VOLUME III

INTERACTIVE BROKERS (IB)
JAVA API GUIDE

Anthony Ng
Department Name: Banking and Finance
University Name: Nanyang Polytechnic
Country: Singapore
Project Start Date: 1 December 2014
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

Part I

IB Java Manual

Chapter 1

Account and Portfolio Group

1.1 Account and Portfolio EClientSocket Methods

1.1.1 reqAccountUpdates()

Call this function to start getting account values, portfolio, and last update time information. The account data will be fed back through the updateAccountTime(), updateAccountValue() and updatePortfolio() EWrapper methods.

void reqAccountUpdates (boolean subscribe, String acctCode)

Parameter	Type	Description
subscribe	boolean	If set to TRUE, the client will start receiving account and portfolio updates. If set to FALSE, the client will stop receiving this information.
acctCode	String	The account code for which to receive account and portfolio updates.

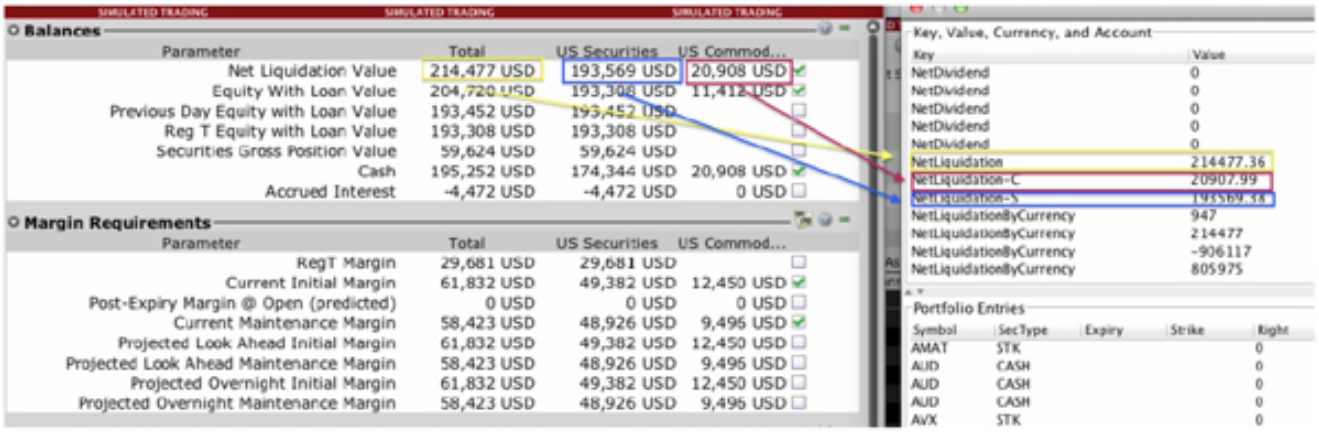
The account information resulting from the invocation of reqAccountUpdates() is the same information that appears in Trader Workstations Account Window. When trying to determine the definition of each variable or key within the API account data, it is essential that you use the TWS Account Window as guidance.

To identify API Account keys:

The APIs updateAccountValue() event handler delivers all of the account information.

- Strings or keys with a suffix of -C, such as AvailableFunds-C, EquityForInitial-C, NetLiquidation-C, correspond to Commodities in the TWS Account Window.
- Keys with a suffix of -S, such as EquityForMaintenance-S, FullAvailableFunds-S or NetLiquidation-S, correspond to Securities in the TWS Account Window.
- Keys without any suffix correspond to Totals in the TWS Account Window.

The image below is an actual example of how to compare TWS's Account Window and the APIs account data. In this particular case, we try to link three specific keys NetLiquidation, NetLiquidation-C, and NetLiquidation-S to the TWS Account Window.



For more information about the information presented in the TWS Account Window, see Account Window in section 23.1.

1.1.2 reqAccountSummary()

Call this method to request and keep up to date the data that appears on the TWS Account Window Summary tab. The data is returned by accountSummary().

Note: This request can only be made when connected to a Financial Advisor (FA) account.

void reqAccountSummary(int reqId, String group, String tags)

Parameter	Type	Description
reqId	int	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
group	String	Set to *All* to return account summary data for all accounts, or set to a specific Advisor Account Group name that has already been created in TWS Global Configuration.
tags	String	A comma-separated list of account tags. Available tags are: • AccountType • NetLiquidation, • TotalCashValue - Total cash including futures pnl • SettledCash - For cash accounts, this is the same as TotalCashValue • AccruedCash - Net accrued interest • BuyingPower - The maximum amount of marginable US stocks the account can buy • EquityWithLoanValue - Cash + stocks + bonds + mutual funds • PreviousDayEquityWithLoanValue, • GrossPositionValue - The sum of the absolute value of all stock and equity option positions • RegTEquity, • RegTMargin, • SMA - Special Memorandum Account • InitMarginReq, • MaintMarginReq, • AvailableFunds, • ExcessLiquidity, • Cushion - Excess liquidity as a percentage of net liquidation value • FullInitMarginReq, • FullMaintMarginReq, • FullAvailableFunds,

Parameter	Type	Description
tags	String	A comma-separated list of account tags. Available tags are: • FullExcessLiquidity, • LookAheadNextChange - Time when look-ahead values take effect • LookAheadInitMarginReq, • LookAheadMaintMarginReq, • LookAheadAvailableFunds, • LookAheadExcessLiquidity, • HighestSeverity - A measure of how close the account is to liquidation • DayTradesRemaining - The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1" means that the user can put on unlimited day trades. • Leverage - GrossPositionValue / NetLiquidation

1.1.3 cancelAccountSummary()

Cancels the request for Account Window Summary tab data.

void cancelAccountSummary(int messageId, int version As Integer, int reqId As Integer)

Parameter	Type	Description
messageId	Integer	Set this to 63.
version	Integer	Set this to 1.
reqId	Integer	The ID of the data request being canceled.

