.

interest(***config*)
Return the open interest for each contract.

last(***config*)
Return the latest price for each contract.

strike(***config*)
Return the strike price for each contract.

symbol(***config*)
Return the symbol of the contract. For example:

```
chain = tradier.company('TSLA').chain()
print chain.symbol()
```

This will output a large options chain that looks similar to this:

```
{TSLA170714C00292500: TSLA170714C00292500, TSLA170714C00295000:␣
↪TSLA170714C00295000, ...}
```

You can then get updates on particular contracts through the `Option` class using the output symbols from this method as the input for the `Option` class.

8.2 Historical Pricing

class pytradier.company.**History**(*symbol*, *interval=None*, *start=None*, *end=None*)
A class for handling historical pricing of companies.

__init__(*symbol*, *interval=None*, *start=None*, *end=None*)
Create an instance of the History class.

Parameters

- **symbol** – The requested company symbol. By default, the parent `Company` class symbol is passed to this method.
- **interval** – Interval of time for the requested data set. Defaults to `'daily'`, but can also take `weekly` and `monthly`. For example, if the interval is set to `'monthly'`, then each piece of data represents one month. A smaller interval has more data. *Optional.*
- **start** – The start date of the data set in the format `YYYY-MM-DD`. *Optional.*
- **end** – The end date of the data set in the format `YYYY-MM-DD`. *Optional.*

Note: Each method (except `bundle`) returns a dictionary with the date as the key and the requested data as the value. Dictionaries are *unordered*.

---

bundle(*reverse_sort=False*)
Returns a multi-dimension list of all data, sorted by ascending by default. Each element is in the form, `[epoch, open, close, high, low, volume]`.

Parameters **reverse_sort** – Determines the order of sorting the data. Default is False, which sorts by ascending dates.

This method can be useful for machine learning or other data handling methods since this returns a list of numbers, rather than a dictionary with keys.

Example:

```
company = tradier.company(symbol='AAPL')
history = company.history(interval='monthly', start='2011-1-1', end='2012-1-1
↪')
print(history.bundle(reverse_sort=True))
```

The above code will output a multi-dimension list with descending dates:

```
[[1325394000, 58.485714, 64.715714, 64.921429, 58.428571, 1617345800],
[1322715600, 54.648571, 57.857143, 58.441429, 53.954286, 1577342800], ...]
```

close(***config*)
Return the closing price for the interval.

date(***config*)
Return the date in ISO format `YYYY-MM-DD`.

high(***config*)
Return the highest price from the interval.

low(***config*)
Return the lowest point from the interval.

open(***config*)
Return the opening price for the interval. For example, is interval is `daily`, the open price will be the price at market open, and the closing price is the price at market close.

volume(***config*)
Return the total volume for the interval.

8.2.1 Examples

A year's worth of historical prices can be retrieved using this example:

```
company = tradier.company(symbol='AAPL')
history = company.history(interval='daily', start='2011-1-1', end='2012-1-1')
print history.high()
```

8.3 Time and Sales

class pytradier.company.**TimeSales**(*symbol*, *interval=None*, *start=None*, *end=None*, *sfilter=None*)

__init__(*symbol*, *interval=None*, *start=None*, *end=None*, *sfilter=None*)
Create an instance of the TimeSales class.

Parameters

- **symbol** – The company symbol. By default, this is passed in from the parent `Company` class.
- **interval** – The time interval between each sale. Options include `'tick'`, `'1min'`, `'5min'`, and `'15min'`. *Optional.*
- **start** – Start datetime for timesales range represented as `YYYY-MM-DD HH:MM`. *Optional*.
- **end** – End date for timesales range represented as `YYYY-MM-DD HH:MM`. *Optional*.
- **sfilter** – The session conditions to requetst data for. Defaults to `'all'` for all available data points. Use `'open'` for data points within open market hours only. *Optional.*

close(***config*)
Return the price of the end of the data interval.

high(***config*)
Return the highest price from the interval.

low(***config*)
Return the lowest price of the interval.

open(***config*)
Return the price of the start of the data interval. For example, if `interval` was set to `'15min'`, the opening price for that interval would the from the start of the 15 minute interval, and the closing price would be from the end of the interval.

price(***config*)
Return the last price of the interval.

time(***config*)
Return the time of the data.

volume(***config*)
Return the total volume of the interval.

vwap(***config*)
Return the volume weighted average price of the interval.

CHAPTER 9

Orders

Coming soon!

CHAPTER 10

Securities

The securities directory is an easy way to access market data for stocks and options. Both the `stock` and `option` classes inherit from the `quote` class.

10.1 Options

class pytradier.securities.**Option**(**symbols*)

A class for fetching and storing market data for options.

__init__(**symbols*)

Create an instance of the Option class.

Parameters **symbols** – The symbol(s) of the options contract(s) to track.

One or more standard symbols can be used to create in instance:

```
options = tradier.Option('TSLA170707C00295000', 'F170714C00010500')
print options.underlying()
```

The result is a dictionary:

```
{TSLA: TSLA, F:F}
```

add_symbols(**symbols*, ***config*)

Add one or more symbols to the array of tracked symbols.

Parameters **symbols** – A string containing the company's symbol.

