VOLUME I

PYTHON, IbPy AND INTERACTIVE BROKERS

NOTES

Version 1.0
Draft

Anthony Ng

Last Update: 1 Feb 2016
Based on API Reference Guide. July 2015.
Updated through API Release 9.72[1]

[1] `https://www.interactivebrokers.com/en/software/api/api.htm`

Contents

I IbPy **3**

1 Introduction **4**
- 1.1 IbPyLikeJava . . . 4
- 1.2 IbPyOptional . . . 5
- 1.3 IbPy Online Documentation . . . 6

2 Connection and Server **7**

3 Market Data **8**
- 3.1 Introduction . . . 8

4 Orders **18**
- 4.1 Introduction . . . 18
  - 4.1.1 Asset Classes . . . 19
- 4.2 Placing Orders and Examples . . . 24
  - 4.2.1 Placing a Stock Order . . . 24
  - 4.2.2 Placing a FX Order . . . 28
  - 4.2.3 Placing a Futures Order . . . 32
  - 4.2.4 Placing an Option Order . . . 36
  - 4.2.5 Placing a Bond Order . . . 38

5 Account and Portfolio **40**
- 5.1 Introduction . . . 40
- 5.2 Obtaining Account Information . . . 42
- 5.3 Obtaining Portfolio Information . . . 45
- 5.4 Obtaining Account Update Time . . . 47
- 5.5 Obtaining Account Summary . . . 49
- 5.6 Obtaining Position Information . . . 51

II Sample Codes **53**

6 Account and Portfolio **54**

III Quick Reference **72**

7 Summary **73**

Part I

IbPy

Chapter 1

Introduction

IbPy was originally created by Troy Melhase[1]. And the codes are hosted on GitHub. You can find them here @ blampe/IbPy and here @ jplehmann/IbPy. The latter is a forked from blampe/IbPy.

IbPy is a third-party implementation of the API used for accessing the Interactive Brokers on-line trading system. IbPy implements functionality that the Python programmer can use to connect to IB, request stock ticker data, submit orders for stocks and futures, and more.

https://code.google.com/p/ibpy/
https://code.google.com/p/ibpy/wiki/DocumentationIndex

1.1 IbPyLikeJava

Synopsis

```
from ib.ext.EWrapper import EWrapper
from ib.ext.EClientSocket import EClientSocket

class SomeWrapper(EWrapper):
    def tickPrice(self, tickerId, field, price, canAutoExecute):
        ... code to handle tickPrice data ...

    ... code to implement all other EWrapper methods ...

wrapper = SomeWrapper()
connection = EClientSocket(wrapper)
connection.eConnect('localhost', 7496, 0)
connection.reqIds()
... additional requests ...
connection.eDisconnect()
```

Details

IbPy is built by automatic translation of the reference Java implementation supplied by Interactive Brokers. The generated modules live in the ib.ext package, and provide the same interface as their Java counterparts.

You can use IbPy just like you would use the IB Java package. To do so, you must define a subclass of EWrapper,

[1] https://roundrockriver.wordpress.com/2007/02/15/automated-translation-of-java-to-python/#more-81

implement all of its methods, then provide an instance of your subclass to an EClientSocket instance. After you have connected the socket instance, you can call the TWS API to request account information, market information, place orders, etc.

Refer to the IbPy API documentation and to the Interactive Brokers documentation for more information.

1.2 IbPyOptional

Synopsis

```
from ib.opt import ibConnection, message

def my_account_handler(msg):
    ... do something with account msg ...

def my_tick_handler(msg):
    ... do something with market data msg ...

connection = ibConnection()
connection.register(my_account_handler, 'UpdateAccountValue')
connection.register(my_tick_handler, 'TickSize', 'TickPrice')
connection.connect()
connection.reqAccountUpdates(...)
```

Details

IbPy provides an optional interface that does not require subclassing. This interface lives in the ib.opt package, and provides several conveniences for your use.

To interoperate with this package, first define your handlers. Each handler must take a single parameter, a Message instance. Instances of Message have attributes and values set by the connection object before they're passed to your handler.

After your handlers are defined, you associate them with the connection object via the register method. You pass your handler as the first parameter, and you indicate what message types to send it with parameters that follow it. Message types can be strings, or better, Message classes. Both forms are shown here:

```
connection.register(my_account_handler, 'UpdateAccountValue')
connection.register(my_tick_handler, message.TickPrice, message.TickSize)
```

You can break the association between your handlers and messages with the unregister method, like so:

```
connection.unregister(my_tick_handler, message.TickSize)
```

In the above example, my_tick_handler will still be called with TickPrice messages.

Connection objects also allow you to associate a handler with all messages generated. The call looks like this:

```
connection.registerAll(my_generic_handler)
```

And of course, there's an unregisterAll method as well:

```
connection.unregisterAll(my_generic_handler)
```

Attributes

The Connection class exposes the attributes of its connection, so you can write:

```
connection.reqIds()
```

Logging

The Connection class provides a basic logging facility (via the Python logging module). To activate it, call it like this:

```
connection.enableLogging()
```

To deactivate logging, call the same method with False as the first parameter:

```
connection.enableLogging(False)
```

Message Objects

Your handlers are passed a single parameter, an instance of the Message class (or one of its subclasses). These instances will have attributes that match the parameter names from the underlying method call. For example, when you're passed a Message instance generated from a TickSize call, the object might look like this:

```
msg.tickerId = 4
msg.field = 3
msg.size = 100
```

1.3 IbPy Online Documentation

The online IbPy Documentation though difficult to understand is quite useful as a reference.

Chapter 2

Connection and Server

The IB API online reference is always updated. Hence it is best to consult it regularly for changes. Interactive Brokers API[1]. We will be using the Java portion as our points of reference as IbPy was built from the Java API. I will also be using the IbPy optional interface as well. I find it more intuitive for a non software engineer like me. Interactive Brokers (IB) provides Java EClientSocket Methods for extracting data. The Java EWrapper Methods returns or receives the information to Python.

[1] `https://www.interactivebrokers.com/en/software/api/api.htm`

Chapter 3

Market Data

The IB API online reference is always updated. Hence it is best to consult it regularly for changes. Interactive Brokers API[1]. We will be using the Java portion as our points of reference as IbPy was built from the Java API. I will also be using the IbPy optional interface as well. I find it more intuitive for a non software engineer like me. Interactive Brokers (IB) provides Java EClientSocket Methods for extracting data. The Java EWrapper Methods returns or receives the information to Python.

3.1 Introduction

```
'''
accountDownloadEnd      - indicator that inform us acocunt download has ended.
0 - not yet. 1 - yes. ended
managedAccountsNLV - NLV of all managed accounts. It is stored in DataFrame format
comm - commission
managedAccountsList - List of all managed accounts
nextValidId - IB response indicating the next valid ID for orders
openOrderEnd - indicator that inform us open order feed has ended.
0 - not yet. 1 - yes. ended
positions - variable to hold all data feed from IB
portfolio_pos - DataFrame to store all portfolio positions
positionEnd - indicator that inform us position feed has ended.
0 - not yet. 1 - yes. ended
AccountValue - returns all the info about an account.
Can only perform one account at a time based on account that we submit via request
portfolio - portfolio info of specific account that we submit via request
UpdateAccountTime - Account last update time
Mkt_bid - Market bid price of the contract that we request info for
Mkt_ask - Market last price of the contract that we request info for
Mkt_last - Market last price of the contract that we request info for
Mkt_high - Market high price of the contract that we request info for
Mkt_low - Market low price of the contract that we request info for
Mkt_close - Market close price of the contract that we request info for
'''
from ib.opt import Connection, message
import pandas as pd
from datetime import datetime
import time
from ib.ext.ExecutionFilter import ExecutionFilter
```