1.1.4 reqPositions()

Requests real-time position data for all accounts.

void reqPositions()

1.1.5 cancelPositions()

Cancels real-time position updates.

void cancelPositions()

1.2 Account and Portfolio EWrapper Methods

1.2.1 updateAccountValue()

This method is called only when reqAccountUpdates() method on the EClientSocket object has been called.

void updateAccountValue(String key, String value, String currency, String accountName)

Parameter	Type	Description
key	String	A string that indicates one type of account value. There is a long list of possible keys that can be sent, here are just a few examples: • CashBalance - Account cash balance • Currency - Currency string* • DayTradesRemaining - Number of day trades left • EquityWithLoanValue - Equity with Loan Value • InitMarginReq - Current initial margin requirement • LongOptionValue - Long option value* • MaintMarginReq - Current maintenance margin • NetLiquidation - Net liquidation value • OptionMarketValue - Option market value* • ShortOptionValue - Short option value* • StockMarketValue - Stock market value* • UnalteredInitMarginReq - Overnight initial margin requirement* • UnalteredMaintMarginReq - Overnight maintenance margin requirement*
value	String	The value associated with the key.
currency	String	Defines the currency type, in case the value is a currency type.
account	String	States the account to which the message applies. Useful for Financial Advisor sub-account messages.

Note: * is only listed in the C++ portion.

Example - updateAccountValue()

updateAccountValue	value	currency	accountName
updateAccountValue key=AccountType	value=INDIVIDUAL	currency=None	accountName=DU274390
updateAccountValue key=AccountCode	value=DU274390	currency=None	accountName=DU274390
updateAccountValue key=AccountReady	value=true	currency=None	accountName=DU274390
updateAccountValue key=Cushion	value=0.641286	currency=None	accountName=DU274390
updateAccountValue key=DayTradesRemaining	value=-1	currency=None	accountName=DU274390
updateAccountValue key=DayTradesRemainingT+1	value=-1	currency=None	accountName=DU274390
updateAccountValue key=DayTradesRemainingT+2	value=-1	currency=None	accountName=DU274390
updateAccountValue key=DayTradesRemainingT+3	value=-1	currency=None	accountName=DU274390
updateAccountValue key=DayTradesRemainingT+4	value=-1	currency=None	accountName=DU274390
updateAccountValue key=Leverage-S	value=2.11	currency=None	accountName=DU274390
updateAccountValue key=LookAheadNextChange	value=0	currency=None	accountName=DU274390
updateAccountValue key=SegmentTitle-C	value=US Commodities	currency=None	accountName=DU274390
updateAccountValue key=SegmentTitle-S	value=US Securities	currency=None	accountName=DU274390
updateAccountValue key=TradingType-S	value=STKNOPT	currency=None	accountName=DU274390
updateAccountValue key=WhatIfPMEnabled	value=true	currency=None	accountName=DU274390
updateAccountValue key=AccruedCash	value=-1736.75	currency=USD	accountName=DU274390
updateAccountValue key=AccruedCash-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=AccruedCash-S	value=-1736.75	currency=USD	accountName=DU274390
updateAccountValue key=AccruedDividend	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=AccruedDividend-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=AccruedDividend-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=AvailableFunds	value=828014.60	currency=USD	accountName=DU274390
updateAccountValue key=AvailableFunds-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=AvailableFunds-S	value=363212.20	currency=USD	accountName=DU274390
updateAccountValue key=Billable	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=Billable-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=Billable-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=BuyingPower	value=3312058.40	currency=USD	accountName=DU274390
updateAccountValue key=EquityWithLoanValue	value=1292041.17	currency=USD	accountName=DU274390
updateAccountValue key=EquityWithLoanValue-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=EquityWithLoanValue-S	value=827238.77	currency=USD	accountName=DU274390
updateAccountValue key=ExcessLiquidity	value=828568.33	currency=USD	accountName=DU274390
updateAccountValue key=ExcessLiquidity-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=ExcessLiquidity-S	value=363765.93	currency=USD	accountName=DU274390
updateAccountValue key=FullAvailableFunds	value=828014.60	currency=USD	accountName=DU274390
updateAccountValue key=FullAvailableFunds-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=FullAvailableFunds-S	value=363212.20	currency=USD	accountName=DU274390
updateAccountValue key=FullExcessLiquidity	value=828568.33	currency=USD	accountName=DU274390
updateAccountValue key=FullExcessLiquidity-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=FullExcessLiquidity-S	value=363765.93	currency=USD	accountName=DU274390
updateAccountValue key=FullInitMarginReq	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=FullInitMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=FullInitMarginReq-S	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=FullMaintMarginReq	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=FullMaintMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=FullMaintMarginReq-S	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=GrossPositionValue	value=1742788.46	currency=USD	accountName=DU274390
updateAccountValue key=GrossPositionValue-S	value=1742788.46	currency=USD	accountName=DU274390
updateAccountValue key=IndianStockHaircut	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=IndianStockHaircut-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=IndianStockHaircut-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=InitMarginReq	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=InitMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=InitMarginReq-S	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadAvailableFunds	value=828014.60	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadAvailableFunds-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadAvailableFunds-S	value=363212.20	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadExcessLiquidity	value=828568.33	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadExcessLiquidity-C	value=464802.40	currency=USD	accountName=DU274390

updateAccountValue	value	currency	accountName
updateAccountValue key=LookAheadExcessLiquidity-S	value=363765.93	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadInitMarginReq	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadInitMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadInitMarginReq-S	value=464026.57	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadMaintMarginReq	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadMaintMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=LookAheadMaintMarginReq-S	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=MaintMarginReq	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=MaintMarginReq-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=MaintMarginReq-S	value=463472.84	currency=USD	accountName=DU274390
updateAccountValue key=NetLiquidation	value=1292041.17	currency=USD	accountName=DU274390
updateAccountValue key=NetLiquidation-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=NetLiquidation-S	value=827238.77	currency=USD	accountName=DU274390
updateAccountValue key=PASharesValue	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PASharesValue-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PASharesValue-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationExcess	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationExcess-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationExcess-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationMargin	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationMargin-C	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PostExpirationMargin-S	value=0.00	currency=USD	accountName=DU274390
updateAccountValue key=PreviousDayEquityWithLoanValue	value=826886.67	currency=USD	accountName=DU274390
updateAccountValue key=PreviousDayEquityWithLoanValue-S	value=826886.67	currency=USD	accountName=DU274390
updateAccountValue key=RegTEquity	value=827238.77	currency=USD	accountName=DU274390
updateAccountValue key=RegTEquity-S	value=827238.77	currency=USD	accountName=DU274390
updateAccountValue key=RegTMargin	value=871394.23	currency=USD	accountName=DU274390
updateAccountValue key=RegTMargin-S	value=871394.23	currency=USD	accountName=DU274390
updateAccountValue key=SMA	value=889633.98	currency=USD	accountName=DU274390
updateAccountValue key=SMA-S	value=889633.98	currency=USD	accountName=DU274390
updateAccountValue key=TotalCashValue	value=615344.49	currency=USD	accountName=DU274390
updateAccountValue key=TotalCashValue-C	value=464802.40	currency=USD	accountName=DU274390
updateAccountValue key=TotalCashValue-S	value=150542.09	currency=USD	accountName=DU274390
updateAccountValue key=Currency	value=AUD	currency=AUD	accountName=DU274390
updateAccountValue key=CashBalance	value=1671	currency=AUD	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=1671	currency=AUD	accountName=DU274390
updateAccountValue key=AccruedCash	value=-843	currency=AUD	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=827	currency=AUD	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=ExchangeRate	value=0.6987	currency=AUD	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=AUD	accountName=DU274390
updateAccountValue key=RealCurrency	value=AUD	currency=AUD	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=AUD	accountName=DU274390
updateAccountValue key=Currency	value=BASE	currency=BASE	accountName=DU274390
updateAccountValue key=CashBalance	value=615344	currency=BASE	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=615344	currency=BASE	accountName=DU274390

