
# TWS Python API - Receiving Streaming Data and Historical Candlesticks – Study Notes

One of the most common tasks for an API program is to request real time or historical market data. In this lesson we will discuss the different types of ways to request data from the API, and walkthrough the simplest Python API programs which can request/receive real time and historical data.

For reference, the associated sections in the API reference guide can be found at :

- http://interactivebrokers.github.io/tws-api/market_data.html

There are several API functions which provide streaming market data updates:

- reqMktData: Provides updates several times per second.
- reqRealTimeBars: Provides OHLC bars for each 5 second period
- reqHistoricalData (when 'keepUpToDate' is set to True):
  - Provides bars and updates of unfinished bars
- reqTickByTickData: Provides streaming tick-by-tick data for every trade or quote change

For each function a Contract object must first be defined which uniquely defines a single financial instrument. Many contract definition examples are found in the API reference guide at: http://interactivebrokers.github.io/tws-api/contracts.html

**reqMktData**

The most commonly-used function for streaming market data updates is reqMktData- it has the fewest limitations on making requests for many tickers quickly, and the has most options in types of data returned. Every TWS session can receive at least 100 streaming quotes for tickers in TWS watchlists and the API.

The data returned from reqMktData will match a corresponding field for a ticker which would be shown in a TWS watchlist.

When using the reqMktData function, there are four 'market data modes' available:

1) Live streaming (the default)
2) Frozen (typically used for bid/ask prices after market close)

3) Delayed (if the username does not have live market data subscriptions)
4) Delayed-Frozen (combination of types 2 & 3)

- http://interactivebrokers.github.io/tws-api/market_data_type.html

To receive streaming data in an API client, the program will have to have the essential parts described in the prior talk, namely a class which derives from EClient and EWrapper, overridden EWrapper callbacks, a connect() call, and a run() loop.

---

*Example of requesting streaming market data for AAPL*

```
from ibapi.client import EClient
from ibapi.wrapper import EWrapper
from ibapi.contract import Contract
from ibapi.ticktype import TickTypeEnum


class TestApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def error(self, reqId, errorCode, errorString):
        print("Error: ", reqId, " ", errorCode, " ", errorString)

    def tickPrice(self, reqId, tickType, price, attrib):
        print("Tick Price. Ticker Id:", reqId, "tickType:",
TickTypeEnum.to_str(tickType), "Price:", price, end=' ')

    def tickSize(self, reqId, tickType, size):
        print("Tick Size. Ticker Id:", reqId, "tickType:",
TickTypeEnum.to_str(tickType), "Size:", size)


def main():
    app = TestApp()

    app.connect("127.0.0.1", 7497, 0)

    contract = Contract()
    contract.symbol = "AAPL"
    contract.secType = "STK"
    contract.exchange = "SMART"
    contract.currency = "USD"
    contract.primaryExchange = "NASDAQ"

    app.reqMarketDataType(4)  # switch to delayed-frozen data if live is not available
    app.reqMktData(1, contract, "", False, False, [])

    app.run()


if __name__ == "__main__":
    main()
```

---

Live data, whether from the API or an IBKR trading platform, requires that market data subscriptions are enabled for the instrument in Account Management. Some forex, bond, and CFD data is free and enabled by default but other subscriptions have associated fees and so require a funded, live account to add subscriptions. To receive historical candlestick data for an instrument from the API, it is necessary to have live data permissions first.

The first step is to create a Contract object which uniquely identify a financial instrument. This was shown in the previous talk which demonstrated the reqContractDetails function.

**Contract Class**

The API contract class can be used to define any tradeable IBKR product, as well as instruments such as indices. The same contract object can then be used to request historical candlestick data, or to place an order. There are several ways to define any financial instrument in a Contract object using a different combination of parameters. A combo or spread can also be defined using a single API contract object. There are examples for all types of instruments in the API documentation at: http://interactivebrokers.github.io/tws-api/contracts.html

A great way to lookup the fields for a particular instrument that is already in a TWS watchlist is to right-click on the instrument and choose Financial Instrument Info -> Description. The fields shown in the TWS Financial Instrument Description window are the same ones used in the API.

**Code walkthough: reqMktData function parameters and callbacks**

In this example we will show how to request delayed data for a stock, in case you do not yet have an open account with market data permissions. To switch to the delayed data mode for the reqMktData function, we first have to invoke reqMarketDataType. We can use either a parameter of 3, which would indicate delayed data, or 4 which would indicate delayed-frozen data. Delayed frozen data means that after a market closes, the bid/ask values shown will refer to the last available value before close.

After switching to data type 4, we can use the reqMktData function to request delayed data for the defined AAPL contract object. In addition to the Contract object and ticker ID which is used to uniquely identify a particular reqMktData request, the other parameters are a Generic Tick List, used to request other types of data for the instrument, and snapshot Booleans, used to request snapshot rather than streaming data.

**Historical Data**

Historical candlestick data, corresponding to the data shown in TWS charts, can be retrieved from the API using the reqHistoricalData function. This function does require live level 1 (streaming data) for the instrument.

API Reference Guide: http://interactivebrokers.github.io/tws-api/historical_data.html

This data corresponds to the data in TWS charts and will match exactly if a chart is created with all the same parameters. The amount of historical data available from IBKR will depend on the instrument type- for common stocks:

- For common stocks typically there is at least 5-10 years of data available at large bar sizes
- Data is available for 2 years of expired futures
- Data is not available for expired options or futures options

http://interactivebrokers.github.io/tws-api/historical_limitations.html

Some other important historical data limitations are:

- Requests for bar sizes of 30 seconds and below have a limitation that no more than 60 requests can be made in 10 minutes
- Requests for large amounts of data at longer bar sizes will be paced so that data will take longer to be returned

**Code walkthough: reqHistoricalData**

To request historical data with the reqHistoricalData function, the other parameters in the request are the tickerId (which uniquely defines the request), the candlestick bar size, duration, trade type, and the keepUpToDate Boolean which indicates if data should continue streaming in real time after historical data has been returned.

---

*Forex Historical Data Example*

```
from ibapi.client import EClient
from ibapi.wrapper import EWrapper
from ibapi.contract import Contract

class TestApp(EWrapper, EClient):
    def __init__(self):
        EClient.__init__(self, self)

    def error(self, reqId, errorCode, errorString):
        print("Error: ", reqId, " ", errorCode, " ", errorString)

    def historicalData(self, reqId, bar):
        print("HistoricalData. ", reqId, " Date:", bar.date, "Open:", bar.open,
"High:", bar.high, "Low:", bar.low, "Close:", bar.close, "Volume:", bar.volume,
"Count:", bar.barCount, "WAP:", bar.average)

def main():
    app = TestApp()

    app.connect("127.0.0.1", 7497, 0)

    # define contract for EUR.USD forex pair
    contract = Contract()
    contract.symbol = "EUR"
    contract.secType = "CASH"
    contract.exchange = "IDEALPRO"
    contract.currency = "USD"

    app.reqHistoricalData(1, contract, "", "1 D", "1 min", "MIDPOINT", 0, 1, False,
[])

    app.run()
```

```
if __name__ == "__main__":
    main()
```

---

A common design is to store the returned data in Pandas dataframes. https://pandas.pydata.org/

It is important to note how candlesticks from IB's historical datafeed are constructed for trade data. Some types of trades, typically occurring away from the NBBO, are not included in the historical data: average price trades, odd lots, and derivative trades. Generally only includes trades on the NBBO are included. For that reason the volume will be different than other data sources which include these trade types.