[1]https://www.interactivebrokers.com/en/software/api/api.htm

```
from ib.ext.Contract import Contract

class err:
    all = []
    accountDownloadEnd = 0
    accountSummaryEnd = 0
    AccountValue = []
    comm = []
    exec_rpt = []
    managedAccountsNLV = []
    nextValidId = 0
    openOrderEnd = 0
    positions = []
    positionEnd = 0
    portfolio = []
    portfolio_pos = []
    store_msg = []
    UpdateAccountTime = []


    def return_message(self,msg):
        self.store_msg = msg
        if msg.errorCode == 504:
            print("Error Code:" + str(msg.errorCode) + " IB Not Connected")
        elif msg.errorCode == 2104 or 2106:
            print("all ok: " + str(msg.errorCode) + " " + str(msg.errorMsg))
        else:
            print(msg)

    def reply_handler(self,msg):
        self.all.append(msg)
        if msg.typeName == "accountDownloadEnd":
            self.accountDownloadEnd = 1
        elif msg.typeName == "accountSummary":
            if msg.tag == "NetLiquidation":
                self.managedAccountsNLV.append([msg.account, msg.value,msg.currency])
        elif msg.typeName == "accountSummaryEnd":
            self.accountSummaryEnd = 1
            self.managedAccountsNLV = pd.DataFrame(self.managedAccountsNLV,
                                                   columns =
                                                   ["Account","NLV","Currency"])
            self.managedAccountsNLV["NLV"] = self.managedAccountsNLV["NLV"].astype("float")
            self.managedAccountsNLV["Date"] = datetime.now().strftime("%d/%m/%Y %H:%M")
            tmp = pd.read_csv("/data/tmp.csv",index_col=0)
            tmp = tmp.append(self.managedAccountsNLV)
            tmp.to_csv("/data/tmp.csv")
        elif msg.typeName == "commission":
            print "comm"
            self.comm.append(msg)
        elif msg.typeName == "managedAccounts":
            self.managedAccountsList = filter(None,
                                              [x.strip() for x in msg. \
                                              accountsList.split(',')])
        elif msg.typeName == "nextValidId":
            self.nextValidId = msg.orderId
        elif msg.typeName == "openOrder":
            pass
            # contract, keys, order, orderId, orderState
        elif msg.typeName == "openOrderEnd":
            self.openOrderEnd = 1
        elif msg.typeName == "orderStatus":
            pass
            # avgFillPrice, clientId, filled, lastFillPrice, orderId, parentId,
            # permId, remaining, status, typeName, whyHeld
        elif msg.typeName == "position":
            self.positions.append(msg)
            self.portfolio_pos.append([msg.account,
                                      msg.contract.m_conId,
                                      msg.contract.m_currency,
                                      msg.contract.m_exchange,
                                      msg.contract.m_expiry,
                                      msg.contract.m_includeExpired,
                                      msg.contract.m_localSymbol,
                                      msg.contract.m_multiplier,
                                      msg.contract.m_right,
                                      msg.contract.m_secType,
                                      msg.contract.m_strike,
                                      msg.contract.m_symbol,
                                      msg.contract.m_tradingClass,
                                      msg.pos,
                                      msg.avgCost])
        elif msg.typeName == "positionEnd":
            self.positionEnd = 1
        elif msg.typeName == "updateAccountValue":
            self.AccountValue.append([msg.accountName,
                                      msg.currency,
                                      msg.key,
                                      msg.typeName,
                                      msg.value])
        elif msg.typeName == "updatePortfolio":
            self.portfolio.append([msg.accountName,
                                   msg.averageCost,
                                   msg.contract.m_comboLegs,
                                   msg.contract.m_comboLegsDescrip,
                                   msg.contract.m_conId,
                                   msg.contract.m_currency,
                                   msg.contract.m_exchange,
                                   msg.contract.m_expiry,
                                   msg.contract.m_includeExpired,
                                   msg.contract.m_localSymbol,
                                   msg.contract.m_multiplier,
                                   msg.contract.m_primaryExch,
                                   msg.contract.m_right,
                                   msg.contract.m_secId,
                                   msg.contract.m_secIdType,
                                   msg.contract.m_secType,
                                   msg.contract.m_strike,
                                   msg.contract.m_symbol,
                                   msg.contract.m_tradingClass,
                                   msg.contract.m_underComp,
                                   msg.marketPrice,
                                   msg.marketValue,
                                   msg.position,
                                   msg.realizedPNL,
                                   msg.typeName,
                                   msg.unrealizedPNL])
        elif msg.typeName == "updateAccountTime":
            self.UpdateAccountTime = msg.timeStamp
        elif msg.typeName == "tickPrice":
            if msg.field == 1:
                self.Mkt_bid = msg.price
            elif msg.field == 2:
                self.Mkt_ask = msg.price
            elif msg.field == 4:
                self.Mkt_last = msg.price
            elif msg.field == 6:
                self.Mkt_high = msg.price
            elif msg.field == 7:
                self.Mkt_low = msg.price
            elif msg.field == 9:
                self.Mkt_close = msg.price
            else:
                pass

        else:
            #print(msg)
            #print("Other Messages: " + str(msg))
            pass



def make_contract(symbol, secType):
    contract = Contract()
    if secType == "STK":
        contract.m_symbol = symbol
        contract.m_secType = 'STK'
        contract.m_exchange = 'SMART'
        contract.m_primaryExch = 'SMART'
        contract.m_currency = 'USD'
        contract.m_localSymbol = symbol
    elif secType == "CASH":
        contract.m_symbol = symbol
        contract.m_secType = secType
        contract.m_exchange = 'IDEALPRO'
        contract.m_currency = 'USD'
    return contract

def exec_filter(client_id):
    contract = make_contract('EUR', "CASH")
    filt = ExecutionFilter()
    filt.m_clientId = client_id
    filt.m_acctCode = "DU254946"
    #filt.m_time = "20160122-00:00:00"
    filt.m_symbol = contract.m_symbol
    filt.m_secType = contract.m_secType
    filt.m_exchange = contract.m_exchange
    return filt

def exec_info(msg):
    global exec_tmp
    exec_tmp.append(msg)
    msg.contract.__dict__
    msg.contract.m_currency
    msg.contract.m_exchange
    msg.contract.m_expiry
    msg.contract.m_includeExpired
    msg.contract.m_localSymbol
    msg.contract.m_multiplier
    msg.contract.m_right
    msg.contract.m_secType
    msg.contract.m_strike
    msg.contract.m_symbol
    msg.contract.m_tradingClass
    msg.execution.__dict__
    msg.contract.m_acctNumber
    msg.contract.m_avgPrice
    msg.contract.m_clientId
    msg.contract.m_cumQty
    msg.contract.m_evMultiplier
    msg.contract.m_evRule
    msg.contract.m_exchange
    msg.contract.m_execId
    msg.contract.m_liquidation
    msg.contract.m_orderId
    msg.contract.m_orderRef
    msg.contract.m_permId
    msg.contract.m_price
    msg.contract.m_shares
    msg.contract.m_side
    msg.contract.m_time
    print msg.contract.m_symbol
    print msg.execution.m_cumQty

def comm_info(msg):
    global comm_tmp
    comm_tmp.append(msg)
    msg.commissionReport.m_commission
    msg.m_currency
    msg.m_execId
    print msg.commissionReport.m_commission


if __name__ == "__main__":
    ACCT_NO = "DU254946"
    ib = err()
    conn = Connection.create(port=4001, clientId = 136)
    conn.register(ib.return_message,'Error')
    conn.register(exec_info, message.execDetails)
    conn.registerAll(ib.reply_handler)
    conn.connect()

    contract = make_contract("EUR","CASH")
    print ib.nextValidId
    time.sleep(2)
#     conn.reqMktData(ib.nextValidId, contract, "", False)
    conn.reqMktData(5, contract, "", False)
    time.sleep(2)
    conn.reqAccountSummary(1,"All","NetLiquidation")
    conn.reqPositions()
    conn.reqExecutions(0, exec_filter(136))
    conn.reqAccountUpdates(1, ACCT_NO)
    conn.reqOpenOrders()
    conn.reqAllOpenOrders()
    time.sleep(2)
    # conn.reqMktDepth(100, contract, 5)
    # conn.reqRealTimeBars(200, contract, 5, "TRADES", 0)
    try:
        while 1:
            #print ib.all[-1]
            print ib.Mkt_bid, ib.Mkt_ask
    except (KeyboardInterrupt, ):
        conn.disconnect()
        print('\nKeyboard interrupt.\n')

    # probably not the best way to append...
    ib.portfolio_pos = pd.DataFrame(ib.portfolio_pos,
                                    columns = ["accountName",
                                               "conId",
                                               "currency",
                                               "exchange",
                                               "expiry",
                                               "includeExpired",
                                               "localSymbol",
                                               "multiplier",
                                               "right",
                                               "secType",
                                               "strike",
                                               "symbol",
                                               "tradingClass",
                                               "position",
                                               "avgCost"])
    ib.portfolio = pd.DataFrame(ib.portfolio,
                                columns = ["accountName",
                                           "averageCost",
                                           "m_comboLegs",
                                           "m_comboLegsDescrip",
                                           "m_conId",
                                           "m_currency",
                                           "m_exchange",
                                           "m_expiry",
                                           "m_includeExpired",
                                           "m_localSymbol",
                                           "m_multiplier",
                                           "m_primaryExch",
                                           "m_right",
                                           "m_secId",
                                           "m_secIdType",
                                           "m_secType",
                                           "m_strike",
                                           "m_symbol",
                                           "m_tradingClass",
                                           "m_underComp",
                                           "marketPrice",
                                           "marketValue",
                                           "position",
                                           "realizedPNL",
                                           "typeName",
                                           "unrealizedPNL"])
```

```
Server Version: 76
TWS Time at connection:20160126 12:52:29 SGT
0
all ok: 2104 Market data farm connection is OK:cashfarm
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
1.0849 1.085
```

```

ib.AccountValue
Out[63]:
[['DU254946', 'USD', 'NetLiquidation', 'updateAccountValue', '793760.80'],
 ['DU254946', None, 'AccountType', 'updateAccountValue', 'INDIVIDUAL'],
 ['DU254946', None, 'AccountCode', 'updateAccountValue', 'DU254946'],
 ['DU254946', None, 'AccountReady', 'updateAccountValue', 'true'],
 ['DU254946', None, 'Cushion', 'updateAccountValue', '0.987874'],
 ['DU254946', None, 'DayTradesRemaining', 'updateAccountValue', '-1'],
 ['DU254946', None, 'DayTradesRemainingT+1', 'updateAccountValue', '-1'],
 ['DU254946', None, 'DayTradesRemainingT+2', 'updateAccountValue', '-1'],
 ['DU254946', None, 'DayTradesRemainingT+3', 'updateAccountValue', '-1'],
 ['DU254946', None, 'DayTradesRemainingT+4', 'updateAccountValue', '-1'],
```

```
ib.Mkt_ask
Out[64]: 1.085
ib.Mkt_bid
Out[65]: 1.0849
ib.Mkt_close
Out[66]: 1.085
ib.Mkt_high
Out[67]: 1.08595
ib.Mkt_low
Out[68]: 1.08415
ib.UpdateAccountTime
Out[69]: '12:50'
```

```
ib.managedAccountsList
Out[75]:
```

```
['DI246990',
 'DU254946',
 'DU254949',
 'DU254959',
 'DU254980',
 'DU255100',
 'DU255105',
 'DU255156',
 'DU255276',
 'DU255277',
 'DU255278',
 'DU255279',
 'DU255280',
...
 'DU307220',
 'DU307223',
 'DU307226',
 'DU307235',
 'DU307237',
 'DU307239',
 'DU307257',
 'DU307364']
```

```
ib.managedAccountsNLV
Out[76]:
      Account          NLV  Currency              Date
0    DI246990   1006055.32       USD  26/01/2016 12:49
1    DU254946    793760.80       USD  26/01/2016 12:49
2    DU254949    321747.10       USD  26/01/2016 12:49
3    DU254959   1004405.26       USD  26/01/2016 12:49
4    DU254980   1064036.28       USD  26/01/2016 12:49
5    DU255100     58770.92       USD  26/01/2016 12:49
6    DU255105   1000000.00       USD  26/01/2016 12:49
7    DU255156    913521.67       USD  26/01/2016 12:49
8    DU255276    806296.24       USD  26/01/2016 12:49
9    DU255277    269258.99       USD  26/01/2016 12:49
10   DU255278   1082883.50       USD  26/01/2016 12:49
11   DU255279    251658.01       USD  26/01/2016 12:49
12   DU255280   1121567.53       USD  26/01/2016 12:49
13   DU255282    668808.30       USD  26/01/2016 12:49
14   DU255423   1000000.00       USD  26/01/2016 12:49
15   DU255426    957989.38       USD  26/01/2016 12:49
16   DU255451   1000000.00       USD  26/01/2016 12:49
17   DU255575   1000000.00       USD  26/01/2016 12:49
18   DU255601   1024372.59       USD  26/01/2016 12:49
19   DU255646   1000000.00       USD  26/01/2016 12:49
20   DU255649    646191.63       USD  26/01/2016 12:49
21   DU255671   1000000.00       USD  26/01/2016 12:49
22   DU255675    952316.72       USD  26/01/2016 12:49
23   DU255799   1000000.00       USD  26/01/2016 12:49
24   DU255847    375491.49       USD  26/01/2016 12:49
25   DU256052    996959.65       USD  26/01/2016 12:49
26   DU261464    337910.69       USD  26/01/2016 12:49
27   DU261465    254248.16       USD  26/01/2016 12:49
28   DU261467   1010921.14       USD  26/01/2016 12:49
29   DU261468    739141.28       USD  26/01/2016 12:49
..        ...          ...       ...               ...
```

```
52  DU274370  1106102.14      USD  26/01/2016 12:49
53  DU274371  1223332.62      USD  26/01/2016 12:49
54  DU274373  1000553.50      USD  26/01/2016 12:49
55  DU274374   868818.60      USD  26/01/2016 12:49
56  DU274375   847001.28      USD  26/01/2016 12:49
57  DU274383   943022.84      USD  26/01/2016 12:49
58  DU274385   999575.03      USD  26/01/2016 12:49
59  DU274388   998197.27      USD  26/01/2016 12:49
60  DU274390  1220112.86      USD  26/01/2016 12:49
61  DU278185   361088.30      USD  26/01/2016 12:49
62  DU301744  1000000.00      USD  26/01/2016 12:49
63  DU302326  1019276.75      USD  26/01/2016 12:49
64  DU302430  1009453.96      USD  26/01/2016 12:49
65  DU303696  1003159.26      USD  26/01/2016 12:49
66  DU305139  1001465.04      USD  26/01/2016 12:49
67  DU305370  1011638.36      USD  26/01/2016 12:49
68  DU305371  1000780.20      USD  26/01/2016 12:49
69  DU305398  1004990.88      USD  26/01/2016 12:49
70  DU307150  1011025.25      USD  26/01/2016 12:49
71  DU307174   989908.63      USD  26/01/2016 12:49
72  DU307211  1002808.75      USD  26/01/2016 12:49
73  DU307218  1004497.41      USD  26/01/2016 12:49
74  DU307220  1001553.00      USD  26/01/2016 12:49
75  DU307223  1014062.80      USD  26/01/2016 12:49
76  DU307226  1003316.70      USD  26/01/2016 12:49
77  DU307235   979535.62      USD  26/01/2016 12:49
78  DU307237  1019218.50      USD  26/01/2016 12:49
79  DU307239  1009227.13      USD  26/01/2016 12:49
80  DU307257   988867.90      USD  26/01/2016 12:49
81  DU307364   994265.55      USD  26/01/2016 12:49

[82 rows x 4 columns]
```

```
ib.portfolio_pos
Out[78]:
    accountName      conId currency exchange    expiry includeExpired  \
0      DU274375     270639      USD   NASDAQ      None          False
1      DU274374   12087792      USD     None      None          False
2      DU274374  173710102      USD   NASDAQ      None          False
3      DU307220  130651996      USD     NYSE      None          False
4      DU307220   15124833      USD   NASDAQ      None          False
5      DU307220     272093      USD   NASDAQ      None          False
6      DU274373   14433401      USD     None      None          False
7      DU274373    4725951      USD     NYSE      None          False
8      DU307223   12087797      USD     None      None          False
9      DU307223   15016059      JPY     None      None          False
10     DU261464  177525433      USD     None  20160318          False
11     DU261467   14433401      USD     None      None          False
12     DU274370   14433401      USD     None      None          False
13     DU274370     265598      USD   NASDAQ      None          False
14     DU274370   72687598      USD   NASDAQ      None          False
15     DU274370  113342317      CNH     None      None          False
16     DU255426  137935324      USD     ARCA      None          False
17     DU274364  177525433      USD     None  20160318          False
18     DU305371   12087792      USD     None      None          False
19     DU305371  147555266      USD     None  20160225          False
```

```
ib.portfolio_pos.columns
Out[79]:
Index([u'accountName', u'conId', u'currency', u'exchange', u'expiry',
       u'includeExpired', u'localSymbol', u'multiplier', u'right', u'secType',
       u'strike', u'symbol', u'tradingClass', u'position', u'avgCost'],
      dtype='object')
```

Chapter 4

Orders

The IB API online reference is always updated. Hence it is best to consult it regularly for changes. Interactive Brokers API[1]. We will be using the Java portion as our points of reference as IbPy was built from the Java API. I will also be using the IbPy optional interface as well.

4.1 Introduction

Under Orders, there are 8 Methods available. The five functions are `placeOrder()`, `cancelOrder()`, `reqOpenOrders()`, `reqAllOpenOrders()`, `reqAutoOpenOrders()`, `reqIDs()`, `exerciseOptions()`, and `reqGlobalCancel()`. The table below links the EClientSocket Methods (**information requestor**) with the corresponding EWrapper Methods (**callback method**).

EClientSocket Methods	EWrapper Methods

placeOrder()
cancelOrder()	

reqOpenOrders()
reqAllOpenOrders()
redAutoOpenOrders()
	orderStatus()
	openOrder()
	openorderEnd()

reqIDs()	nextValidID()
exerciseOptions()	
reqGlobalCancel()	
	deltaNeutralValidation()

Things we want to do:-

- place order
- cancel order
- Cancel all orders
- Track orders such as find out any open orders & auto open orders (?)
- Request the next valid ID for order placement
- Exercise option

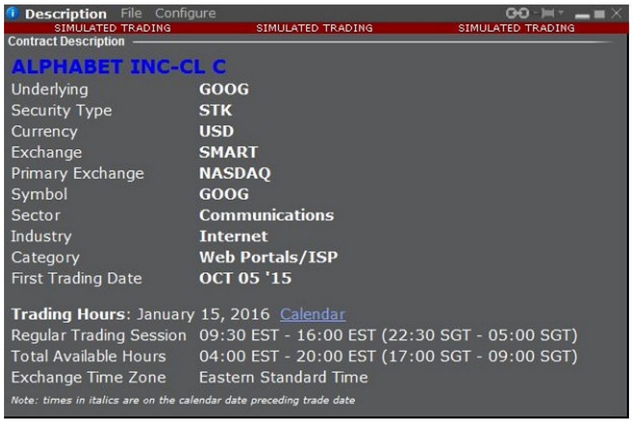
[1] `https://www.interactivebrokers.com/en/software/api/api.htm`

4.1.1 Asset Classes

Often, when we do not know what are the input values required for certain stock for the `placeOrder()` method. One way is to go to TWS and select **Trading Tools** > **Watchlist**. In **Watchlist**, under contract enter a symbol (E.g., AAPL) and select stock (SMART). On AAPLE, right click your mouse and select **Contract Info** > Description and you should something similar to the pictures below.

Another alternative is to access the online guide[2] provided by interactive brokers. **Stocks**

	ALPHABET INC-CL C	**AAPLE INC**
Underlying	GOOG	AAPL
Security Type*	STK	STK
Currency*	USD	USD
Exchange*	SMART	SMART
Primary Exchange	NASDAQ	NASDAQ
Symbol*	GOOG	AAPL
Sector	Communications	Technology
Industry	Internet	Computers
Category	Web Portals/ISP	Computers



Contract Description

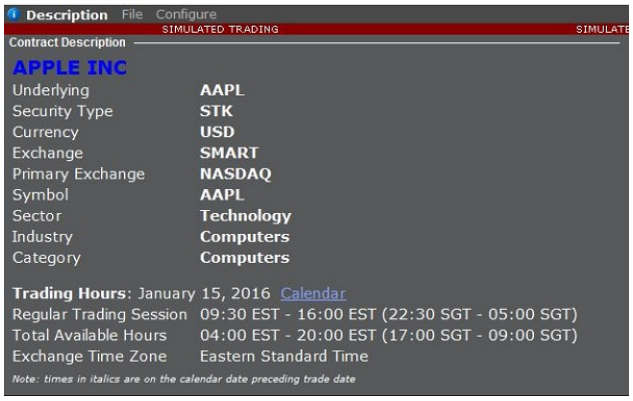
ALPHABET INC-CL C

Underlying	GOOG
Security Type	STK
Currency	USD
Exchange	SMART
Primary Exchange	NASDAQ
Symbol	GOOG
Sector	Communications
Industry	Internet
Category	Web Portals/ISP
First Trading Date	OCT 05 '15

Trading Hours: January 15, 2016 Calendar

Regular Trading Session	09:30 EST - 16:00 EST (22:30 SGT - 05:00 SGT)
Total Available Hours	04:00 EST - 20:00 EST (17:00 SGT - 09:00 SGT)
Exchange Time Zone	Eastern Standard Time

Note: times in italics are on the calendar date preceding trade date



Contract Description

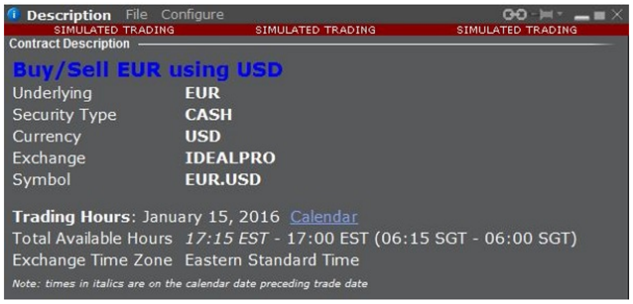
APPLE INC

Underlying	AAPL
Security Type	STK
Currency	USD
Exchange	SMART
Primary Exchange	NASDAQ
Symbol	AAPL
Sector	Technology
Industry	Computers
Category	Computers

Trading Hours: January 15, 2016 Calendar

Regular Trading Session	09:30 EST - 16:00 EST (22:30 SGT - 05:00 SGT)
Total Available Hours	04:00 EST - 20:00 EST (17:00 SGT - 09:00 SGT)
Exchange Time Zone	Eastern Standard Time