updateAccountValue	value	currency	accountName
updateAccountValue key=AccruedCash	value=-1737	currency=BASE	accountName=DU274390
updateAccountValue key=StockMarketValue	value=678433	currency=BASE	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=1292041	currency=BASE	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=-76593	currency=BASE	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=ExchangeRate	value=1.00	currency=BASE	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=BASE	accountName=DU274390
updateAccountValue key=RealCurrency	value=BASE	currency=BASE	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=BASE	accountName=DU274390
updateAccountValue key=Currency	value=CAD	currency=CAD	accountName=DU274390
updateAccountValue key=CashBalance	value=70148	currency=CAD	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=70148	currency=CAD	accountName=DU274390
updateAccountValue key=AccruedCash	value=-292	currency=CAD	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=69856	currency=CAD	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=ExchangeRate	value=0.7026988	currency=CAD	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=CAD	accountName=DU274390
updateAccountValue key=RealCurrency	value=CAD	currency=CAD	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=CAD	accountName=DU274390
updateAccountValue key=Currency	value=CNH	currency=CNH	accountName=DU274390
updateAccountValue key=CashBalance	value=-67825	currency=CNH	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=-67825	currency=CNH	accountName=DU274390
updateAccountValue key=AccruedCash	value=-1626	currency=CNH	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=-69451	currency=CNH	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=ExchangeRate	value=0.1520346	currency=CNH	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=CNH	accountName=DU274390

updateAccountValue	value	currency	accountName
updateAccountValue key=MoneyMarketFundValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=CNH	accountName=DU274390
updateAccountValue key=RealCurrency	value=CNH	currency=CNH	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=CNH	accountName=DU274390
updateAccountValue key=Currency	value=EUR	currency=EUR	accountName=DU274390
updateAccountValue key=CashBalance	value=22085	currency=EUR	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=22085	currency=EUR	accountName=DU274390
updateAccountValue key=AccruedCash	value=-276	currency=EUR	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=21809	currency=EUR	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=ExchangeRate	value=1.087785	currency=EUR	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=EUR	accountName=DU274390
updateAccountValue key=RealCurrency	value=EUR	currency=EUR	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=EUR	accountName=DU274390
updateAccountValue key=Currency	value=GBP	currency=GBP	accountName=DU274390
updateAccountValue key=CashBalance	value=-16866	currency=GBP	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=-16866	currency=GBP	accountName=DU274390
updateAccountValue key=AccruedCash	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=-16866	currency=GBP	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=ExchangeRate	value=1.45408	currency=GBP	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=GBP	accountName=DU274390
updateAccountValue key=RealCurrency	value=GBP	currency=GBP	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=GBP	accountName=DU274390
updateAccountValue key=Currency	value=HKD	currency=HKD	accountName=DU274390
updateAccountValue key=CashBalance	value=-1649	currency=HKD	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=-1649	currency=HKD	accountName=DU274390

updateAccountValue	value	currency	accountName
updateAccountValue key=AccruedCash	value=-2352	currency=HKD	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=-4002	currency=HKD	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=ExchangeRate	value=0.128851	currency=HKD	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=HKD	accountName=DU274390
updateAccountValue key=RealCurrency	value=HKD	currency=HKD	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=HKD	accountName=DU274390
updateAccountValue key=Currency	value=JPY	currency=JPY	accountName=DU274390
updateAccountValue key=CashBalance	value=5258450	currency=JPY	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=5258450	currency=JPY	accountName=DU274390
updateAccountValue key=AccruedCash	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=StockMarketValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=5258450	currency=JPY	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=ExchangeRate	value=0.0085039	currency=JPY	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=MoneyMarketFundValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=JPY	accountName=DU274390
updateAccountValue key=RealCurrency	value=JPY	currency=JPY	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=JPY	accountName=DU274390
updateAccountValue key=Currency	value=USD	currency=USD	accountName=DU274390
updateAccountValue key=CashBalance	value=531193	currency=USD	accountName=DU274390
updateAccountValue key=TotalCashBalance	value=531193	currency=USD	accountName=DU274390
updateAccountValue key=AccruedCash	value=-92	currency=USD	accountName=DU274390
updateAccountValue key=StockMarketValue	value=678433	currency=USD	accountName=DU274390
updateAccountValue key=OptionMarketValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=FutureOptionValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=FuturesPNL	value=0	currency=USD	accountName=DU274390
updateAccountValue key=NetLiquidationByCurrency	value=1209534	currency=USD	accountName=DU274390
updateAccountValue key=UnrealizedPnL	value=-76593	currency=USD	accountName=DU274390
updateAccountValue key=RealizedPnL	value=0	currency=USD	accountName=DU274390
updateAccountValue key=ExchangeRate	value=1.00	currency=USD	accountName=DU274390
updateAccountValue key=FundValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=NetDividend	value=0	currency=USD	accountName=DU274390
updateAccountValue key=MutualFundValue	value=0	currency=USD	accountName=DU274390

updateAccountValue	**value**	**currency**	**accountName**
updateAccountValue key=MoneyMarketFundValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=CorporateBondValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=TBondValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=TBillValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=WarrantValue	value=0	currency=USD	accountName=DU274390
updateAccountValue key=FxCashBalance	value=0	currency=USD	accountName=DU274390
updateAccountValue key=AccountOrGroup	value=DU274390	currency=USD	accountName=DU274390
updateAccountValue key=RealCurrency	value=USD	currency=USD	accountName=DU274390
updateAccountValue key=IssuerOptionValue	value=0	currency=USD	accountName=DU274390

1.2.2 updatePortfolio()

This method is called only when reqAccountUpdates() method on the EClientSocket object has been called.

void updatePortfolio(Contract contract, int position, double marketPrice, double marketValue, double averageCost, double unrealizedPNL, double realizedPNL, String accountName)

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.