Multiple symbols can be added at once:

```
stocks = tradier.Stock('AAPL', 'MSFT')  # create instance of the Stock class
print stocks.symbol()  # output: {AAPL: AAPL, MSFT: MSFT}

stocks.add_symbols('GOOG', 'TSLA')
print stocks.symbol()  # output: {AAPL: AAPL, MSFT: MSFT, GOOG: GOOG, TSLA:␣
↪TSLA}
```

(continues on next page)

(continued from previous page)

ask(***config*)
Return the latest ask price.

ask_date(***config*)
Return the date and time of the latest ask in Unix Epoch time.

askexch(*full_name=False*, ***config*)
Return the exchange of the current ask. This will return a dictionary of each symbol and the respective exchange code. The `full_name` parameter can be specified to return the name of the exchange rather than the exchange code. For example:

```
stock = tradier.stock('AAPL', 'TSLA')
print(stock.bidexch())
print(stock.bidexch(full_name=True))
```

This will return the following dictionaries:

```
{'AAPL': 'Q', 'TSLA': 'Z'}
{'AAPL': 'NASDAQ OMX', 'TSLA': 'BATS'}
```

asksize(***config*)
Return the size of the current ask.

average_volume(***config*)
Return the average volume.

bid(***config*)
Return the latest bid price.

bid_date(***config*)
Return the date and time of the latest bid in Unix Epoch time.

bidexch(*full_name=False*, ***config*)
Return the exchange of the current bid. This will return a dictionary of each symbol and the respective exchange code. The `full_name` parameter can be specified to return the name of the exchange rather than the exchange code. For example:

```
stock = tradier.stock('AAPL', 'TSLA')
print(stock.bidexch())
print(stock.bidexch(full_name=True))
```

This will return the following dictionaries:

```
{'AAPL': 'Q', 'TSLA': 'Z'}
{'AAPL': 'NASDAQ OMX', 'TSLA': 'BATS'}
```

bidsize(***config*)
Return the size of the current bid.

change(***config*)
Return the dollar change.

change_percentage(***config*)
Return the percentage change

close(***config*)
Return the closing price.

contract_size(**config)
Return the size of the contract.

desc(**config)
Return a short description for each symbol.

exchange(**config)
Return the exchange on which the symbol is traded.

Note: This returns a symbol according to Tradier's exchange codes

---

expiration(**config)
Return the expiration.

expiration_type(**config)
Return the type of expiration (`standard`, `weeklys`).

high(**config)
Return the trading day high.

last_volume(**config)
Return the latest volume.

low(**config)
Return the trading day low.

open(**config)
Return the opening price.

open_interest(**config)
Return the open interest for the contract.

option_type(**config)
Return the option type (`put` or `call`).

prevclose(**config)
Return the previous closing price.

strike(**config)
Return the strike price of the option.

symbol(**config)
Return the symbol of each key in the dictionary.

trade_date(**config)
Return the date and time of last trade in Unix Epoch time.

underlying(**config)
Return the underlying symbol for the contract.

volume(**config)
Return the volume for the day.

week_52_high(**config)
Return the 52 week high.

week_52_low(**config)
Return the 52 week low.

10.2 Stock

class pytradier.securities.**Stock**(**symbols*)

A class for fetching and storing market data for stocks.

add_symbols(**symbols*, ***config*)

Add one or more symbols to the array of tracked symbols.

Parameters **symbols** – A string containing the company's symbol.

Multiple symbols can be added at once:

```
stocks = tradier.Stock('AAPL', 'MSFT')  # create instance of the Stock class
print stocks.symbol()  # output: {AAPL: AAPL, MSFT: MSFT}

stocks.add_symbols('GOOG', 'TSLA')
print stocks.symbol()  # output: {AAPL: AAPL, MSFT: MSFT, GOOG: GOOG, TSLA:␣
↪TSLA}
```

ask(***config*)

Return the latest ask price.

ask_date(***config*)

Return the date and time of the latest ask in Unix Epoch time.

askexch(*full_name=False*, ***config*)

Return the exchange of the current ask. This will return a dictionary of each symbol and the respective exchange code. The `full_name` parameter can be specified to return the name of the exchange rather than the exchange code. For example:

```
stock = tradier.stock('AAPL', 'TSLA')
print(stock.bidexch())
print(stock.bidexch(full_name=True))
```

This will return the following dictionaries:

```
{'AAPL': 'Q', 'TSLA': 'Z'}
{'AAPL': 'NASDAQ OMX', 'TSLA': 'BATS'}
```

asksize(***config*)

Return the size of the current ask.

average_volume(***config*)

Return the average volume.

bid(***config*)

Return the latest bid price.

bid_date(***config*)

Return the date and time of the latest bid in Unix Epoch time.

bidexch(*full_name=False*, ***config*)

Return the exchange of the current bid. This will return a dictionary of each symbol and the respective exchange code. The `full_name` parameter can be specified to return the name of the exchange rather than the exchange code. For example:

```
stock = tradier.stock('AAPL', 'TSLA')
print(stock.bidexch())
print(stock.bidexch(full_name=True))
```

This will return the following dictionaries:

```
{'AAPL': 'Q', 'TSLA': 'Z'}
{'AAPL': 'NASDAQ OMX', 'TSLA': 'BATS'}
```

bidsize(***config*)
Return the size of the current bid.

change(***config*)
Return the dollar change.

change_percentage(***config*)
Return the percentage change

close(***config*)
Return the closing price.

desc(***config*)
Return a short description for each symbol.

exchange(***config*)
Return the exchange on which the symbol is traded.

Note: This returns a symbol according to Tradier's exchange codes

---

high(***config*)
Return the trading day high.

last_volume(***config*)
Return the latest volume.

low(***config*)
Return the trading day low.

open(***config*)
Return the opening price.

prevclose(***config*)
Return the previous closing price.

symbol(***config*)
Return the symbol of each key in the dictionary.

trade_date(***config*)
Return the date and time of last trade in Unix Epoch time.

volume(***config*)
Return the volume for the day.

week_52_high(***config*)
Return the 52 week high.

week_52_low(***config*)
Return the 52 week low.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Index

Symbols

A

B

C

D

T

U

V

W

Y