Note: times in italics are on the calendar date preceding trade date



Contract Description

Buy/Sell EUR using USD

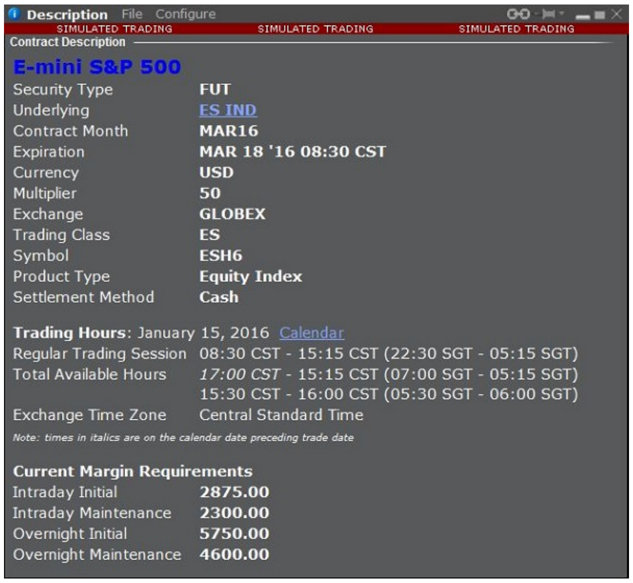
Underlying	EUR
Security Type	CASH
Currency	USD
Exchange	IDEALPRO
Symbol	EUR.USD

Trading Hours: January 15, 2016 Calendar

Total Available Hours	*17:15 EST* - 17:00 EST (06:15 SGT - 06:00 SGT)
Exchange Time Zone	Eastern Standard Time

Note: times in italics are on the calendar date preceding trade date

[2] `https://www.interactivebrokers.com/en/index.php?f=products&p=stk`


Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING

Contract Description

E-mini S&P 500

Security Type	FUT
Underlying	ES IND
Contract Month	MAR16
Expiration	MAR 18 '16 08:30 CST
Currency	USD
Multiplier	50
Exchange	GLOBEX
Trading Class	ES
Symbol	ESH6
Product Type	Equity Index
Settlement Method	Cash

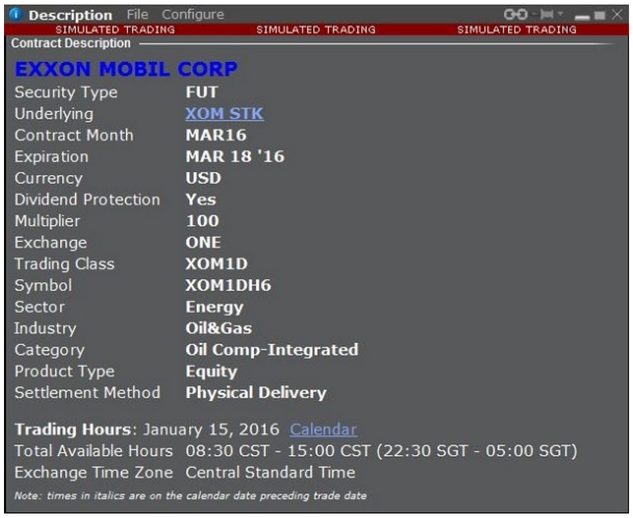
Trading Hours: January 15, 2016 Calendar

Regular Trading Session	08:30 CST - 15:15 CST (22:30 SGT - 05:15 SGT)
Total Available Hours	*17:00 CST* - 15:15 CST (07:00 SGT - 05:15 SGT)
	15:30 CST - 16:00 CST (05:30 SGT - 06:00 SGT)
Exchange Time Zone	Central Standard Time

Note: times in italics are on the calendar date preceding trade date

Current Margin Requirements

Intraday Initial	2875.00
Intraday Maintenance	2300.00
Overnight Initial	5750.00
Overnight Maintenance	4600.00


Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING

Contract Description

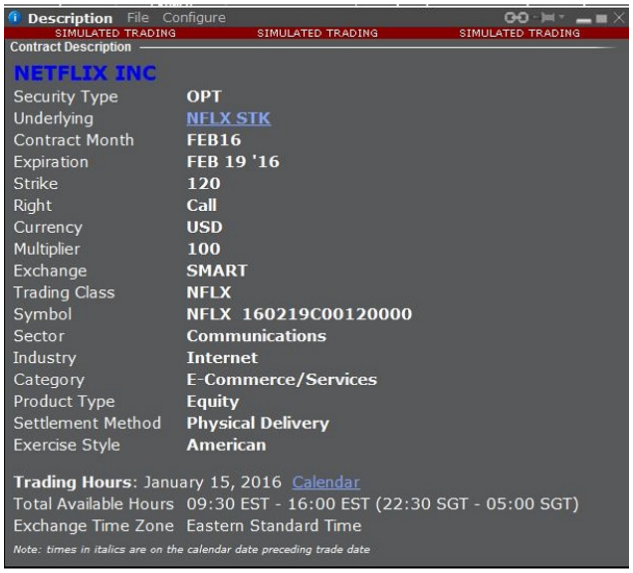
EXXON MOBIL CORP

Security Type	FUT
Underlying	XOM STK
Contract Month	MAR16
Expiration	MAR 18 '16
Currency	USD
Dividend Protection	Yes
Multiplier	100
Exchange	ONE
Trading Class	XOM1D
Symbol	XOM1DH6
Sector	Energy
Industry	Oil&Gas
Category	Oil Comp-Integrated
Product Type	Equity
Settlement Method	Physical Delivery

Trading Hours: January 15, 2016 Calendar

Total Available Hours	08:30 CST - 15:00 CST (22:30 SGT - 05:00 SGT)
Exchange Time Zone	Central Standard Time

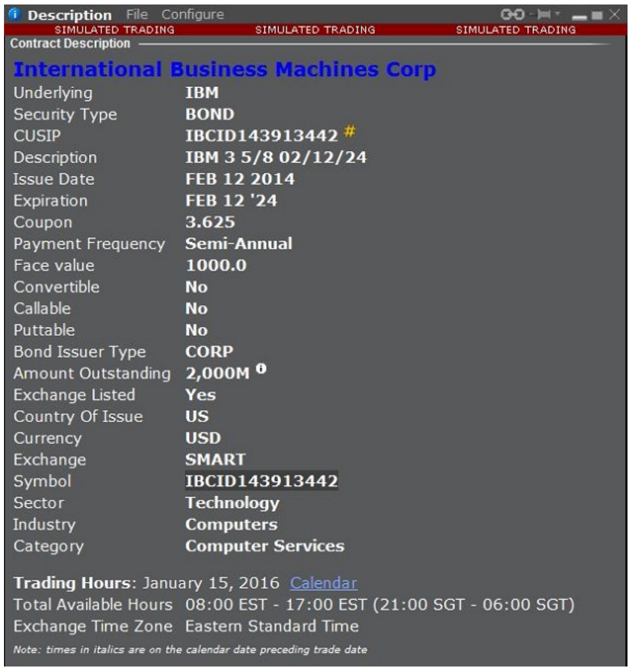
Note: times in italics are on the calendar date preceding trade date


Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING
Contract Description

NETFLIX INC

Security Type	OPT
Underlying	NFLX STK
Contract Month	FEB16
Expiration	FEB 19 '16
Strike	120
Right	Call
Currency	USD
Multiplier	100
Exchange	SMART
Trading Class	NFLX
Symbol	NFLX 160219C00120000
Sector	Communications
Industry	Internet
Category	E-Commerce/Services
Product Type	Equity
Settlement Method	Physical Delivery
Exercise Style	American

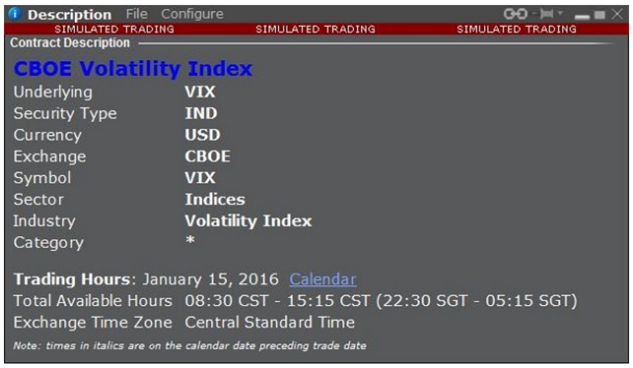
Trading Hours: January 15, 2016 Calendar
Total Available Hours 09:30 EST - 16:00 EST (22:30 SGT - 05:00 SGT)
Exchange Time Zone Eastern Standard Time
Note: times in italics are on the calendar date preceding trade date


Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING
Contract Description

International Business Machines Corp

Underlying	IBM
Security Type	BOND
CUSIP	IBCID143913442 #
Description	IBM 3 5/8 02/12/24
Issue Date	FEB 12 2014
Expiration	FEB 12 '24
Coupon	3.625
Payment Frequency	Semi-Annual
Face value	1000.0
Convertible	No
Callable	No
Puttable	No
Bond Issuer Type	CORP
Amount Outstanding	2,000M
Exchange Listed	Yes
Country Of Issue	US
Currency	USD
Exchange	SMART
Symbol	IBCID143913442
Sector	Technology
Industry	Computers
Category	Computer Services

Trading Hours: January 15, 2016 Calendar
Total Available Hours 08:00 EST - 17:00 EST (21:00 SGT - 06:00 SGT)
Exchange Time Zone Eastern Standard Time
Note: times in italics are on the calendar date preceding trade date

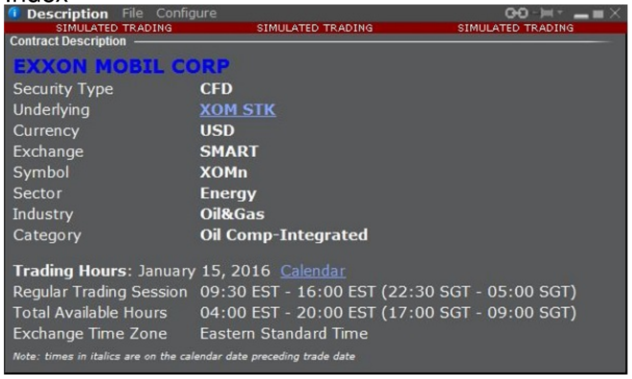

Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING
Contract Description

CBOE Volatility Index

Underlying	VIX
Security Type	IND
Currency	USD
Exchange	CBOE
Symbol	VIX
Sector	Indices
Industry	Volatility Index
Category	*

Trading Hours: January 15, 2016 Calendar
Total Available Hours 08:30 CST - 15:15 CST (22:30 SGT - 05:15 SGT)
Exchange Time Zone Central Standard Time
Note: times in italics are on the calendar date preceding trade date

Index


Description File Configure

SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING

Contract Description

EXXON MOBIL CORP

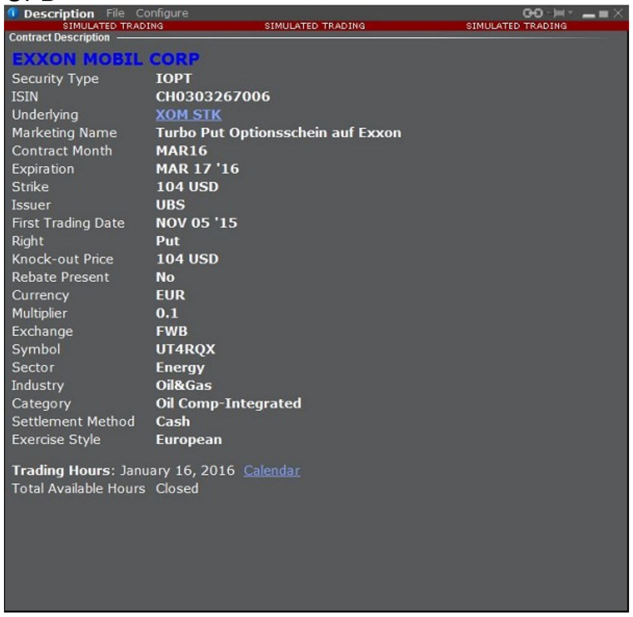
Security Type	CFD
Underlying	XOM STK
Currency	USD
Exchange	SMART
Symbol	XOMn
Sector	Energy
Industry	Oil&Gas
Category	Oil Comp-Integrated

Trading Hours: January 15, 2016 Calendar

Regular Trading Session	09:30 EST - 16:00 EST (22:30 SGT - 05:00 SGT)
Total Available Hours	04:00 EST - 20:00 EST (17:00 SGT - 09:00 SGT)
Exchange Time Zone	Eastern Standard Time

Note: times in italics are on the calendar date preceding trade date

CFD


Description File Configure

SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING

Contract Description

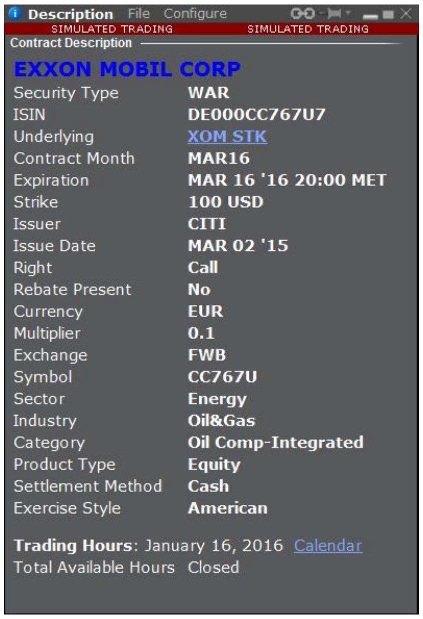
EXXON MOBIL CORP

Security Type	IOPT
ISIN	CH0303267006
Underlying	XOM STK
Marketing Name	Turbo Put Optionsschein auf Exxon
Contract Month	MAR16
Expiration	MAR 17 '16
Strike	104 USD
Issuer	UBS
First Trading Date	NOV 05 '15
Right	Put
Knock-out Price	104 USD
Rebate Present	No
Currency	EUR
Multiplier	0.1
Exchange	FWB
Symbol	UT4RQX
Sector	Energy
Industry	Oil&Gas
Category	Oil Comp-Integrated
Settlement Method	Cash
Exercise Style	European

Trading Hours: January 16, 2016 Calendar

Total Available Hours	Closed

Structured Products

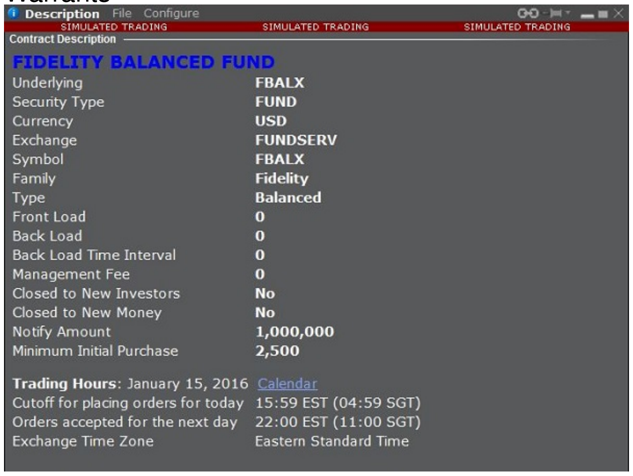

Description File Configure
SIMULATED TRADING SIMULATED TRADING
Contract Description

EXXON MOBIL CORP

Security Type	WAR
ISIN	DE000CC767U7
Underlying	XOM STK
Contract Month	MAR16
Expiration	MAR 16 '16 20:00 MET
Strike	100 USD
Issuer	CITI
Issue Date	MAR 02 '15
Right	Call
Rebate Present	No
Currency	EUR
Multiplier	0.1
Exchange	FWB
Symbol	CC767U
Sector	Energy
Industry	Oil&Gas
Category	Oil Comp-Integrated
Product Type	Equity
Settlement Method	Cash
Exercise Style	American

Trading Hours: January 16, 2016 Calendar
Total Available Hours Closed

Warrants


Description File Configure
SIMULATED TRADING SIMULATED TRADING SIMULATED TRADING
Contract Description