1.2.3 updateAccountTime()

This method is called only when reqAccountUpdates() method on the EClientSocket object has been called.

void updateAccountTime(String timeStamp)

Parameter	Type	Description
timeStamp	String	This indicates the last update time of the account information.

1.2.4 accountDownloadEnd()

This event is called after a batch updateAccountValue() and updatePortfolio() is sent.

void accountDownloadEnd(String accountName)

Parameter	Type	Description
accountName	String	The name of the account.

1.2.5 accountSummary()

Returns the data from the TWS Account Window Summary tab in response to reqAccountSummary().

void accountSummary(int reqId, String account, String tag, String value, String currency)

Parameter	Type	Description
reqId	int	The ID of the data request.
account	String	The account ID.
tag	String	The tag from the data request. Available tags are: • AccountType • TotalCashValue - Total cash including futures pnl • SettledCash - For cash accounts, this is the same as TotalCashValue • AccruedCash - Net accrued interest • BuyingPower - The maximum amount of marginable US stocks the account can buy • EquityWithLoanValue - Cash + stocks + bonds + mutual funds • PreviousEquityWithLoanValue • GrossPositionValue - The sum of the absolute value of all stock and equity option positions • RegTEquity • RegTMargin • SMA - Special Memorandum Account • InitMarginReq • MaintMarginReq • AvailableFunds • ExcessLiquidity • Cushion - Excess liquidity as a percentage of net liquidation value • FullInitMarginReq • FullMaintMarginReq • FullAvailableFunds • FullExcessLiquidity • LookAheadNextChange - Time when look-ahead values take effect • LookAheadInitMarginReq • LookAheadMaintMarginReq • LookAheadAvailableFunds • LookAheadExcessLiquidity • HighestSeverity - A measure of how close the account is to liquidation • DayTradesRemaining - The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1" means that the user can put on unlimited day trades. • Leverage - GrossPositionValue / NetLiquidation
value	String	The value of the tag.
currency	String	The currency of the tag.

1.2.6 accountSummaryEnd()

This method is called once all account summary data for a given request are received.

void accountSummaryEnd(int reqId)

Parameter	Type	Description
reqId	int	The ID of the data request.

1.2.7 position()

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.

1.2.8 positionEnd()

This is called once all position data for a given request are received and functions as an end marker for the position() data.

void positionEnd()

Chapter 2

Order Group

2.1 Order EClientSocket Methods

2.1.1 placeOrder()

void placeOrder(int id, Contract contract, Order order)

Parameter	Type	Description
id	int	The order Id. **You must specify a unique value**. When the order status returns, it will be identified by this tag. This tag is also used when canceling the order.
contract	Contract	This class contains attributes used to describe the contract.
order	Order	This structure contains the details of the order. Note: Each client MUST connect with a unique clientId.

2.1.2 cancelOrder()

Call this method to cancel an order.

void cancelOrder(int id)

Parameter	Type	Description
id	int	The order Id that was specified previously in the call to placeOrder()

2.1.3 reqOpenOrders()

Call this method to request any open orders that were placed from this API client. Each open order will be fed back through the `openOrder()` and `orderStatus()` methods on the EWrapper.

Note: The client with a clientId of ”0” will also receive the TWS-owned open orders. These orders will be associated with the client and a new orderId will be generated. This association will persist over multiple API and TWS sessions.

void reqOpenOrders()

2.1.4 reqAllOpenOrders()

Call this method to request all open orders that were placed from all API clients linked to one TWS, and also from the TWS. Note that you can run up to 8 API clients from a single TWS. Each open order will be fed back through the `openOrder()` and `orderStatus()` methods on the EWrapper.

Note: No association is made between the returned orders and the requesting client. **void reqAllOpenOrders()**

2.1.5 reqAutoOpenOrders()

Call this method to request that newly created TWS orders be implicitly associated with the client. When a new TWS order is created, the order will be associated with the client and automatically fed back through the `openOrder()` and `orderStatus()` methods on the EWrapper.

Note: TWS orders can only be bound to clients with a clientId of 0.

void reqAutoOpenOrders(boolean bAutoBind)

Parameter	Type	Description
AutoBind	boolean	If set to TRUE, newly created TWS orders will be implicitly associated with the client. If set to FALSE, no association will be made.

2.1.6 reqIDs()

Call this method to request the next valid ID that can be used when placing an order. After calling this method, the `nextValidId()` event will be triggered, and the id returned is that next valid ID. That ID will reflect any autobinding that has occurred (which generates new IDs and increments the next valid ID therein).

Public synchronized Void reqIds (int numIds)

Parameter	Type	Description
numIds	int	Set to 1.

2.1.7 exerciseOptions()

Call the `exerciseOptions()` method to exercise options. Note: SMART is not an allowed exchange in `exerciseOptions()` calls, and TWS does a request for the position in question whenever any API initiated exercise or lapse is attempted.

void exerciseOptions(int tickerId, Contract contract, int exerciseAction, int exerciseQuantity, String account, int override)

Parameter	Type	Description
tickerId	int	The Id for the exercise request.
contract	Contract	This class contains attributes used to describe the contract.
exerciseAction	int	Specifies whether to exercise the specified option or let the option lapse. Valid values are: • 1 = exercise • 2 = lapse
exerciseQuantity	int	The number of contracts to be exercised
account	String	For institutional orders. Specifies the IB account.
override	int	Specifies whether your setting will override the system's natural action. For example, if your action is "exercise" and the option is not in-the-money, by natural action the option would not exercise. If you have override set to "yes" the natural action would be overridden and the out-of-the money option would be exercised. Values are: • 0 = do not override • 1 = override

2.1.8 reqGlobalCancel()

Use this method to cancel all open orders globally. It cancels both API and TWS open orders.

