
IEXTools Documentation

Release 0.0.4

Victor Frazao

Jun 21, 2021

Contents:

1	API Client	3
1.1	Usage	3
1.2	Module Docs	4
1.3	Indices and tables	4
2	Data Downloader	5
2.1	Usage	5
2.2	Module Docs	6
2.3	Indices and tables	6
3	Parser	7
3.1	Usage	7
3.2	Module Docs	8
3.3	Indices and tables	8
4	Installation	9
4.1	Requirements	9
5	Indices and tables	11

This package provides tools for working with data provided by IEX's REST API and tools to decode and use IEX's binary market data (dubbed "HIST"). For more information on the type of data offered by IEX please visit their website: <https://iextrading.com/developer/docs> and <https://iextrading.com/trading/market-data/>

1.1 Usage

Purpose: Interact with the IEX web API. All methods return Python dictionaries.

The web API has a large number of endpoints returning data:

```
>>> from IEXTools import IEXAPI
>>> api = IEXAPI()
>>> meths = [d for d in dir(api) if not d.startswith('_')]
>>> for i, j, k in zip(meths[::3], meths[1::3], meths[2::3]):
...     print(i.ljust(20), j.ljust(20), k.ljust(20))
...
```

BASE	batch	book
chart	collections	company
crypto	deep	deep_book
deep_trades	delayed_quote	dividends
earnings	earnings_today	effective_spread
financials	hist	iex_auction
iex_corp_actions	iex_dividends	iex_historical
iex_historical_daily	iex_next_day_ex_div	iex_official_price
iex_short_interest	iex_stats_intraday	iex_stats_recent
iex_stats_records	iex_symbols	iex_threshold_securities
largest_trades	last	logo
market	news	ohlcv
operational_halt	peers	previous
price	quote	relevant
sector_performance	security_event	short_sale_price_test_status
splits	stats	stock_list
symbols	system_event	timeout
timeseries	today_ipos	tops
trade_break	trading_status	upcoming_ipos

Users should consult the docstrings of a given function or IEX's docs for additional information on how to use a given endpoint. All endpoints documented in the IEX API docs are implemented in this class:

```
>>> help(api.ohlcv)
Help on method ohlc in module IEX_API:

ohlcv(symbol: str) -> dict method of IEX_API.IEX_API instance
    Returns the open, high, low, and close prices for a given company.

    https://iextrading.com/developer/docs/#ohlcv

>>> apple_ohlcv = api.ohlcv('aapl')
>>> print(IEX_API.pretty_json(apple_ohlcv))
{
  "close": {
    "price": 204.47,
    "time": 1541797200568
  },
  "high": 206.01,
  "low": 202.25,
  "open": {
    "price": 205.55,
    "time": 1541773800180
  }
}
```

All symbols available on the API can be retrieved using the *symbols* method:

```
>>> all_symbols = api.symbols()
>>> len(all_symbols)
8756
>>> api.symbols()[1]
{'symbol': 'AA', 'name': 'Alcoa Corporation', 'date': '2018-11-09', 'isEnabled': True,
  'type': 'cs', 'iexId': '12042'}
```

1.2 Module Docs

1.3 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

2.1 Usage

Purpose: Download IEX's pcap files containing nanosecond precision stock data - the so called HIST files.

The *DataDownloader* class can be instantiated without any arguments by simply calling the class.

```
d1 = IEXTools.DataDownloader()
```

There are three available methods in this class:

```
>>> print([method for method in dir(IEXTools.DataDownloader) if not method.startswith(
↳ '_' )])

['decompress', 'download', 'download_decompressed']
```

- download: Downloads the gzipped TOPS or DEEP file for a given datetime input
- decompress: Unzips the compressed HIST file into a pcap
- download_decompressed: downloads and decompresses the HIST file - deletes the zipped file at the end

Warning, IEX HIST files are generally very large (multiple gbs)

Usage:

```
>>> import IEXTools
>>> from datetime import datetime
>>> d1 = IEXTools.DataDownloader()
>>> d1.download_decompressed(datetime(2018, 7, 13), feed_type='tops')
'20180713_IEXTP1_TOPS1.6.pcap'
```

2.2 Module Docs

2.3 Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)

3.1 Usage

Purpose: Parse the binary PCAP / HIST files offered by IEX.

To create a Parser object simply supply the file path as an argument.

```
>>> from IEXTools import Parser, messages
>>> p = Parser(r'IEX TOPS Sample\20180103_IEXTP1_TOPS1.6.pcap')
>>> p
Parser("IEX TOPS Sample\\20180103_IEXTP1_TOPS1.6.pcap", tops=True, deep=False)
```

This instantiates a Parser object with the pcap file opened. You can also optionally specify what type of HIST file you are loading, either TOPS (*tops=True*) or DEEP (*deep=True*).

Use the *get_next_message* method of the Parser object to return a message object. The message objects are documented in the *messages.py* module. You can optionally specify a list of message classes to restrict the returned messages to only those types.

```
>>> allowed = [messages.TradeReport]
>>> p.get_next_message(allowed)
TradeReport(flags=64, timestamp=1514984427833117218, symbol='ZVZZT', size=975, price_
↳int=100150, trade_id=577243)
```

The last message retrieved can also be accessed through the Parser object itself with the *message* attribute. Similarly the message type can be accessed as *Parser.message_type* and the binary encoded message can be accessed with *Parser.message_binary*. The message object also has several attributes that may be accessed (varies by object).

```
>>> p.message
TradeReport(flags=64, timestamp=1514984427833117218, symbol='ZVZZT', size=975, price_
↳int=100150, trade_id=577243)
>>> p.message.date_time
datetime.datetime(2018, 1, 3, 13, 0, 27, 833117, tzinfo=datetime.timezone.utc)
>>> p.message.price
```

(continues on next page)

(continued from previous page)

```
10.015
>>> p.message_type
84
>>> p.message_binary
b'@
" \xca=uON\x06\x15ZVZZT\xcf\x03\x00\x006\x87\x01\x00\x00\x00\x00\x00\x00\xdb\xce\x08\x00\x00\x00\x00\x00
'
```

The program also allows you to use it with a context manager and loop through it like a file:

```
with Parser(file_path) as iex_messages:
    for message in iex_messages:
        do_something(message)
```

Benchmarks On my personal laptop (Lenovo ThinkPad X1 Carbon, Windows 10):

```
Beginning test - 1,000,000 messages - all messages, not printing
Parsed 1,000,000 messages in 52.2 seconds -- 19141.6 messages per second

Beginning test - 1,000,000 messages - only TradeReport and QuoteUpdate messages, not
printing
Parsed 1,000,000 messages in 54.0 seconds -- 18512.9 messages per second
```

By not specifying the *allowed* argument the parser returns 1,000,000 parsed messages approximately 3% faster. However, in order to return 1,000,000 parsed messages the Parser with the *allowed* argument set may have to read through more than 1,000,000 messages. Testing suggests that actually decoding the message takes about 10 microseconds (130,000 messages per second).

3.2 Module Docs

3.3 Indices and tables

- genindex
- modindex
- search

Note: both of these installation methods will fail without Python 3.7 or greater.

Install from PyPI:

```
$ py -m pip install IEXTools
```

OR

Navigate to the folder containing the README.md file and run the pip command to install the package:

```
$ pip install .
```

4.1 Requirements

- Python 3.7 or greater
- requests

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`