FIDELITY BALANCED FUND

Underlying	FBALX
Security Type	FUND
Currency	USD
Exchange	FUNDSERV
Symbol	FBALX
Family	Fidelity
Type	Balanced
Front Load	0
Back Load	0
Back Load Time Interval	0
Management Fee	0
Closed to New Investors	No
Closed to New Money	No
Notify Amount	1,000,000
Minimum Initial Purchase	2,500

Trading Hours: January 15, 2016 Calendar
Cutoff for placing orders for today 15:59 EST (04:59 SGT)
Orders accepted for the next day 22:00 EST (11:00 SGT)
Exchange Time Zone Eastern Standard Time

4.2 Placing Orders and Examples

4.2.1 Placing a Stock Order

```
"""
/IbPy/ibpy scripts/ibpy_Order_STK.py"""
from ib.ext.Contract import Contract
from ib.ext.Order import Order
from ib.opt import Connection, message
import time

def error_handler(msg):
    print "Error : %s" % msg

def reply_handler(msg):
    print "Server Response: %s, %s" % (msg.typeName, msg)

def listen_order(msg):
    global tmp
    tmp.append(msg)

def create_contract(symbol, secType, exchange, currency):
    contract = Contract()
    contract.m_symbol = symbol
    contract.m_secType = secType
    contract.m_exchange = exchange
    contract.m_currency = currency
    return contract

def create_order(account, orderType, totalQuantity, action):
    order = Order()
    order.m_account = account
    order.m_orderType = orderType
    order.m_totalQuantity = totalQuantity
    order.m_action = action
    return order

if __name__ == "__main__":
    tmp = []
    tws_conn = Connection.create(port=4001, clientId=125)
    tws_conn.connect()
    tws_conn.register(listen_order, message.orderStatus)
    tws_conn.register(error_handler, 'Error')
    tws_conn.registerAll(reply_handler)

    order_id = 21

    contract_info = create_contract('GOOG', 'STK', 'SMART', 'USD')
    order_info = create_order("DU274390", 'MKT', 100, 'BUY')
    tws_conn.placeOrder(order_id, contract_info, order_info)
    time.sleep(3)
    tws_conn.disconnect()

    print(tmp)
```

In this example, we placed an order to purchase **100** shares of **GOOG**, which is a **STK** (stock) via the **SMART** exchange and it is in **USD** currency. The account we placed this **MKT BUY** (Market Buy) order is **DU274390**[3]. We used two functions to create the object with the relevant information required by IB. The first function is `create_contract` which create a contract object. In this example we instantiate it with `contract_info`. We can dig into the `contract_info` object by using `dir()` as can be seen here.

```
dir(contract_info)
Out[39]:
['_Contract__init___0',
 '__class__',
 '__delattr__',
 '__dict__',
 '__doc__',
 '__eq__',
 '__format__',
 '__getattribute__',
 '__hash__',
 '__init__',
 '__module__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'clone',
 'm_comboLegs',
 'm_comboLegsDescrip',
 'm_conId',
 'm_currency',
 'm_exchange',
 'm_expiry',
 'm_includeExpired',
 'm_localSymbol',
 'm_multiplier',
 'm_primaryExch',
 'm_right',
 'm_secId',
 'm_secIdType',
 'm_secType',
 'm_strike',
 'm_symbol',
 'm_tradingClass',
 'm_underComp']
```

The minimum contract information for Equity order required **symbol, security type, exchange and currency**:

```
def create_contract(symbol, secType, exchange, currency):
contract_info = create_contract('GOOG', 'STK', 'SMART', 'USD')
```

To create an order, it required information on **account, order type, total quantity and action**:

```
def create_order(account, orderType, totalQuantity, action):
order_info = create_order("DU274390", 'MKT', 100, 'BUY')
```

[3]Naturally, use your own account number here. Normally, you do not need to specify account number if you only operate one account. However, the account I logged on is an instructor account, which another word is an institutional account.

For a more comprehensive guide, please refer to Java SocketClient Properties in section **??**. The output from running the program:

```
Server Version: 76
TWS Time at connection:20160115 13:41:04 SGT
```

Following is the message from the server running the listerner function `managedAccounts()`. Every time we connect to the server, `managedAccounts()` will be called.

```
Server Response: managedAccounts, <managedAccounts accountsList=DI246990,DU249087,
DU249131,DU249147,DU249177,DU249290,DU249298,DU249300,DU249326,DU249327,
DU249340,DU249348,DU249349,DU249360,DU249365,DU249372,DU249381,DU249410,
DU249411,DU249421,DU249423,DU249424,DU249426,DU249428,DU249429,DU249430,
DU249431,DU249432,DU249443,DU249469,DU249608,DU249677,DU249785,DU250252,
DU250268,DU250445,DU250454,DU250455,DU250456,DU250457,DU250458,DU250906,
DU250908,DU251484,DU251965,DU252047,DU254813,DU254814,DU254825,DU254841,
DU254851,DU254854,DU254865,DU254873,DU254939,DU254940,DU254944,DU254945,
DU254946,DU254947,DU254948,DU254949,DU254959,DU254980,DU255100,DU255105,
DU255156,DU255276,DU255277,DU255278,DU255279,DU255280,DU255282,DU255423,
DU255426,DU255451,DU255575,DU255601,DU255646,DU255649,DU255671,DU255675,
DU255799,DU255847,DU256052,DU261464,DU261465,DU261467,DU261468,DU261471,
DU261472,DU261473,DU261604,DU261611,DU261695,DU261729,DU261751,DU262166,
DU262192,DU262601,DU262745,DU262931,DU264829,DU274362,DU274363,DU274364,
DU274365,DU274366,DU274367,DU274368,DU274369,DU274370,DU274371,DU274373,
DU274374,DU274375,DU274383,DU274385,DU274388,DU274390,DU278185,DU301744,
DU302326,>
```

Upon connection, the server response provides update on the next valid id.

```
Server Response: nextValidId, <nextValidId orderId=1>
Error : <error id=-1, errorCode=2104, errorMsg=Market data farm connection is OK:hfarm>
```

The error response that follows, which is not an error, provided us with a cryptic error code of 2104. According to the API manual 2104 error code is **A market data farm is connected.** It turns out that it is nothing serious at all. The error code of 2106 correspond to **A historical data farm is connected.**

```
Server Response: error, <error id=-1, errorCode=2104,
errorMsg=Market data farm connection is OK:hfarm>
Error : <error id=-1, errorCode=2106, errorMsg=HMDS data farm connection is OK:ushmds>
Server Response: error, <error id=-1, errorCode=2106,
errorMsg=HMDS data farm connection is OK:ushmds>
```

When we look at the next error code relating to Order message, it is indeed an error. I have placed an order before market open. According to the API manual, it is **Order message error**. It went on to explain that the order will not be placed until the market open. The reason we have this error returned to us twice is because we did the following:

```
tws_conn.register(error_handler, 'Error')
tws_conn.registerAll(reply_handler)
```

The first `register` method asked IB to return all messages with the 'Error' tag. The second `register` method asked IB to send all messages to the function `reply_handler`.

```
Error : <error id=21, errorCode=399, errorMsg=Order Message:
BUY 100 GOOG NASDAQ.NMS
Warning: your order will not be placed at the exchange until
2016-01-15 09:30:00 US/Eastern>

Server Response: error, <error id=21, errorCode=399, errorMsg=Order Message:
BUY 100 GOOG NASDAQ.NMS
Warning: your order will not be placed at the exchange until
2016-01-15 09:30:00 US/Eastern>
```

The message below informed us that there is an open order under orderId 21.

```
Server Response: openOrder,
<openOrder orderId=21,
contract=<ib.ext.Contract.Contract object at 0x000000001A2B5A90>,
order=<ib.ext.Order.Order object at 0x000000001A04DB70>,
orderState=<ib.ext.OrderState.OrderState object at 0x000000001A2B59B0>>
```

The following message keeps us informed of the order status. Currently, it is still in the **PreSubmitted** stage.

```
Server Response: orderStatus,
<orderStatus orderId=21,
status=PreSubmitted,
filled=0,
remaining=100,
avgFillPrice=0.0,
permId=719831662,
parentId=0,
lastFillPrice=0.0,
clientId=125,
whyHeld=None>
```

Lastly, the crypitc message below is a print out of the `tmp` variable.

```
[<ib.opt.message.OrderStatus object at 0x000000001A03EF48>,
<ib.opt.message.OrderStatus object at 0x000000001A03EE58>,
<ib.opt.message.OrderStatus object at 0x000000001A03EED0>,
<ib.opt.message.OrderStatus object at 0x0000000019C8CB10>]
```

There are four stored objects, each corresponds to an unfilled order. Here is a print out of the order status and details.

```
len(tmp)
Out[15]: 4

print(tmp[0])
<orderStatus orderId=23,
status=PreSubmitted,
filled=0,
remaining=100,
avgFillPrice=0.0,
permId=719831666,
parentId=0,
lastFillPrice=0.0,
clientId=126,
whyHeld=None>
```

4.2.2 Placing a FX Order

```python
"""
/IbPy/ibpy scripts/ibpy_Order_FX.py"""
from ib.ext.Contract import Contract
from ib.ext.Order import Order
from ib.opt import Connection, message
import time

def error_handler(msg):
    print "Error : %s" % msg

def reply_handler(msg):
    print "Server Response: %s, %s" % (msg.typeName, msg)

def listen_order(msg):
    global tmp
    tmp.append(msg)

def create_contract(symbol, secType, exchange, currency):
    contract = Contract()
    contract.m_symbol = symbol
    contract.m_secType = secType
    contract.m_exchange = exchange
    contract.m_currency = currency
    return contract

def create_order(account, orderType, totalQuantity, action):
    order = Order()
    order.m_account = account
    order.m_orderType = orderType
    order.m_totalQuantity = totalQuantity
    order.m_action = action
    return order

if __name__ == "__main__":
    tmp = []
    tws_conn = Connection.create(port=4001, clientId=200)
    tws_conn.connect()
    tws_conn.register(listen_order, message.orderStatus)
    tws_conn.register(error_handler, 'Error')
    tws_conn.registerAll(reply_handler)

    order_id = 201

    contract_info = create_contract('NZD', 'CASH', 'IDEALPRO', 'USD')
    order_info = create_order("DU274390", 'MKT', 5000000, 'SELL')
    tws_conn.placeOrder(order_id, contract_info, order_info)

    time.sleep(3)
    tws_conn.disconnect()

    print(tmp)
```

In what is to follow, we will see repeatedly **four** different methods being called by IB. They are `openOrder`, `orderStatus`, `execDetails` and `commissionReport`. It shows a progress update as and when an order is in the process of being filled.

Before any orders being filled:

```
Server Response: openOrder,
<openOrder orderId=201,
contract=<ib.ext.Contract.Contract object at 0x000000001A2997F0>,
order=<ib.ext.Order.Order object at 0x000000001A299C88>,
orderState=<ib.ext.OrderState.OrderState object at 0x000000001A299CC0>>
```

```
Server Response: orderStatus,
<orderStatus orderId=201,
status=Submitted,
filled=0,
remaining=5000000,
avgFillPrice=0.0,
permId=719831669,
parentId=0,
lastFillPrice=0.0,
clientId=200,
whyHeld=None>
```

```
Server Response: execDetails,
<execDetails reqId=-1,
contract=<ib.ext.Contract.Contract object at 0x000000001A299C88>,
execution=<ib.ext.Execution.Execution object at 0x000000001A299668>>
```

```
Server Response: openOrder,
<openOrder orderId=201,
contract=<ib.ext.Contract.Contract object at 0x000000001A299CC0>,
order=<ib.ext.Order.Order object at 0x000000001A299668>,
orderState=<ib.ext.OrderState.OrderState object at 0x000000001A299C18>>
```

After 1 million being filled:

```
Server Response: orderStatus,
<orderStatus orderId=201,
status=Submitted,
filled=1000000,
remaining=4000000,
avgFillPrice=0.6436,
permId=719831669,
parentId=0,
lastFillPrice=0.6436,
clientId=200,
whyHeld=None>
```

```
Server Response: commissionReport,
<commissionReport commissionReport=
<ib.ext.CommissionReport.CommissionReport object at 0x000000001A2997F0>>
```

After the order has been completedly filled:

```
Server Response: orderStatus,
<orderStatus orderId=201,
status=Filled,
filled=5000000,
remaining=0,
avgFillPrice=0.64362,
permId=719831669,
parentId=0,
lastFillPrice=0.64366,
clientId=200,
whyHeld=None>
```

The following is a print out of the data captured `listen_order`. There are quite a bit of duplication. Below is an extract of it:

```
print(tmp[0])
<orderStatus orderId=201,
status=Submitted,
filled=0,
remaining=5000000,
avgFillPrice=0.0,
permId=719831669,
parentId=0,
lastFillPrice=0.0,
clientId=200,
whyHeld=None>

print(tmp[3])
<orderStatus orderId=201,
status=Submitted,
filled=1000000,
remaining=4000000,
avgFillPrice=0.6436,
permId=719831669,
parentId=0,
lastFillPrice=0.6436,
clientId=200,
whyHeld=None>

print(tmp[6])
<orderStatus orderId=201,
status=Submitted,
filled=2000000,
remaining=3000000,
avgFillPrice=0.6436,
permId=719831669,
parentId=0,
lastFillPrice=0.6436,
clientId=200,
whyHeld=None>
```

```
print(tmp[10])
<orderStatus orderId=201,
status=Submitted,
filled=4000000,
remaining=1000000,
avgFillPrice=0.64361,
permId=719831669,
parentId=0,
lastFillPrice=0.64362,
clientId=200,
whyHeld=None>

print(tmp[13])
<orderStatus orderId=201,
status=Filled,
filled=5000000,
remaining=0,
avgFillPrice=0.64362,
permId=719831669,
parentId=0,
lastFillPrice=0.64366,
clientId=200,
whyHeld=None>
```

4.2.3 Placing a Futures Order

```
"""
/IbPy/ibpy scripts/ibpy_Order_FUT.py"""
from ib.ext.Contract import Contract
from ib.ext.Order import Order
from ib.opt import Connection, message
import time

def error_handler(msg):
    print "Error : %s" % msg

def reply_handler(msg):
    print "Server Response: %s, %s" % (msg.typeName, msg)

def listen_orderstatus(msg):
    global orderstatus_tmp
    orderstatus_tmp.append(msg)

def listen_openorder(msg):
    global openorder_tmp
    openorder_tmp.append(msg)

def comm_report(msg):
    global comm_tmp
    comm_tmp.append(msg)

def exec_info(msg):
    global exec_tmp
    exec_tmp.append(msg)

def create_contract(symbol, secType, exchange, currency, expiry):
    contract = Contract()
    contract.m_symbol = symbol
    contract.m_secType = secType
    contract.m_exchange = exchange
    contract.m_currency = currency
    contract.m_expiry = expiry
    return contract

def create_order(account, orderType, totalQuantity, action):
    order = Order()
    order.m_account = account
    order.m_orderType = orderType
    order.m_totalQuantity = totalQuantity
    order.m_action = action
    return order

if __name__ == "__main__":
    orderstatus_tmp = []
    openorder_tmp = []
    comm_tmp = []
    exec_tmp = []
    tws_conn = Connection.create(port=4001, clientId=302)
    tws_conn.connect()
    tws_conn.register(listen_orderstatus, message.orderStatus)
    tws_conn.register(listen_openorder, message.openOrder)
    tws_conn.register(comm_report, message.commissionReport)
    tws_conn.register(exec_info, message.execDetails)
    tws_conn.register(error_handler, 'Error')
    tws_conn.registerAll(reply_handler)

    order_id = 307

    contract_info = create_contract('ES', 'FUT', 'GLOBEX', 'USD', '201603')
    order_info = create_order("DU274390", 'MKT', 1, 'BUY')
    tws_conn.placeOrder(order_id, contract_info, order_info)

    time.sleep(10)
    tws_conn.disconnect()
```