If the order was created in TWS, it also gets canceled. If the order was initiated in the API, it also gets canceled.

void reqGlobalCancel()

2.2 Order EWrapper Methods

2.2.1 orderStatus()

This method is called whenever the status of an order changes. It is also fired after reconnecting to TWS if the client has any open orders.

void orderStatus(int orderId, String status, int filled, int remaining, double avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, String whyHeld)

Note: It is possible that orderStatus() may return duplicate messages. It is essential that you filter the message accordingly.

Parameter	Type	Description
id	int	The order Id that was specified previously in the call to `placeOrder()`
status	String	The order status. Possible values include: • **PendingSubmit** - indicates that you have transmitted the order, but have not yet received confirmation that it has been accepted by the order destination. NOTE: This order status is not sent by TWS and should be explicitly set by the API developer when an order is submitted. • **PendingCancel** - indicates that you have sent a request to cancel the order but have not yet received cancel confirmation from the order destination. At this point, your order is not confirmed canceled. You may still receive an execution while your cancellation request is pending. NOTE: This order status is not sent by TWS and should be explicitly set by the API developer when an order is canceled. • **PreSubmitted** - indicates that a simulated order type has been accepted by the IB system and that this order has yet to be elected. The order is held in the IB system until the election criteria are met. At that time the order is transmitted to the order destination as specified . • **Submitted** - indicates that your order has been accepted at the order destination and is working. • **Cancelled** - indicates that the balance of your order has been confirmed canceled by the IB system. This could occur unexpectedly when IB or the destination has rejected your order. • **Filled** - indicates that the order has been completely filled. • **Inactive** - indicates that the order has been accepted by the system (simulated orders) or an exchange (native orders) but that currently the order is inactive due to system, exchange or other issues.
filled	int	Specifies the number of shares that have been executed. For more information about partial fills, see Order Status for Partial Fills.
remaining	int	Specifies the number of shares still outstanding.
avgFillPrice	double	The average price of the shares that have been executed. This parameter is valid only if the **filled** parameter value is greater than zero. Otherwise, the price parameter will be zero.
permId	int	The TWS id used to identify orders. Remains the same over TWS sessions.

Parameter	Type	Description
parentId	int	The order ID of the parent order, used for bracket and auto trailing stop orders.
lastFilledPrice	double	The last price of the shares that have been executed. This parameter is valid only if the filled parameter value is greater than zero. Otherwise, the price parameter will be zero.
clientId	int	The ID of the client (or TWS) that placed the order. Note that TWS orders have a fixed clientId and orderId of 0 that distinguishes them from API orders.
whyHeld	String	This field is used to identify an order held when TWS is trying to locate shares for a short sell. The value used to indicate this is 'locate'.

Order Status for Partial Fills and Example

The following example demonstrates how the orderStatus() event behaves when there is a partial fill of an order.

You place an order for 1000 shares of XYZ stock. There are four separate executions before the order for 1000 total shares is completed. The first partial fill executes with 200, then the second partial fill executes with 200 shares. The third partial fill executes with another 200 shares and finally, the last partial fill executes with 400 shares.

- First execution: 200
- Second execution: 200
- Third execution: 200
- Fourth execution: 400

In the `orderStatus()` event, here is the sequence of status messages that will be received by the API based on this example:

Execution	Status Message
1st Execution	Status = Submitted Filled qnty = 200 Remaining qnty = 800
2nd Execution	Status = Submitted Filled qnty = 400 Remaining qnty = 600
3rd Execution	Status = Submitted Filled qnty = 600 Remaining qnty = 400
4th Execution	Status = Filled Filled qnty = 1000 Remaining qnty = 0

2.2.2 openOrder()

This method is called to feed in open orders.

void openOrder(int orderId, Contract contract, Order order, OrderState orderState)

Parameter	Type	Description
orderId	int	The order Id assigned by TWS. Used to cancel or update the order.
contract	Contract	The Contract class attributes describe the contract.
order	Order	The Order class attributes define the details of the order.
orderState	OrderState	The orderState attributes include margin and commissions fields for both pre and post trade data.

2.2.3 openOrderEnd()

This is called at the end of a given request for open orders.

void openOrderEnd()

2.2.4 nextValidId()

This method is called after a successful connection to TWS.

void nextValidId(int orderId)

Parameter	Type	Description
orderId	int	The next available order Id received from TWS upon connection. Increment all successive orders by one based on this Id.

2.2.5 deltaNeutralValidation()

Upon accepting a Delta-Neutral RFQ(request for quote), the server sends a deltaNeutralValidation() message with the UnderComp structure. If the delta and price fields are empty in the original request, the confirmation will contain the current values from the server. These values are locked when the RFQ is processed and remain locked until the RFQ is canceled.

void deltaNeutralValidation(int reqId, UnderComp underComp)

Parameter	Type	Description
reqID	int	The Id of the data request.
underComp	UnderComp	Underlying component.

Chapter 3

Market Data Group

3.1 Market Data EClientSocket Methods

3.1.1 reqMktData()

Call this method to request market data. The market data will be returned by the tickPrice(), tickSize(), tickOptionComputation(), tickGeneric(), tickString() and tickEFP() methods.

void reqMktData(int tickerId, Contract contract, String genericTicklist, boolean snapshot, List mktDataOptions)

Parameter	Type	Description
tickerId	int	The ticker id. Must be a unique value. When the market data returns, it will be identified by this tag. This is also used when canceling the market data.
contract	Contract	This class contains attributes used to describe the contract.
genericTicklist	String	A comma delimited list of generic tick types. Tick types can be found in the Generic Tick Types page.
snapshot	boolean	Check to return a single snapshot of market data and have the market data subscription cancel. Do not enter any genericTicklist values if you use snapshot.
mktDataOptions	List	For internal use only. Use default value XYZ.

3.1.2 cancelMktData()

After calling this method, market data for the specified Id will stop flowing.

void cancelMktData(int tickerId)

Parameter	Type	Description
tickerId	int	The Id that was specified in the call to reqMktData().