Order status:

```
print(orderstatus_tmp[2])
<orderStatus orderId=306,
status=Filled,
filled=1,
remaining=0,
avgFillPrice=1896.5,
permId=719831677,
parentId=0,
lastFillPrice=1896.5,
clientId=302,
whyHeld=None>
```

Contract details:

```
openorder_tmp[0].contract.__dict__
Out[29]:
{'m_comboLegsDescrip': None,
 'm_conId': 177525433,
 'm_currency': 'USD',
 'm_exchange': 'GLOBEX',
 'm_expiry': '20160318',
 'm_includeExpired': False,
 'm_localSymbol': 'ESH6',
 'm_multiplier': '50',
 'm_right': '?',
 'm_secType': 'FUT',
 'm_strike': 0.0,
 'm_symbol': 'ES',
 'm_tradingClass': 'ES'}
```

Open Order details:

```
openorder_tmp[0].order.__dict__
Out[30]:
{'m_account': 'DU274390',
 'm_action': 'BUY',
 'm_activeStartTime': '',
 'm_activeStopTime': '',
 'm_algoStrategy': None,
 'm_allOrNone': False,
 'm_auctionStrategy': 0,
 'm_auxPrice': 0.0,
 'm_basisPoints': 2147483647,
 'm_basisPointsType': 2147483647,
'm_blockOrder': False,
'm_clearingAccount': None,
'm_clearingIntent': 'IB',
'm_clientId': 302,
'm_continuousUpdate': 0,
'm_delta': 2147483647,
'm_deltaNeutralAuxPrice': 2147483647,
'm_deltaNeutralClearingAccount': None,
'm_deltaNeutralClearingIntent': None,
'm_deltaNeutralConId': 0,
'm_deltaNeutralDesignatedLocation': None,
'm_deltaNeutralOpenClose': '?',
'm_deltaNeutralOrderType': 'None',
'm_deltaNeutralSettlingFirm': None,
'm_deltaNeutralShortSale': False,
'm_deltaNeutralShortSaleSlot': 0,
'm_designatedLocation': None,
'm_discretionaryAmt': 0.0,
'm_displaySize': 0,
'm_eTradeOnly': False,
'm_exemptCode': -1,
'm_faGroup': None,
'm_faMethod': None,
'm_faPercentage': None,
'm_faProfile': None,
'm_firmQuoteOnly': False,
'm_goodAfterTime': None,
'm_goodTillDate': None,
'm_hedgeType': None,
'm_hidden': False,
'm_lmtPrice': 1896.5,
'm_minQty': 2147483647,
'm_nbboPriceCap': 2147483647,
'm_notHeld': False,
'm_ocaGroup': None,
'm_ocaType': 3,
'm_openClose': 'O',
'm_optOutSmartRouting': False,
'm_orderId': 306,
'm_orderRef': None,
'm_orderType': 'MKT',
'm_origin': 0,
'm_outsideRth': False,
'm_parentId': 0,
'm_percentOffset': 2147483647,
'm_permId': 719831677,
'm_referencePriceType': 0,
'm_rule80A': None,
'm_scaleAutoReset': False,
'm_scaleInitFillQty': 2147483647,
'm_scaleInitLevelSize': 2147483647,
'm_scaleInitPosition': 2147483647,
'm_scalePriceAdjustInterval': 2147483647,
'm_scalePriceAdjustValue': 2147483647,
'm_scalePriceIncrement': 2147483647,
'm_scaleProfitOffset': 2147483647,
'm_scaleRandomPercent': False,
 'm_scaleSubsLevelSize': 2147483647,
 'm_scaleTable': '',
 'm_settlingFirm': None,
 'm_shortSaleSlot': 0,
 'm_startingPrice': 2147483647,
 'm_stockRangeLower': 2147483647,
 'm_stockRangeUpper': 2147483647,
 'm_stockRefPrice': 2147483647,
 'm_sweepToFill': False,
 'm_tif': 'DAY',
 'm_totalQuantity': 1,
 'm_trailStopPrice': 2147483647,
 'm_trailingPercent': 2147483647,
 'm_transmit': True,
 'm_triggerMethod': 0,
 'm_volatility': 2147483647,
 'm_volatilityType': 0,
 'm_whatIf': False}
```

Open Order State details:

```
openorder_tmp[0].orderState.__dict__
Out[32]:
{'m_commission': 2147483647,
 'm_commissionCurrency': None,
 'm_equityWithLoan': '1.7976931348623157E308',
 'm_initMargin': '1.7976931348623157E308',
 'm_maintMargin': '1.7976931348623157E308',
 'm_maxCommission': 2147483647,
 'm_minCommission': 2147483647,
 'm_status': 'Filled',
 'm_warningText': None}
```

Execution details:

```
exec_tmp[0].execution.__dict__
Out[36]:
{'m_acctNumber': 'DU274390',
 'm_avgPrice': 1895.25,
 'm_clientId': 302,
 'm_cumQty': 1,
 'm_evMultiplier': 0,
 'm_evRule': None,
 'm_exchange': 'GLOBEX',
 'm_execId': '0001f4e5.5698510d.01.01',
 'm_liquidation': 0,
 'm_orderId': 307,
 'm_orderRef': None,
 'm_permId': 719831681,
 'm_price': 1895.25,
 'm_shares': 1,
 'm_side': 'BOT',
 'm_time': '20160115  16:50:52'}
```

4.2.4 Placing an Option Order

```
"""
/IbPy/ibpy scripts/ibpy_Order.py"""
from ib.ext.Contract import Contract
from ib.ext.Order import Order
from ib.opt import Connection, message
import time

def error_handler(msg):
    print "Error : %s" % msg

def reply_handler(msg):
    print "Server Response: %s, %s" % (msg.typeName, msg)

def listen_orderstatus(msg):
    global orderstatus_tmp
    orderstatus_tmp.append(msg)

def listen_openorder(msg):
    global openorder_tmp
    openorder_tmp.append(msg)

def comm_report(msg):
    global comm_tmp
    comm_tmp.append(msg)

def exec_info(msg):
    global exec_tmp
    exec_tmp.append(msg)


def create_contract(symbol, secType, exchange, currency, right, strike, expiry):
    contract = Contract()
    contract.m_symbol = symbol
    contract.m_secType = secType
    contract.m_exchange = exchange
    contract.m_currency = currency
    contract.m_right = right
    contract.m_strike = strike
    contract.m_expiry = expiry
    return contract

def create_order(account, orderType, totalQuantity, action):
    order = Order()
    order.m_account = account
    order.m_orderType = orderType
    order.m_totalQuantity = totalQuantity
    order.m_action = action
    return order

if __name__ == "__main__":
    orderstatus_tmp = []
    openorder_tmp = []
    comm_tmp = []
    exec_tmp = []
    tws_conn = Connection.create(port=4001, clientId=450)
    tws_conn.connect()
    tws_conn.register(listen_orderstatus, message.orderStatus)
    tws_conn.register(listen_openorder, message.openOrder)
    tws_conn.register(comm_report, message.commissionReport)
    tws_conn.register(exec_info, message.execDetails)
    tws_conn.register(error_handler, 'Error')
    tws_conn.registerAll(reply_handler)

    order_id = 402
    contract_info = create_contract('NFLX', 'OPT', 'SMART',
                                    'USD', 'C', '110', '201603')
    order_info = create_order("DU274390", 'MKT', 1, 'BUY')
    tws_conn.placeOrder(order_id, contract_info, order_info)

    time.sleep(5)
    tws_conn.disconnect()
```

```
Error : <error id=402, errorCode=399, errorMsg=Order Message:
BUY 1 NFLX MAR 18 '16 110 Call
Warning: your order will not be placed at the exchange
until 2016-01-15 09:30:00 US/Eastern>
Server Response: error, <error id=402, errorCode=399, errorMsg=Order Message:
BUY 1 NFLX MAR 18 '16 110 Call
Warning: your order will not be placed at the exchange until
2016-01-15 09:30:00 US/Eastern>
Server Response: openOrder, <openOrder orderId=402,
contract=<ib.ext.Contract.Contract object at 0x000000000B478978>,
order=<ib.ext.Order.Order object at 0x000000001A0B8860>,
orderState=<ib.ext.OrderState.OrderState object at 0x000000001A026B00>>
Server Response: orderStatus, <orderStatus orderId=402, status=PreSubmitted,
filled=0, remaining=1, avgFillPrice=0.0, permId=719831698, parentId=0,
lastFillPrice=0.0, clientId=450, whyHeld=None>
Server Response: openOrder, <openOrder orderId=402,
contract=<ib.ext.Contract.Contract object at 0x000000001A026A58>,
order=<ib.ext.Order.Order object at 0x000000000B4787B8>,
orderState=<ib.ext.OrderState.OrderState object at 0x000000001A0265F8>>
Server Response: orderStatus, <orderStatus orderId=402,
status=Submitted, filled=0, remaining=1, avgFillPrice=0.0, permId=719831698,
parentId=0, lastFillPrice=0.0, clientId=450, whyHeld=None>
```

4.2.5 Placing a Bond Order

```
"""
/IbPy/ibpy scripts/ibpy_Order_CORP.py"""
from ib.ext.Contract import Contract
from ib.ext.Order import Order
from ib.opt import Connection, message
import time

def error_handler(msg):
    print "Error : %s" % msg

def reply_handler(msg):
    print "Server Response: %s, %s" % (msg.typeName, msg)

def listen_orderstatus(msg):
    global orderstatus_tmp
    orderstatus_tmp.append(msg)

def listen_openorder(msg):
    global openorder_tmp
    openorder_tmp.append(msg)

def comm_report(msg):
    global comm_tmp
    comm_tmp.append(msg)

def exec_info(msg):
    global exec_tmp
    exec_tmp.append(msg)

def create_contract(symbol,
                    secType,
                    exchange,
                    currency,
                    bondType):
    contract = Contract()
    contract.m_symbol = symbol
    contract.m_secType = secType
    contract.m_exchange = exchange
    contract.m_currency = currency
    contract.m_bondType = bondType
    return contract

def create_order(account, orderType, totalQuantity, action):
    order = Order()
    order.m_account = account
    order.m_orderType = orderType
    order.m_totalQuantity = totalQuantity
    order.m_action = action
    return order

if __name__ == "__main__":
    orderstatus_tmp = []
    openorder_tmp = []
    comm_tmp = []
    exec_tmp = []
    tws_conn = Connection.create(port=4001, clientId=501)
    tws_conn.connect()
    tws_conn.register(listen_orderstatus, message.orderStatus)
    tws_conn.register(listen_openorder, message.openOrder)
    tws_conn.register(comm_report, message.commissionReport)
    tws_conn.register(exec_info, message.execDetails)
    tws_conn.register(error_handler, 'Error')
    tws_conn.registerAll(reply_handler)

    order_id = 522

    contract_info = create_contract('IBCID143913442',
                                    'BOND',
                                    'SMART',
                                    'USD',
                                    'CORP'
                                    )
    order_info = create_order("DU274383", 'MKT', 100, 'BUY')
    tws_conn.placeOrder(order_id, contract_info, order_info)
    time.sleep(10)

    tws_conn.disconnect()
```

Bond Price is 104.079, which is 104.079%. The Par Value of Bond is $1000. Hence the value per bond is $1040.79. When we place order to purchase bond, we are putting number of bonds we are buying. In this case, we are buying 100 bonds. From the orderStatus report, at a filled price of 104.079 per bond, the 100 bonds total value is $104,079.

```
print(orderstatus_tmp[-1])
<orderStatus orderId=522,
status=Filled,
filled=100,
remaining=0,
avgFillPrice=104.079,
permId=530118450,
parentId=0,
lastFillPrice=104.079,
clientId=501,
whyHeld=None>
```

Chapter 5

Account and Portfolio

The IB API online reference is always updated. Hence it is best to consult it regularly for changes. Interactive Brokers API[1]. We will be using the Java portion as our points of reference as IbPy was built from the Java API. I will also be using the IbPy optional interface as well. I find it more intuitive for a non software engineer like me. Interactive Brokers (IB) provides Java EClientSocket Methods for extracting data. The Java EWrapper Methods returns or receives the information to Python.

5.1 Introduction

In order to extract data from Interactive Brokers via IbPy, we need to familiarise ourselves with IB's protocol. IB uses what is called, in computer science, a **callback method**[2]. The idea of a callback is very similar to how things work in real life. You are performing a task but needed information from another co-worker. You, **the caller or information requestor**, made a call to your co-worker for information. Your co-worker explained that he is in the middle of another task but will get back to you (**callback**) when he had completed the other task and had gathered the information we need. Whilst waiting for him, you proceeded with another part of you task. When he eventually **called back**, you proceed to complete the parts that required the supplied information.

In real life, we often perform many parts of a tasks in parallel and asynchronously. However, computer does not know how to do that unless we provide such instructions. Without such instructions, everything will be on halt until the information arrives, which of course is silly and inefficient. Hence, we utilised **callback function or method** to perform parallel tasks while the information is on its way. Similarly, the idea here is that we do need to wait for the information to arrive. This may seemed trivial but often programs terminate before information arrived and we erroneously concluded that there's a bug in the program. In reality, the information was already on the way but we were not patient enough to see it through.

Another concept that we need to grasp is that we need to tell the computer where to pass the information to. In real life, we would provide the co-worker a number or contact details to return the necessary information. With computers, we need to provide such instructions as well. This in computer terms is called the **listener or callback method**.

To provide an example using IB built in methods, we make an account information request via the EClientSocket Method (**information requestor**) called `reqAccountUpdates()` and informed the computer to return the information to the EWrapper method (**callback method**) called `updateAccountValue()`. An example of this is provided in section 5.2.

Under Account and Portfolio, there are 5 request methods available. The five methods are `reqAccountUpdates()`, `reqAccountSummary()`, `cancelAccountSummary()`, `reqPositions()` and `cancelPositions()`. The table below links the EClientSocket request Methods with the corresponding EWrapper callback Methods.

[1] `https://www.interactivebrokers.com/en/software/api/api.htm`

[2] `https://en.wikipedia.org/wiki/Callback_(computer_programming)`

EClientSocket Methods	**EWrapper Methods**

reqAcccountUpdates()	updateAccountValue()
	updatePortfolio()
	updateAccountTime()
	accountDownloadEnd()

reqAccountSummary()	accountSummary()
	accountSummaryEnd()

cancelAccountSummary()	

reqPositions()	position()
	positionEnd()

cancelPositions()	

From the summary table, we can see that make a call via `reqAccountUpdates()`, IB will return the information via `updateAccountValue()`, `updatePortfolio()` and `updateAccountTime()`. There is no corresponding EWrapper method for the `cancelAccountSummary()` and `cancelPositions()`.

5.2 Obtaining Account Information

The IB API provides a rich source of information. Once we have made the request for information via the `reqAccountUpdates()` method, the information returned can be collected via four listener methods (also called Java EWrapper Methods). These are `updateAccountValue()`, `updatePortfolio()`, `updateAccountTime()`, and `accountDownloadEnd()`. We shall take some time to walk through each of these.

to initiate the process, we need to make a call to the `reqAccountUpdates()` method. The detail description can be found in section **??**. Let's use an example to illustrate this:

```
conn.reqAccountUpdates(1,ACCT_NO)
```

Ignoring the prefix `conn` for now, the example code shows that we need to provide two arguments. The first argument is `"subscribe"` which is a boolean input and the second is `"acctCode"` which is a string. In the above example, we use **"1"** which set the condition to **"TRUE"** to indicate that we would like to start receiving account and portfolio updates. The `ACCT_NO` is a variable which should be an IB account number. In the example to follow in section 5.2, we use `DU274390` as the input value. You would replace the value with your own IB account code.

Example - reqAccountUpdates() & updateAccountValue()

```
"""
/IbPy/ibpy scripts/ibpy_Account_Values.py"""
from ib.opt import Connection, message
import time

def updateAccount_handler(msg):
    global tmp
    global tmp2
    if msg.key in ['AccountCode', 'NetLiquidation']:
        tmp.append(msg.value)
    tmp2.append(msg)

if __name__ == "__main__":
    tmp = []
    tmp2 = []
    ACCT_NO = "DU274390"
    conn = Connection.create(port=4001, clientId = 100)
    conn.connect()
    conn.register(updateAccount_handler, message.updateAccountValue)
    conn.reqAccountUpdates(1,ACCT_NO)
    time.sleep(5)
    conn.disconnect()

    print(tmp)
```

The following is the output:

```
Server Version: 76
TWS Time at connection:20160114 15:56:21 SGT
['DU274390', '1294338.76']
```

Prior to calling the `reqAccountUpdates()` method, we need to have ran the following code:

```
conn.register(updateAccount_handler, message.updateAccountValue)
```

The purpose of this line is to inform IB that, when we request for account information via the `reqAccountUpdates` method, all the returned data with the **updateAccountValue** attributes are to be collected via the self-defined `updateAccount_handler` method. In this case, we are only gathering messages with the **updateAccountValue** attributes because we inform IbPy by specifying with `message.updateAccountValue` that we are only interested in messages with **updateAccountValue** attributes.

Aside from information gathered by `updateAccountValue()`, as mentioned earlier, IB also return other information. These are `updatePortfolio()`, `updateAccountTime()`, and `accountDownloadEnd()`. We will go into Portfolio information in section 5.3.

The reason that we need to ran the `register()` method is that we need the listener function to start "listening" per se, prior to us calling the `reqAccountUpdates()`, is so that any returned messages from IB is captured.

In the `updateAccount_handler` function that we defined, we listened for all information. However, we chose to record only information on **Account Code** and **Net Liquitation Value** and append these value in a global variable called `tmp`. In fact, in the example, IB returned 308 fields of information as can be seen in the data attached in section **??**.

```
def updateAccount_handler(msg):
    global tmp
    global tmp2
    if msg.key in ['AccountCode', 'NetLiquidation']:
        tmp.append(msg.value)
```

We also used another variable `tmp2` that listened and collected all the information that IB sent to us, which is how we obtained all in the information in section **??**.

```
tmp2.append(msg)
```

Notice below that `tmp2` is an object that collected all the message returned from IB after we called the `reqAccountUpdate()` method. I have printed them some of them here for reference. Line breaks were added for ease of reading. Refer to section **??** for the complete list.

```
print(tmp2[0])
<updateAccountValue key=AccountType, value=INDIVIDUAL, currency=None, accountName=DU274390>

len(tmp2)
Out[4]: 308

for oo in range(0, len(tmp2)):
    print(tmp2[oo])
...:
<updateAccountValue key=AccountType, value=INDIVIDUAL, currency=None, accountName=DU274390>
<updateAccountValue key=AccountCode, value=DU274390, currency=None, accountName=DU274390>
...
<updateAccountValue key=RealCurrency, value=USD, currency=USD, accountName=DU274390>
<updateAccountValue key=IssuerOptionValue, value=0, currency=USD, accountName=DU274390>
```

For the sake of completeness, below are three tables with the list of values returned by IB.

AccountType	DayTradesRemainingT+1	LookAheadNextChange
AccountCode	DayTradesRemainingT+2	SegmentTitle-C
AccountReady	DayTradesRemainingT+3	SegmentTitle-S
Cushion	DayTradesRemainingT+4	TradingType-S
DayTradesRemaining	Leverage-S	WhatIfPMEnabled

Table 5.1: Account Information (15 fields)

AccruedCash	FullInitMarginReq-C	MaintMarginReq-C
AccruedCash-C	FullInitMarginReq-S	MaintMarginReq-S
AccruedCash-S	FullMaintMarginReq	NetLiquidation
AccruedDividend	FullMaintMarginReq-C	NetLiquidation-C
AccruedDividend-C	FullMaintMarginReq-S	NetLiquidation-S
AccruedDividend-S	GrossPositionValue	PASharesValue
AvailableFunds	GrossPositionValue-S	PASharesValue-C
AvailableFunds-C	IndianStockHaircut	PASharesValue-S
AvailableFunds-S	IndianStockHaircut-C	PostExpirationExcess
Billable	IndianStockHaircut-S	PostExpirationExcess-C
Billable-C	InitMarginReq	PostExpirationExcess-S
Billable-S	InitMarginReq-C	PostExpirationMargin
BuyingPower	InitMarginReq-S	PostExpirationMargin-C
EquityWithLoanValue	LookAheadAvailableFunds	PostExpirationMargin-S
EquityWithLoanValue-C	LookAheadAvailableFunds-C	PreviousDayEquityWithLoanValue
EquityWithLoanValue-S	LookAheadAvailableFunds-S	PreviousDayEquityWithLoanValue-S
ExcessLiquidity	LookAheadExcessLiquidity	RegTEquity
ExcessLiquidity-C	LookAheadExcessLiquidity-C	RegTEquity-S
ExcessLiquidity-S	LookAheadExcessLiquidity-S	RegTMargin
FullAvailableFunds	LookAheadInitMarginReq	RegTMargin-S
FullAvailableFunds-C	LookAheadInitMarginReq-C	SMA
FullAvailableFunds-S	LookAheadInitMarginReq-S	SMA-S
FullExcessLiquidity	LookAheadMaintMarginReq	TotalCashValue
FullExcessLiquidity-C	LookAheadMaintMarginReq-C	TotalCashValue-C
FullExcessLiquidity-S	LookAheadMaintMarginReq-S	TotalCashValue-S
FullInitMarginReq	MaintMarginReq	

Table 5.2: In USD only (77 fields)

Currency	NetLiquidationByCurrency	CorporateBondValue
CashBalance	UnrealizedPnL	TBondValue
TotalCashBalance	RealizedPnL	TBillValue
AccruedCash	ExchangeRate	WarrantValue
StockMarketValue	FundValue	FxCashBalance
OptionMarketValue	NetDividend	AccountOrGroup
FutureOptionValue	MutualFundValue	RealCurrency
FuturesPNL	MoneyMarketFundValue	IssuerOptionValue

Table 5.3: In Multicurrency. The values are duplicated for each currency (24 fields)

5.3 Obtaining Portfolio Information

As mentioned in section 5.2, there are other information that was returned by IB. However, in the previous section, we specify in the program to listen only for Account Information. The example to follow illustrates how we can program to listen for Portfolio information.

Below is a reproduction of the table in section **??** for ease of reference to understand the information being returned by IB.

Parameter	Type	Description
contract	Contract	This structure contains a description of the contract which is being traded. The exchange field in a contract is not set for portfolio update.
position	int	This integer indicates the position on the contract. If the position is 0, it means the position has just cleared.
marketPrice	double	Unit price of the instrument.
marketValue	double	The total market value of the instrument.
averageCost	double	The average cost per share is calculated by dividing your cost (execution price + commission) by the quantity of your position.
unrealizedPNL	double	The difference between the current market value of your open positions and the average cost, or Value - Average Cost.
realizedPNL	double	Shows your profit on closed positions, which is the difference between your entry execution cost (execution price + commissions to open the position) and exit execution cost (execution price + commissions to close the position)
accountName	String	The name of the account to which the message applies. Useful for Financial Advisor subaccount messages.

```
"""
/IbPy/ibpy scripts/ibpy_Portfolio_Info.py"""
from ib.opt import Connection, message
import time

def updateAccount_handler(msg):
    global tmp
    tmp.append(msg)

if __name__ == "__main__":
    tmp = []
    ACCT_NO = "DU274390"
    conn = Connection.create(port=4001, clientId = 100)
    conn.connect()

    conn.register(updateAccount_handler, message.updatePortfolio)
    conn.reqAccountUpdates(1,ACCT_NO)
    time.sleep(5)

    conn.disconnect()
    print(tmp)
```

I added line breaks for readability. The output:

```
TWS Time at connection:20160114 16:02:15 SGT
[<ib.opt.message.UpdatePortfolio object at 0x000000001A13ECE0>,
<ib.opt.message.UpdatePortfolio object at 0x000000001A13ED48>,
<ib.opt.message.UpdatePortfolio object at 0x000000001A13EDB0>,
<ib.opt.message.UpdatePortfolio object at 0x000000001A13EE18>]
```

Note that there are 4 objects stored in tmp.

```
In [16]: tmp
Out[16]:
[<ib.opt.message.UpdatePortfolio at 0x1a13ece0>,
 <ib.opt.message.UpdatePortfolio at 0x1a13ed48>,
 <ib.opt.message.UpdatePortfolio at 0x1a13edb0>,
 <ib.opt.message.UpdatePortfolio at 0x1a13ee18>]
```

When we print the first object as shown below, the portfolio details are revealed. However, the first message is a contract object which we need to unpack again for more info.

```
print(tmp[0])
<updatePortfolio contract=<ib.ext.Contract.Contract object at 0x000000001A351EF0>,
position=-11000,
marketPrice=24.6000004,
marketValue=-270600.0,
averageCost=25.16725885,
unrealizedPNL=6239.84,
realizedPNL=0.0,
accountName=DU274390>
```

As can be seen below, you can extract the contract details by using the __dict__ method.

```
In [18]:tmp[0].contract.__dict__
Out[18]:
{'m_conId': 268084,
 'm_currency': 'USD',
 'm_expiry': None,
 'm_includeExpired': False,
 'm_localSymbol': 'CSCO',
 'm_multiplier': None,
 'm_primaryExch': 'NASDAQ',
 'm_right': '0',
 'm_secType': 'STK',
 'm_strike': 0.0,
 'm_symbol': 'CSCO',
 'm_tradingClass': 'NMS'}
```

```
In [19]: tmp[0].contract.m_symbol
Out[19]: 'CSCO'
```

5.4 Obtaining Account Update Time

To obtain information on when we last had an account update and receiving a notification of the end of account download are quite straight forward. The following examples, will serve to illustrate it can done.

For `updateAccountTime()`, the information returned is the timestamp of last few updates. For `accountDownloadEnd()`, the information returned is the `accountName`.

Example - updateAccountTime()

```
"""
/IbPy/ibpy scripts/ibpy_Account_Time.py"""
from ib.opt import Connection, message
import time
def updateAccount_handler(msg):
    global tmp
    tmp.append(msg)

if __name__ == "__main__":
    tmp = []
    ACCT_NO = "DU274390"

    conn = Connection.create(port=4001, clientId = 101)
    conn.connect()
    conn.register(updateAccount_handler, message.updateAccountTime)

    conn.reqAccountUpdates(1,ACCT_NO)
    time.sleep(1)

    conn.disconnect()
    for oo in range(0, len(tmp)):
        print(tmp[oo])
```

Output

```
TWS Time at connection:20160114 16:45:23 SGT
<updateAccountTime timeStamp=15:55>
<updateAccountTime timeStamp=15:55>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=16:43>
<updateAccountTime timeStamp=15:45>
<updateAccountTime timeStamp=15:55>
<updateAccountTime timeStamp=15:55>
<updateAccountTime timeStamp=15:55>
<updateAccountTime timeStamp=15:55>
```

Example - accountDownloadEnd()

```
"""
/IbPy/ibpy scripts/ibpy_account_Download_End.py"""
from ib.opt import Connection, message
import time

def dl_end(msg):
    global res
    res=msg

if __name__ == "__main__":
    res = []
    ACCT_NO = "DU274390"
    conn = Connection.create(port=4001, clientId = 100)
    conn.connect()
    conn.register(dl_end, message.accountDownloadEnd)
    conn.reqAccountUpdates(1,ACCT_NO)
    time.sleep(1)
    conn.disconnect()
    print("Result " + str(res.accountName))
```

The name of the account is returned. The attribute is `accountName`. Below is the Output:

```
runfile('G:/IbPy/ibpy_accountDownloadEnd.py', wdir='G:/IbPy')
TWS Time at connection:20160114 16:53:59 SGT
Result DU274390
```

5.5 Obtaining Account Summary

According to the IB API guide, we can utilise the `reqAccountSummary()` method to gather the data that appears on the TWS Account Window Summary tab. Unfortunately, the IB guide is not very clear what information can be extracted for Account Summary when one submit the request via the `reqAccountSummary()` method. For now we demonstrate how one can make a request call and we will come back to what tag we can used to obtain other information.

Example - accountSummary()

```
# -*- coding: utf-8 -*-
"""
/IbPy/ibpy scripts/ibpy_Account_Summary.py"""
from ib.opt import Connection, message
import time

def update_acct_summ(msg):
    global tmp
    tmp.append(msg)

if __name__ == "__main__":
    reqID = 1001
    dat_list = ["AccountType","NetLiquidation","TotalCashValue"]
    tmp = []

    conn = Connection.create(port=4001, clientId = 100)
    conn.connect()

    conn.register(update_acct_summ, message.accountSummary)
    for oo in dat_list:
        conn.reqAccountSummary(reqID,"All",oo)
        time.sleep(1)
        #print(tmp)
    conn.disconnect()
```

Note that the length of `tmp` is quite long, 363 in this case. The reason for that is that there're a lot of sub accounts, 121 to be precise under this instructor's account. Because we requested for 3 tags - **"AccountType", "NetLiquidation", "TotalCashValue"** - hence we received 363 messages.

```
len(tmp)
Out[55]: 363

print(tmp[100])
<accountSummary reqId=1001,
account=DU262745,
tag=AccountType,
value=INDIVIDUAL,
currency=None>

print(tmp[200])
<accountSummary reqId=1001,
account=DU255649,
tag=NetLiquidation,
value=646393.86,
currency=USD>
```

```
print(tmp[300])
<accountSummary reqId=1001,
account=DU254946,
tag=TotalCashValue,
value=795538.70,
currency=USD>
```

A natural question to ask is what tags can be used as input? With trial and error, I have been able to identify the following fields which can be used as tags for the **reqAccountSummary(int reqId, String group, String tags)** submission. The tables are very similar to the one in section 5.2 except for the multicurrency table.