3.1.3 calculateImpliedVolatility()

Call this function to calculate volatility for a supplied option price and underlying price.

calculateImpliedVolatility(int reqId, Contract optionContract, double optionPrice, double underPrice)

Parameter	Type	Description
reqId	int	The ticker id.
optionContract	Contract	Describes the contract.
optionPrice	double	The price of the option.
underPrice	double	Price of the underlying.

3.1.4 cancelcalculateImpliedVolatility()

Call this function to cancel a request to calculate volatility for a supplied option price and underlying price.

cancelcalculateImpliedVolatility(int reqId)

Parameter	Type	Description
reqId	int	The ticker id.

3.1.5 calculateOptionPrice()

Call this function to calculate option price and greek values for a supplied volatility and underlying price.

void calculateOptionPrice(int reqId, Contract contract, double volatility, double underPrice)

Parameter	Type	Description
conid	int	The ticker ID.
contract	Contract	Describes the contract.
volatility	double	The volatility.
underPrice	double	Price of the underlying.

3.1.6 cancelCalculateOptionPrice()

Call this function to cancel a request to calculate the option price and greek values for a supplied volatility and underlying price.

cancelCalculateOptionPrice(int reqId)

Parameter	Type	Description
reqId	int	The ticker id.

3.1.7 reqMktDataType()

The API can receive frozen market data from Trader Workstation. Frozen market data is the last data recorded in our system. During normal trading hours, the API receives real-time market data. If you use this function, you are telling TWS to automatically switch to frozen market data after the close. Then, before the opening of the next trading day, market data will automatically switch back to real-time market data.

reqMarketDataType(int type)

Parameter	Type	Description
type	int	1 for real-time streaming market data or 2 for frozen market data.

3.2 Market Data EWrapper Methods

3.2.1 tickPrice()

This method is called when the market data changes. Prices are updated immediately with no delay.

void tickPrice(int tickerId, int field, double price, int canAutoExecute)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
field	int	Specifies the type of price. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 1 will map to bidPrice, a field value of 2 will map to askPrice, etc. • 1 = bid • 2 = ask • 4 = last • 6 = high • 7 = low • 9 = close
price	here	Specifies the price for the specified field
canAutoExecute	double	Specifies whether the price tick is available for automatic execution. Possible values are: • 0 = not eligible for automatic execution • 1 = eligible for automatic execution

3.2.2 tickSize()

This method is called when the market data changes. Sizes are updated immediately with no delay.

void tickSize(int tickerId, int field, int size)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
field	int	Specifies the type of price. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 0 will map to bidSize, a field value of 3 will map to askSize, etc. • 0 = bid size • 3 = ask size • 5 = last size • 8 = volume
size	int	Specifies the size for the specified field

3.2.3 tickOptionComputation()

This method is called when the market in an option or its underlier moves. TWSs option model volatilities, prices, and deltas, along with the present value of dividends expected on that options underlier are received.

void tickOptionComputation(int tickerId, int field, double impliedVol, double delta, double optPrice, double pvDividend, double gamma, double vega, double theta, double undPrice)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
field	int	Specifies the type of option computation. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 13 will map to modelOptComp, etc. • 10 = Bid • 11 = Ask • 12 = Last
impliedVol	double	The implied volatility calculated by the TWS option modeler, using the specified ticktype value.
delta	double	The option delta value.
optPrice	double	The option price.
pvDividend	double	The present value of dividends expected on the options underlier
gamma	double	The option gamma value.
vega	double	The option vega value.
theta	double	The option theta value.
undPrice	double	The price of the underlying.

3.2.4 tickGeneric()

This method is called when the market data changes. Values are updated immediately with no delay.

void tickGeneric(int tickerId, int tickType, double value)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
tickType	int	Specifies the type of price. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 46 will map to shortable, etc.
value	double	The value of the specified field

3.2.5 tickString()

This method is called when the market data changes. Values are updated immediately with no delay.

void tickString(int tickerId, int tickType, String value)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
field	int	Specifies the type of price. Pass the field value into TickType.getField(int tick- Type) to retrieve the field description. For example, a field value of 45 will map to lastTimestamp, etc.
value	String	The value of the specified field

3.2.6 tickEFP()

This method is called when the market data changes. Values are updated immediately with no delay.

void tickEFP(int tickerId, int tickType, double basisPoints, String formattedBasisPoints, double impliedFuture, int holdDays, String futureExpiry, double dividendImpact, double dividendsToExpiry)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktData()
field	int	Specifies the type of price. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 38 will map to bidEFP, etc.
basisPoints	double	Annualized basis points, which is representative of the financing rate that can be directly compared to broker rates
formattedBasisPoints	String	Annualized basis points as a formatted string that depicts them in percentage form
impliedFuture	double	Implied futures price
holdDays	int	The number of hold days until the expiry of the EFP
futureExpiry	String	The expiration date of the single stock future
dividendImpact	double	The dividend impact upon the annualized basis points interest rate
dividendsToExpiry	double	The dividends expected until the expiration of the single stock future

3.2.7 tickSnapshotEnd()

This is called when a snapshot market data subscription has been fully handled and there is nothing more to wait for. This also covers the timeout case.

void tickSnapshotEnd(int reqId)

Parameter	Type	Description
reqID	int	Id of the data request

3.2.8 marketDataType()

TWS sends a marketDataType(type) callback to the API, where type is set to Frozen or RealTime, to announce that market data has been switched between frozen and real-time. This notification occurs only when market data switches between real-time and frozen. The marketDataType( ) callback accepts a reqId parameter and is sent per every subscription because different contracts can generally trade on a different schedule.

void marketDataType(int reqId, int marketDataType)

Parameter	Type	Description
int reqId	int	Id of the data request
marketDataType	int	1 for real-time streaming market data or 2 for frozen market data.

Chapter 4

Connection and Server Group

4.1 Connection and Server EClientSocket Methods

4.1.1 EClientSocket()

This is the constructor.

EClientSocket(AnyWrapper anyWrapper)

Parameter	Type	Description
anyWrapper	AnyWrapper	The reference to an object that was derived from the AnyWrapper base interface. Note EWrapper extends AnyWrapper.

4.1.2 eConnect()

This function must be called before any other. There is no feedback for a successful connection, but a subsequent attempt to connect will return the message "Already connected."

void eConnect(String host, int port, int clientId)

Parameter	Type	Description
host	String	The host name or IP address of the machine where TWS is running. Leave blank to connect to the local host.
port	int	Must match the port specified in TWS on the Configure>API>Socket Port field.
clientId	int	A number used to identify this client connection. All orders placed/- modified from this client will be associated with this client identifier. Note: Note: Each client MUST connect with a unique clientId.