AccountType	DayTradesRemainingT+1	LookAheadNextChange
AccountCode	DayTradesRemainingT+2	SegmentTitle-C
AccountReady	DayTradesRemainingT+3	SegmentTitle-S
Cushion	DayTradesRemainingT+4	TradingType-S
DayTradesRemaining	Leverage-S	WhatIfPMEnabled

Table 5.4: Account Information (15 fields)

AccruedCash	FullInitMarginReq-C	MaintMarginReq-C
AccruedCash-C	FullInitMarginReq-S	MaintMarginReq-S
AccruedCash-S	FullMaintMarginReq	NetLiquidation
AccruedDividend	FullMaintMarginReq-C	NetLiquidation-C
AccruedDividend-C	FullMaintMarginReq-S	NetLiquidation-S
AccruedDividend-S	GrossPositionValue	PASharesValue
AvailableFunds	GrossPositionValue-S	PASharesValue-C
AvailableFunds-C	IndianStockHaircut	PASharesValue-S
AvailableFunds-S	IndianStockHaircut-C	PostExpirationExcess
Billable	IndianStockHaircut-S	PostExpirationExcess-C
Billable-C	InitMarginReq	PostExpirationExcess-S
Billable-S	InitMarginReq-C	PostExpirationMargin
BuyingPower	InitMarginReq-S	PostExpirationMargin-C
EquityWithLoanValue	LookAheadAvailableFunds	PostExpirationMargin-S
EquityWithLoanValue-C	LookAheadAvailableFunds-C	PreviousDayEquityWithLoanValue
EquityWithLoanValue-S	LookAheadAvailableFunds-S	PreviousDayEquityWithLoanValue-S
ExcessLiquidity	LookAheadExcessLiquidity	RegTEquity
ExcessLiquidity-C	LookAheadExcessLiquidity-C	RegTEquity-S
ExcessLiquidity-S	LookAheadExcessLiquidity-S	RegTMargin
FullAvailableFunds	LookAheadInitMarginReq	RegTMargin-S
FullAvailableFunds-C	LookAheadInitMarginReq-C	SMA
FullAvailableFunds-S	LookAheadInitMarginReq-S	SMA-S
FullExcessLiquidity	LookAheadMaintMarginReq	TotalCashValue
FullExcessLiquidity-C	LookAheadMaintMarginReq-C	TotalCashValue-C
FullExcessLiquidity-S	LookAheadMaintMarginReq-S	TotalCashValue-S
FullInitMarginReq	MaintMarginReq	

Table 5.5: In USD only (77 fields)

5.6 Obtaining Position Information

In order to extract position information, we need to utilise the `reqPosition()` method. For ease of reference, a section of the IB Java manual on the listener method `positon()` is reproduced here. For complete reference, please refer to section **??**. This event returns real-time positions for all accounts in response to the `reqPositions()` method.

void position(String account, Contract contract, int pos)

Parameter	Type	Description
account	String	The account.
contract	Contract	This structure contains a full description of the contract that was executed.
pos	double	The position.

```
"""
/IbPy/ibpy scripts/ibpy_Positions.py"""
from ib.opt import Connection, message
import time

def update_position(msg):
    global tmp
    tmp.append(msg)


if __name__ == "__main__":
    tmp = []
    conn = Connection.create(port=4001, clientId = 111)
    conn.connect()

    conn.register(update_position, message.position)
    conn.reqPositions()

    time.sleep(1)
    conn.disconnect()
```

From the output below, we can clearly see 4 fields of information per position have been return. As expected, we see account, contract and position size information. Although not recorded as one of the information, average cost was also provided.

```
len(tmp)
Out[90]: 194

print(tmp[0])
<position account=DU255426,
contract=<ib.ext.Contract.Contract object at 0x000000001A25DA90>,
pos=100,
avgCost=20.89>
```

The contract information can be extracted as below:

```
In [95]: tmp[0].contract.__dict__
Out[95]:
{'m_conId': 137935324,
 'm_currency': 'USD',
 'm_exchange': 'ARCA',
 'm_expiry': None,
 'm_includeExpired': False,
 'm_localSymbol': 'VXX',
 'm_multiplier': None,
'm_right': None,
'm_secType': 'STK',
'm_strike': 0.0,
'm_symbol': 'VXX',
'm_tradingClass': 'VXX'}
```

Part II

Sample Codes

Chapter 6

Account and Portfolio

ib_class

```
import time
from datetime import datetime
from IBWrapper import IBWrapper, contract
from ib.ext.EClientSocket import EClientSocket
from ib.ext.ScannerSubscription import ScannerSubscription


if __name__=="__main__":
    callback = IBWrapper()          # Instantiate IBWrapper
    tws = EClientSocket(callback)   # Instantiate EClientSocket
    host = ""
    port = 4001
    clientId = 5000
    tws.eConnect(host, port, clientId)     # Connect to TWS
    tws.setServerLogLevel(5)
    accountName = "DU254946"
    create = contract()             # Instantiate contract class

    # Initiate attributes to receive data. At some point we need a separate class for this
    callback.initiate_variables()

    # Account and Portfolio ##############################################################
    # reqAccountUpdates   --->   updateAccountTime      self.update_AccountTime
    #                            updateAccountValue     self.update_AccountValue
    #                            updatePortfolio        self.update_Portfolio
    # accountDownloadEnd                                self.accountDownloadEnd_flag
    # reqAccountSummary   --->   accountSummary         self.account_Summary
    # cancelAccountSummary
    # accountSummaryEnd                                 self.account_SummaryEnd_flag
    # reqPositions        --->   position               self.update_Position
    # cancelPositions
    # positionEnd                                       self.positionEnd_flag
    ##################################################################################'''
    '''print "Testing Account and Portfolio \n"
    tws.reqAccountUpdates(1, accountName)
    tws.reqAccountSummary(1,"All","NetLiquidation")
    #tws.cancelAccountSummary(1)
    tws.reqPositions()
    #tws.cancelPositions()
```

```
'''

# Orders #############################################################################
# placeOrder            --->   orderStatus**         self.order_Status
# cancelorder
#                       --->   openOrderEnd          self.open_OrderEnd_flag
# reqOpenOrders         --->   openOrder*            self.open_Order
#                       --->   orderStatus**
# reqAllOpenOrders      --->   openOrder*
#                       --->   orderStatus**
# reqAutoOpenOrders     --->   openOrder*
#                       --->   orderStatus**
# reqIds                --->   nextValidId           self.next_ValidId
#                       --->   deltaNeutralValidation
# exerciseOptions
# reqGlobalCancel
##################################################################################'''
'''print "Testing Orders Group \n"
# Example 1 - placing order to buy stock
tws.reqIds(1)   # Need to request next valid order Id
time.sleep(2)   # wait for response from server
order_id = callback.next_ValidId
contract_info1 = create.create_contract('GOOG', 'STK', 'SMART', 'USD')
order_info1 = create.create_order(accountName, 'MKT', 100, 'BUY')
tws.placeOrder(order_id, contract_info1, order_info1)

# Example 2 - placing order to buy FX
tws.reqIds(1)
time.sleep(1)
order_id = callback.next_ValidId
contract_info2 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
order_info2 = create.create_order(accountName, 'MKT', 100000, 'BUY')
tws.placeOrder(order_id, contract_info2, order_info2)

#tws.cancelOrder(order_id)   # Cancel example 2 order
#tws.reqOpenOrders()
#tws.reqAllOpenOrders()
#tws.reqAutoOpenOrders(1)    # clientId had to be 0 for this to work
tws.reqGlobalCancel()'''
```

```
# Market Data #########################################################################
# reqMktData           --->   tickPrice             self.tick_Price
#                      --->   tickSize              self.tick_Size
#                      --->   tickOptionComputation self.tick_OptionComputation
#                      --->   tickGeneric           self.tick_Generic
#                      --->   tickString            self.tick_String
#                      --->   tickEFP               self.tick_EFP
#                      --->   tickSnapshotEnd       self.tickSnapshotEnd_flag
# cancelMktData
# calculateImpliedVolatility >tickOptionComputation self.tick_OptionComputation
# cancelcalculateImpliedVolatility
# calculateOptionPrice --->   tickOptionComputation self.tick_OptionComputation
# cancelCalculateOptionPrice
# reqMktDataType       --->   marketDataType        self.market_DataType
#####################################################################################'''
'''print "Testing Market Data Group \n"
contract_info3 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
tws.reqMktData(1, contract_info3, "", False)

contract_info4 = create.create_contract('NFLX 160318C00100000',
                                        'OPT', 'SMART', 'USD',
                                        'CALL', '100', '20160318',
                                        100, "NFLX")
tws.calculateImpliedVolatility(2, contract_info4, 3.60, 94.41)
tws.calculateOptionPrice(3, contract_info4, 0.42, 94.41)
tws.reqMarketDataType(2)                            # need to test this when mkt opens
time.sleep(2)
tws.cancelMktData(1)
tws.cancelCalculateImpliedVolatility(2)
tws.cancelCalculateOptionPrice(3)'''
```

```
# Connection and Server ##############################################################
# EClientSocket
# eConnect
# eDisconnect           --->   connectionClosed
# isConnected
# setServerLogLevel
# reqCurrentTime        --->   currentTime           self.current_Time
# serverVersion
# TwsConnectionTime
#                       --->   error
####################################################################################'''
'''print "Testing Connection and Server Group \n"
print  tws.isConnected()
tws.setServerLogLevel(5)
tws.reqCurrentTime()
print "Server Version " + str(tws.serverVersion())
print "TWS Connection Time %s " % tws.TwsConnectionTime()'''




# Executions #######################################################################
# reqExecutions         --->   execDetails           self.exec_Details_reqId
#                                                    self.exec_Details_contract
#                                                    self.exec_Details_execution
#                       --->   execDetailsEnd        self.exec_DetailsEnd_flag
#                       --->   commissionReport      self.commission_Report
####################################################################################'''
'''print "Testing Executions Group \n"
order_id = []
tws.reqIds(1)
while not order_id:
    time.sleep(0.1)
    order_id = callback.next_ValidId
    print "waiting for id"
order_id = callback.next_ValidId
print ("Just got it. The next order id is: ", order_id)
contract_info5 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
order_info5 = create.create_order(accountName, 'MKT', 100000, 'BUY')
tws.placeOrder(order_id, contract_info5, order_info5)
time.sleep(2)
tws.reqExecutions(0, create.exec_filter(9999, accountName, contract_info5))'''
```

```
# Contract ############################################################################
# reqContractDetails    --->   contractDetails       self.contract_Details_reqId
#                                                     self.contract_Details
#                       --->   contractDetailsEnd    self.contract_DetailsEnd_reqId
#                                                     self.contract_Details_flag
#                       --->   bondContractDetails   self.bond_ContractDetails_reqId
#                                                     self.bond_ContractDetails
#####################################################################################'''
'''print "Testing Contract Group \n"
# Example 1 - Option
contract_Details6 = create.create_contract('NFLX 160318C00100000', 'OPT', 'SMART',
                                           'USD', 'CALL', '100', '20160318',
                                           100, "NFLX")
tws.reqContractDetails(5000, contract_Details6)
while not callback.contract_Details_flag:
    time.sleep(1)
callback.contract_Details_flag = False
print callback.contract_Details_reqId
print callback.contract_Details.__dict__

# Example 2 - Stock
contract_Details7 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
tws.reqContractDetails(5001, contract_Details7)
while not callback.contract_Details_flag:
    time.sleep(1)
callback.contract_Details_flag = False
print callback.contract_Details_reqId
print callback.contract_Details.__dict__

# Example 3 - FX
contract_Details8 = create.create_contract('IBCID143913442',
                                            'BOND', 'SMART',
                                            'USD','CORP')
tws.reqContractDetails(5002, contract_Details8)
while not callback.contract_Details_flag:
    time.sleep(1)
callback.contract_Details_flag = False
print callback.bond_ContractDetails_reqId
print callback.bond_ContractDetails.__dict__ '''
```

```
# Market Depth ######################################################################
# reqMktDepth          --->   updateMktDepth        self.update_MktDepth
#                      --->   update_MktDepthL2     self.update_MktDepthL2
##################################################################################'''
'''print "Testing Market Depth Group \n"
contract_info9 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
tws.reqMktDepth(7000, contract_info9, 3)
time.sleep(5)
print callback.update_MktDepth
tws.cancelMktDepth(7000)'''




# News Bulletin #####################################################################
# reqNewsBulletins     --->   updateNewsBulletin    self.update_NewsBulletin_msgId
#                                                   self.update_NewsBulletin_msgType
#                                                   self.update_NewsBulletin_message
#                                                   self.update_NewsBulletin_origExchange
##################################################################################'''
'''print "Testing News Bulletin Group \n"
tws.reqNewsBulletins(1)
time.sleep(20)
tws.cancelNewsBulletins()'''




# Financial Advisors Group ##########################################################
# reqManagedAccts      --->   managedAccounts       self.managed_Accounts
##################################################################################'''
'''print "Testing Financial Advisors Group \n"
tws.reqManagedAccts()
#tws.requestFA()       # non FA account. Unable to test.
'''




# Historical Data ###################################################################
# reqHistoricalData    --->   historicalData        self.historical_Data
##################################################################################'''
'''print "Testing Historical Data Group \n"
contract_Details10 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
data_endtime = datetime.now().strftime("%Y%m%d %H:%M:%S")
tws.reqHistoricalData(9000, contract_Details10, data_endtime,
                      "1 M", "1 day", "BID", 0, 1)
time.sleep(3)
tws.cancelHistoricalData(9000)'''
```

```
# Market Scanners ####################################################################
# reqScannerParameters --->   scannerParameters     self.scanner_Parameters
#                      --->   scannerData           self.scanner_Data
#                      --->   scannerDataEnd        self.scanner_Data_End_reqID
#                                                   self.scanner_Data_reqID
#####################################################################################'''
'''print "Testing Market Scanners Group \n"
subscript = ScannerSubscription()
subscript.numberOfRows(3)
subscript.locationCode('STK.NYSE')
tws.reqScannerSubscription(700, subscript)
tws.reqScannerParameters()
time.sleep(3)
tws.cancelScannerSubscription(700)   '''




# Real Time Bars #####################################################################
# reqRealTimeBars      --->   realtimeBar           self.real_timeBar
#####################################################################################'''
'''print "Testing Real Time Bars Group \n"
contract_Details11 = create.create_contract('EUR', 'CASH', 'IDEALPRO', 'USD')
tws.reqRealTimeBars(10000, contract_Details11, 5, "MIDPOINT", 0)
time.sleep(10)
tws.cancelRealTimeBars(10000)'''




# Fundamental Data ###################################################################
# reportType = ReportSnapshot
#              ReportsFinSummary
#              ReportsFinSummary
#              ReportRatios
#              ReportFinStatements
#              RESC
#              Calendar Report
# Unfortunately, no access. 430, 'We are sorry, but fundamentals data for the
# security specified is not available.Not allowed'
#####################################################################################'''
'''print "Testing Fundamental Data Group \n"
contract_info12 = create.create_contract('AAPL', 'STK', 'SMART', 'USD')
reportType = "ReportSnapshot"
tws.reqFundamentalData(10100, contract_info12, reportType)
time.sleep(10)
tws.cancelFundamentalData(10100)'''




# Disconnect from TWS
tws.isConnected()
tws.eDisconnect()
```

IBWrapper

```
'''
Wrapper - Organised by groups. E.g., Accont and Portfolio group, Orders group etc
2016-01-31
'''

from ib.ext.EWrapper import EWrapper
from ib.ext.Contract import Contract
from ib.ext.ExecutionFilter import ExecutionFilter
from ib.ext.Order import Order

class IBWrapper(EWrapper):
    def initiate_variables(self):
        # Account and Portfolio
        setattr(self, "accountDownloadEnd_flag", False)
        setattr(self, "update_AccountTime", None)
        setattr(self, "update_AccountValue", [])
        setattr(self, "update_Portfolio", [])
        setattr(self, 'account_Summary', [])
        setattr(self, 'account_SummaryEnd_flag', False)
        setattr(self, 'update_Position', [])
        setattr(self, 'positionEnd_flag', False)
        # Orders
        setattr(self, 'order_Status', [])
        setattr(self, 'open_Order', [])
        setattr(self, 'open_OrderEnd_flag', True)
        # Market Data
        setattr(self, 'tick_Price', [])
        setattr(self, 'tick_Size', [])
        setattr(self, 'tick_OptionComputation', [])
        setattr(self, 'tick_Generic', [])
        setattr(self, 'tick_String', [])
        setattr(self, 'tick_EFP', [])
        setattr(self, 'tickSnapshotEnd_reqId', [])
        setattr(self, 'tickSnapshotEnd_flag', False)
        # Connection and Server
        setattr(self, 'connection_Closed', False)
        # Executions
        setattr(self, "exec_Details_reqId", [])
        setattr(self, "exec_Details_contract", [])
        setattr(self, "exec_Details_execution", [])
        setattr(self, "exec_DetailsEnd_flag", False)
        # Contract
        setattr(self, "contract_Details_flag", False)
        # Market Depth
        setattr(self, 'update_MktDepth', [])
        setattr(self, 'update_MktDepthL2', [])
        # Historical Data
        setattr(self, 'historical_Data', [])
        setattr(self, 'historical_DataEnd_flag', False)
        # Market Scanners
        setattr(self, 'scanner_Data_End_flag', False)
        setattr(self, 'scanner_Data', [])
        # Real Time Bars
        setattr(self, 'real_timeBar', [])
    # Account and Portfolio ##################################################
    def updateAccountValue(self, key, value, currency, accountName):
        update_AccountValue = self.update_AccountValue
        update_AccountValue.append((key, value, currency, accountName))

    def updatePortfolio(self, contract, position, marketPrice, marketValue,
                        averageCost, unrealizedPnL, realizedPnL, accountName):
        update_Portfolio = self.update_Portfolio
        update_Portfolio.append((contract.m_conId, contract.m_currency,
                                 contract.m_expiry, contract.m_includeExpired,
                                 contract.m_localSymbol, contract.m_multiplier,
                                 contract.m_primaryExch, contract.m_right,
                                 contract.m_secType, contract.m_strike,
                                 contract.m_symbol, contract.m_tradingClass,
                                 position, marketPrice, marketValue,
                                 averageCost, unrealizedPnL, realizedPnL,
                                 accountName))

    def updateAccountTime(self, timeStamp):
        self.update_AccountTime = timeStamp

    def accountDownloadEnd(self, accountName=None):
        self.accountDownloadEnd_accountName = accountName
        self.accountDownloadEnd_flag = True

    def accountSummary(self, reqId=None, account=None, tag=None, value=None,
                       currency=None):
        account_Summary = self.account_Summary
        account_Summary.append((reqId, account, tag, value, currency))

    def accountSummaryEnd(self, reqId):
        self.accountSummaryEnd_reqId = reqId
        self.account_SummaryEnd_flag = True

    def position(self, account, contract, pos, avgCost):
        update_Position = self.update_Position
        update_Position.append((account, contract.m_conId, contract.m_currency,
                                contract.m_exchange, contract.m_expiry,
                                contract.m_includeExpired, contract.m_localSymbol,
                                contract.m_multiplier, contract.m_right,
                                contract.m_secType, contract.m_strike,
                                contract.m_symbol, contract.m_tradingClass,
                                pos, avgCost))

    def positionEnd(self):
        setattr(self, 'positionEnd_flag', True)
    # Orders ##################################################################
    def orderStatus(self, orderId, status, filled, remaining, avgFillPrice,
                    permId, parentId, lastFillPrice, clientId, whyHeld):
        order_Status = self.order_Status
        order_Status.append((orderId, status, filled, remaining, avgFillPrice,
                            permId, parentId, lastFillPrice, clientId, whyHeld))

    def openOrder(self, orderId, contract, order, orderState):
        open_Order = self.open_Order
        open_Order.append((orderId, contract, order, orderState))

    def openOrderEnd(self):
        setattr(self, 'open_OrderEnd_flag', True)

    def nextValidId(self, orderId):
        self.next_ValidId = orderId

    def deltaNeutralValidation(self, reqId, underComp):
        pass
```

```
# Market Data ##############################################################
def tickPrice(self, tickerId, field, price, canAutoExecute):
    tick_Price = self.tick_Price
    tick_Price.append((tickerId, field, price, canAutoExecute))

def tickSize(self, tickerId, field, size):
    tick_Size = self.tick_Size
    tick_Size.append((tickerId, field, size))

def tickOptionComputation(self, tickerId, field, impliedVol, delta,
                          optPrice, pvDividend, gamma, vega, theta,
                          undPrice):
    tick_OptionComputation = self.tick_OptionComputation
    tick_OptionComputation.append((tickerId, field, impliedVol, delta,
                                   optPrice, pvDividend, gamma, vega,
                                   theta, undPrice))

def tickGeneric(self, tickerId, tickType, value):
    tick_Generic = self.tick_Generic
    tick_Generic.append((tickerId, tickType, value))

def tickString(self, tickerId, field, value):
    tick_String = self.tick_String
    tick_String.append((tickerId, field, value))

def tickEFP(self, tickerId, tickType, basisPoints, formattedBasisPoints,
            impliedFuture, holdDays, futureExpiry, dividendImpact,
            dividendsToExpiry):
    tick_EFP = self.tick_EFP
    tick_EFP.append((tickerId, tickType, basisPoints, formattedBasisPoints,
                     impliedFuture, holdDays, futureExpiry, dividendImpact,
                     dividendsToExpiry))

def tickSnapshotEnd(self, reqId):
    self.tickSnapshotEnd_reqId = reqId
    setattr(self, 'tickSnapshotEnd_flag', True)

def marketDataType(self, reqId, marketDataType):
    setattr(self, 'market_DataType', marketDataType)
    print "market_DataType" + str(self.market_DataType)
# Connection and Server ####################################################
def currentTime(self, time):
    self.current_Time = time

def error(self, id=None, errorCode=None, errorString=None):
    #print id
    print [id, errorCode, errorString]

def error_0(self, strval=None):
    print "error_0"

def error_1(self, id=0, errorCode=None, errorMsg=None):
    print "error_1"

'''def error_0(self, strval):
    pass

def error_1(self, id, errorCode, errorMsg):
    pass'''

def connectionClosed(self):
    self.connection_Closed = True


# Executions ###############################################################
def execDetails(self, reqId, contract, execution):
    self.exec_Details_reqId = reqId
    self.exec_Details_contract = contract
    self.exec_Details_execution = execution

def execDetailsEnd(self, reqId):
    self.exec_DetailsEnd_reqId = reqId
    setattr(self, "exec_DetailsEnd_flag", True)

def commissionReport(self, commissionReport):
    self.commission_Report = commissionReport

# Contract #################################################################
def contractDetails(self, reqId, contractDetails):
    self.contract_Details_reqId = reqId
    self.contract_Details = contractDetails

def contractDetailsEnd(self, reqId):
    self.contract_DetailsEnd_reqId = reqId
    self.contract_Details_flag = True

def bondContractDetails(self, reqId, contractDetails):
    self.bond_ContractDetails_reqId = reqId
    self.bond_ContractDetails = contractDetails

# Market Depth #############################################################
def updateMktDepth(self, tickerId, position, operation, side, price, size):
    update_MktDepth = self.update_MktDepth
    update_MktDepth.append((tickerId, position, operation, side, price, size))
    #df = pd.DataFrame(self.update_MktDepth, columns = ["tickerId", "position",
    #                                                  "operation", "side",
    #                                                  "price", "size"])
def updateMktDepthL2(self, tickerId, position, marketMaker, operation,
                     side, price, size):
    # I don't get any of this so I can't test it. Following are just place holders.
    print "blah blah. You have L2 data!!!"
    update_MktDepthL2 = self.update_MktDepthL2
    update_MktDepthL2.append((tickerId, position, operation, side,
                              price, size))



# News Bulletin ############################################################
def updateNewsBulletin(self, msgId, msgType, message, origExchange):
    # During the time I test this, I don't get anything. Can't verify. Sorry.
    print "You get News!!!"
    self.update_NewsBulletin_msgId = msgId
    self.update_NewsBulletin_msgType = msgType
    self.update_NewsBulletin_message = message
    self.update_NewsBulletin_origExchange = origExchange


# Financial Advisors #######################################################
def managedAccounts(self, accountsList):
    self.managed_Accounts = accountsList

def receiveFA(self, faDataType, xml):
    pass


# Historical Data  #########################################################
def historicalData(self, reqId, date, open, high, low, close, volume,
                   count, WAP, hasGaps):
    historical_Data = self.historical_Data
    historical_Data.append((reqId, date, open, high, low, close, volume,
                   count, WAP, hasGaps))
    #df = pd.DataFrame(self.historical_Data, columns = ["reqId", "date", "open",
    #                                                    "high", "low", "close",
    #                                                    "volume", "count", "WAP",
    #                                                    "hasGaps"])


# Market Scanners  #########################################################
def scannerParameters(self, xml):
    self.scanner_Parameters = xml

def scannerData(self, reqId, rank, contractDetails, distance, benchmark,
                projetion, legsStr):
    scanner_Data = self.scanner_Data
    scanner_Data.append((reqId, rank, contractDetails, distance, benchmark,
                projetion, legsStr))

def scannerDataEnd(self, reqId):
    self.scanner_Data_End_reqID = reqId
    self.scanner_Data_End_flag = True
```

```
# Real Tume Bars ###########################################################
def realtimeBar(self, reqId, time, open, high, low, close, volume,
                wap, count):
    real_timeBar = self.real_timeBar
    real_timeBar.append((reqId, time, open, high, low, close, volume,
                         wap, count))
    #df = pd.DataFrame(self.real_timeBar, columns = ["reqId", "time", "open", "high",
    #                                               "low", "close", "volume", "wap",
    #                                               "count"])


# Fundamental Data #########################################################
def fundamentalData(self, reqId, data):
    print "Getting Fundamental Data Feed Through"
    self.fundamental_Data_reqId = reqId
    self.fundamental_Data_data = data


# Display Groups ##########################################################
def displayGroupList(self, reqId, groups):
    pass

def displayGroupUpdate(self, reqId, contractInfo):
    pass
```

```
# Create Contract
class contract():
    def create_contract(self, symbol, secType, exchange, currency,
                        right = None, strike = None, expiry = None,
                        multiplier = None, tradingClass = None):
        contract = Contract()
        contract.m_symbol = symbol
        contract.m_secType = secType
        contract.m_exchange = exchange
        contract.m_currency = currency
        contract.m_right = right
        contract.m_strike = strike
        contract.m_expiry = expiry
        contract.m_multiplier = multiplier
        contract.m_tradingClass = tradingClass
        return contract

    def create_order(self, account, orderType, totalQuantity, action):
        order = Order()
        order.m_account = account
        order.m_orderType = orderType
        order.m_totalQuantity = totalQuantity
        order.m_action = action
        return order

    def exec_filter(self, client_id, accountName, contract):
        filt = ExecutionFilter()
        filt.m_clientId = client_id
        filt.m_acctCode = accountName
        #filt.m_time = "20160122-00:00:00"
        filt.m_symbol = contract.m_symbol
        filt.m_secType = contract.m_secType
        filt.m_exchange = contract.m_exchange
        return filt
```

```
'''
openOrder contains the following fields:
        self.tmp = [orderId, contract.m_comboLegs,
                    contract.m_comboLegsDescrip,
                    contract.m_conId,
                    contract.m_currency,
                    contract.m_exchange,
                    contract.m_expiry,
                    contract.m_includeExpired,
                    contract.m_localSymbol,
                    contract.m_multiplier,
                    contract.m_primaryExch,
                    contract.m_right,
                    contract.m_secId,
                    contract.m_secIdType,
                    contract.m_secType,
                    contract.m_strike,
                    contract.m_symbol,
                    contract.m_tradingClass,
                    contract.m_underComp,
                    order.m_account,
                    order.m_action,
                    order.m_activeStartTime,
                    order.m_activeStopTime,
                    order.m_algoParams,
                    order.m_algoStrategy,
                    order.m_allOrNone,
                    order.m_auctionStrategy,
                    order.m_auxPrice,
                    order.m_basisPoints,
                    order.m_basisPointsType,
                    order.m_blockOrder,
                    order.m_clearingAccount,
                    order.m_clearingIntent,
                    order.m_clientId,
                    order.m_continuousUpdate,
                    order.m_delta,
                    order.m_deltaNeutralAuxPrice,
                    order.m_deltaNeutralClearingAccount,
                    order.m_deltaNeutralClearingIntent,
                    order.m_deltaNeutralConId,
                    order.m_deltaNeutralDesignatedLocation,
                    order.m_deltaNeutralOpenClose,
                    order.m_deltaNeutralOrderType,
                    order.m_deltaNeutralSettlingFirm,
                    order.m_deltaNeutralShortSale,
                    order.m_deltaNeutralShortSaleSlot,
                    order.m_designatedLocation,
                    order.m_discretionaryAmt,
                    order.m_displaySize,
                    order.m_eTradeOnly,
                    order.m_exemptCode,
                    order.m_faGroup,
                    order.m_faMethod,
                    order.m_faPercentage,
                    order.m_faProfile,
                    order.m_firmQuoteOnly,
                    order.m_goodAfterTime,
                    order.m_goodTillDate,
                    order.m_hedgeParam,
                    order.m_hedgeType,
                    order.m_hidden,
                    order.m_lmtPrice,
                    order.m_minQty,
                    order.m_nbboPriceCap,
                    order.m_notHeld,
                    order.m_ocaGroup,
                    order.m_ocaType,
                    order.m_openClose,
                    order.m_optOutSmartRouting,
                    order.m_orderComboLegs,
                    order.m_orderId,
                    order.m_orderRef,
                    order.m_orderType,
                    order.m_origin,
                    order.m_outsideRth,
                    order.m_overridePercentageConstraints,
                    order.m_parentId,
                    order.m_percentOffset,
                    order.m_permId,
                    order.m_referencePriceType,
                    order.m_rule80A,
                    order.m_scaleAutoReset,
                    order.m_scaleInitFillQty,
                    order.m_scaleInitLevelSize,
                    order.m_scaleInitPosition,
                    order.m_scalePriceAdjustInterval,
                    order.m_scalePriceAdjustValue,
                    order.m_scalePriceIncrement,
                    order.m_scaleProfitOffset,
                    order.m_scaleRandomPercent,
                    order.m_scaleSubsLevelSize,
                    order.m_scaleTable,
                    order.m_settlingFirm,
                    order.m_shortSaleSlot,
                    order.m_smartComboRoutingParams,
                    order.m_startingPrice,
                    order.m_stockRangeLower,
                    order.m_stockRangeUpper,
                    order.m_stockRefPrice,
                    order.m_sweepToFill,
                    order.m_tif,
                    order.m_totalQuantity,
                    order.m_trailStopPrice,
                    order.m_trailingPercent,
                    order.m_transmit,
                    order.m_triggerMethod,
                    order.m_volatility,
                    order.m_volatilityType,
                    order.m_whatIf,
                    orderState.m_commission,
                    orderState.m_commissionCurrency,
                    orderState.m_equityWithLoan,
                    orderState.m_initMargin,
                    orderState.m_maintMargin,
                    orderState.m_maxCommission,
                    orderState.m_minCommission,
                    orderState.m_status,
                    orderState.m_warningText]
'''
```

Part III

Quick Reference

Chapter 7

Summary

EClientSocket Methods	EWrapper Methods
Account and Portfolio Group	

reqAcccountUpdates()	updateAccountValue()
	updatePortfolio()
	updateAccountTime()
	accountDownloadEnd()

reqAccountSummary()	accountSummary()
	accountSummaryEnd()

cancelAccountSummary()	

reqPositions()	position()
	positionEnd()

cancelPositions()	
Order Group	

placeOrder()
cancelOrder()	

reqOpenOrders()
reqAllOpenOrders()
redAutoOpenOrders()
	orderStatus()
	openOrder()
	openorderEnd()

reqIDs()	nextValidID()
exerciseOptions()	
reqGlobalCancel()	
	deltaNeutralValidation()