4.1.3 eDisconnect()

Call this method to terminate the connections with TWS. Calling this method does not cancel orders that have already been sent.
void eDisconnect()

4.1.4 isConnected()

Call this method to check if there is a connection with TWS.
void isConnected()

4.1.5 setServerLogLevel()

The default level is ERROR. Refer to the API logging page for more details.

void setServerLogLevel(int logLevel)

Parameter	Type	Description
logLevel	int	Specifies the level of log entry detail used by the server (TWS) when processing API requests. Valid values include: • 1 = SYSTEM • 2 = ERROR • 3 = WARNING • 4 = INFORMATION • 5 = DETAIL

4.1.6 reqCurrentTime()

Returns the current system time on the server side via the currentTime() EWrapper method.

void reqCurrentTime()

4.1.7 serverVersion()

Returns the version of the TWS instance to which the API application is connected.

void serverVersion()

4.1.8 TwsConnectionTime()

Returns the time the API application made a connection to TWS.

void TwsConnectionTime ()

4.2 Connection and Server EClientSocket Methods

4.2.1 currentTime()

This method receives the current system time on the server side.

void currentTime(long time)

Parameter	Type	Description
time	long	The current system time on the server side

4.2.2 error()

This method is called when there is an error with the communication or when TWS wants to send a message to the client.

void error(int id, int errorCode, String errorString)

Parameter	Type	Description
id	int	This is the orderId or tickerId of the request that generated the error.
errorCode	int	For information on error codes, see Error Codes.
errorString	String	The textual description of the error.

This method is called when an exception occurs while handling a request.

void error(Exception e)

Parameter	Type	Description
e	Exception	The exception that occurred

This method is called when TWS wants to send an error message to the client. (V1).

void error(String str)

Parameter	Type	Description
str	String	This is the textual description of the error

4.2.3 connectionClosed()

This method is called when TWS closes the sockets connection, or when TWS is shut down.

void connectionClosed()

Chapter 5

Executions Group

5.1 Execution EClientSocket Methods

5.1.1 reqExecutions()

When this method is called, the execution reports from the last 24 hours that meet the filter criteria are downloaded to the client via the execDetails() method. To view executions beyond the past 24 hours, open the Trade Log in TWS and, while the Trade Log is displayed, request the executions again from the API.

void reqExecutions(ExecutionFilter filter)

ExecutionFilter filter = (reqId, clientId, accountName, contract)

Parameter	Type	Description
filter	ExecutionFilter	The filter criteria used to determine which execution reports are returned.
reqId	Int	Id of the execution request. It is used for identification later when the data is returned to execDetails wrapper
clientId	Int	ClientId of the eConnection(Host, Port, ClientId).
accountName	Int	E.g., DU123456.
contract	contractDetails	For FX, minimal info required: m_symbol, m_secType, m_exchange

```
    print "Testing Executions \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", 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))

    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
```

5.2 Execution EWrapper Methods

5.2.1 execDetails()

This method is called when the reqExecutions() method is invoked, or when an order is filled.

void execDetails(int reqId, Contract contract, Execution execution)

Parameter	Type	Description
reqId	int	The reqID that was specified previously in the call to reqExecution().
contract	Contract	This structure contains a full description of the contract that was executed. Note: Refer to the Java SocketClient Properties page for more information.
execution	Execution	This structure contains addition order execution details.

```
callback.exec_Details_contract.__dict__
Out[34]:
{'m_conId': 12087792,
 'm_currency': 'USD',
 'm_exchange': 'IDEALPRO',
 'm_expiry': None,
 'm_includeExpired': False,
 'm_localSymbol': 'EUR.USD',
 'm_multiplier': None,
 'm_right': None,
 'm_secType': 'CASH',
 'm_strike': 0.0,
 'm_symbol': 'EUR',
 'm_tradingClass': 'EUR.USD'}
callback.exec_Details_execution.__dict__
Out[35]:
{'m_acctNumber': 'DU254946',
 'm_avgPrice': 1.0837,
 'm_clientId': 4000,
 'm_cumQty': 100000,
 'm_evMultiplier': 0,
 'm_evRule': None,
 'm_exchange': 'IDEALPRO',
 'm_execId': '0001f4e8.56ae9370.01.01',
 'm_liquidation': 0,
 'm_orderId': 10,
 'm_orderRef': None,
 'm_permId': 810462195,
 'm_price': 1.0837,
 'm_shares': 100000,
 'm_side': 'BOT',
 'm_time': '20160201  09:52:21'}
```

5.2.2 execDetailsEnd()

This method is called once all executions have been sent to a client in response to reqExecutions().

void execDetailsEnd(int reqId)

Parameter	Type	Description
reqID	int	The Id of the data request.

5.2.3 commissionReport()

The commissionReport() callback is triggered as follows:

- Immediately after a trade execution
- By calling reqExecutions().

void commissionReport(CommissionReport commissionReport)

Parameter	Type	Description
commissionReport	CommissionReport	The structure that contains commission details.

```
callback.commission_Report.__dict__
Out[38]:
{'m_commission': 2.1674,
 'm_currency': 'USD',
 'm_execId': '0001f4e8.56ae9370.01.01',
 'm_realizedPNL': 1.7976931348623157e+308,
 'm_yield': 1.7976931348623157e+308,
 'm_yieldRedemptionDate': 0}
```

Chapter 6

Contract Details Group

6.1 Contract Details EClientSocket Methods

6.1.1 reqContractDetails()

Call this method to download all details for a particular contract. The contract details will be received via the contractDetails() method on the EWrapper.

void reqContractDetails (int reqId, Contract contract)

Parameter	Type	Description
reqId	int	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contract	Contract	This class contains attributes used to describe the contract.

6.2 Contract Details EWrapper Methods

6.2.1 contractDetails()

This method is called only when reqContractDetails method on the EClientSocket object has been called.

void contractDetails(int ReqId, ContractDetails contractDetails)

Parameter	Type	Description
reqID	int	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contractDetails	ContractDetails	This structure contains a full description of the contract being looked up.

6.2.2 contractDetailsEnd()

This method is called once all contract details for a given request are received. This helps to define the end of an option chain.

void contractDetailsEnd(int reqId)

Parameter	Type	Description
reqID	int	The Id of the data request.

6.2.3 bondContractDetails()

This method is called only when reqContractDetails method on the EClientSocket object has been called for bonds.

void bondContractDetails(int reqId, ContractDetails contractDetails)

Parameter	Type	Description
reqId	int	The ID of the data request.
contractDetails	ContractDetails	This structure contains a full description of the bond contract being looked up.

Chapter 7

Market Depth Group

7.1 Market Depth EClientSocket Methods

7.1.1 reqMarketDepth()

Call this method to request market depth for a specific contract. The market depth will be returned by the updateMkt-Depth() and updateMktDepthL2() methods.

void reqMktDepth(int tickerId, Contract contract, int numRows, Vector mktDepthOptions)

Parameter	Type	Description
tickerId	int	The ticker Id. Must be a unique value. When the market depth data returns, it will be identified by this tag. This is also used when canceling the market depth.
contract	Contract	This class contains attributes used to describe the contract.
numRows	int	Specifies the number of market depth rows to return.
mktDepthOptions	Vector	For internal use only. Use default value XYZ.

7.1.2 cancelMarketDepth()

After calling this method, market depth data for the specified Id will stop flowing.

void cancelMktDepth(int TickerId)

Parameter	Type	Description
tickerId	int	The Id that was specified in the call to reqMktDepth().

7.2 Market Depth EWrapper Methods

7.2.1 updateMktDepth()

This method is called when the market depth changes.

void updateMktDepth(int tickerId, int position, int operation, int side, double price, int size)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktDepth()
position	int	Specifies the row Id of this market depth entry.
operation	int	Identifies how this order should be applied to the market depth. Valid values are: • 0 = insert (insert this new order into the row identified by 'position') • 1 = update (update the existing order in the row identified by 'position') • 2 = delete (delete the existing order at the row identified by 'position')
side	int	Identifies the side of the book that this order belongs to. Valid values are: • 0 = ask • 1 = bid
price	double	The order price.
size	int	The order size.

7.2.2 updateMktDepthL2()

This method is called when the Level II market depth changes.

void updateMktDepthL2(int tickerId, int position, String marketMaker, int operation, int side, double price, int size)

Parameter	Type	Description
tickerId	int	The ticker Id that was specified previously in the call to reqMktDepth()
position	int	Specifies the row id of this market depth entry.
marketMaker	String	Specifies the exchange hosting this order.
operation	int	Identifies the how this order should be applied to the market depth. Valid values are: • 0 = insert (insert this new order into the row identified by 'position') • 1 = update (update the existing order in the row identified by 'position') • 2 = delete (delete the existing order at the row identified by 'position')
side	int	Identifies the side of the book that this order belongs to. Valid values are: • 0 = ask • 1 = bid
price	double	The order price.
size	int	The order size.

Chapter 8

News Bulletins Group

8.1 News Bulletins EClientSocket Methods

8.1.1 reqNewsBulletins()

Call this method to start receiving news bulletins. Each bulletin will be returned by the updateNewsBulletin() method.

void reqNewsBulletins(boolean allMsgs)

Parameter	Type	Description
allMsgs	boolean	If set to TRUE, returns all the existing bulletins for the current day and any new ones. IF set to FALSE, will only return new bulletins.

8.1.2 cancelNewsBulletins()

Call this method to stop receiving news bulletins.

void cancelNewsBulletins()

8.2 News Bulletins EWrapper Methods

8.2.1 updateNewsBulletin()

This method is triggered for each new bulletin if the client has subscribed (i.e. by calling the reqNewsBulletins() method.

void updateNewsBulletin(int msgId, int msgType, String message, String origExchange)

Parameter	Type	Description
msgId	int	The bulletin ID, incrementing for each new bulletin.
msgType	int	Specifies the type of bulletin. Valid values include: • 1 = Regular news bulletin • 2 = Exchange no longer available for trading • 3 = Exchange is available for trading
message	String	The bulletin's message text.
origExchange	String	The exchange from which this message originated.

Chapter 9

Financial Advisors Group

9.1 Financial Advisors EClientSocket Methods

9.1.1 reqManagedAccts()

Call this method to request the list of managed accounts. The list will be returned by the managedAccounts() method on the EWrapper. Note: This request can only be made when connected to a Financial Advisor (FA) account

void reqManagedAccts()

9.1.2 requestFA()

Call this method to request FA configuration information from TWS. The data returns in an XML string via the receiveFA() method.

void requestFA(int faDataType)

Parameter	Type	Description
faDataType	int	Specifies the type of Financial Advisor configuration data being requested. Valid values include: • 1 = GROUPS • 2 = PROFILE • 3 = ACCOUNT ALIASES

9.1.3 replaceFa()

Call this method to request new FA configuration information from TWS. The data returns in an XML string via a ”receiveFA” method.

void replaceFA(int faDataType, String xml)

Parameter	Type	Description
faDataType	int	Specifies the type of Financial Advisor configuration data being requested. Valid values include: • 1 = GROUPS • 2 = PROFILE • 3 = ACCOUNT ALIASES
xml	String	The XML string containing the new FA configuration information.

9.2 Financial Advisors EWrapper Methods

9.2.1 ManagedAccounts()

This method is called when a successful connection is made to an account. It is also called when the reqManagedAccts() method is invoked.

void managedAccounts(String accountsList)

Parameter	Type	Description
accountsList	String	The comma delimited list of FA managed accounts.

9.2.2 receiveFA()

This method receives previously requested FA configuration information from TWS.

receiveFA(int faDataType, String xml)

Parameter	Type	Description
faDataType	int	Specifies the type of Financial Advisor configuration data being received from TWS. Valid values include: • 1 = GROUPS • 2 = PROFILE • 3 =ACCOUNT ALIASES
xml	String	The XML string containing the previously requested FA configuration information.

Chapter 10

Historical Data Group

10.1 Historical Data EClientSocket Methods

10.1.1 reqhistoricalData()

Call the reqHistoricalData() method to start receiving historical data results through the historicalData() EWrapper method.

void reqHistoricalData (int id, Contract contract, String endDateTime, String durationStr, String barSizeSetting, String whatToShow, int useRTH, int formatDate, List chartOptions)

Parameter	Type	Description
tickerId	int	The Id for the request. Must be a unique value. When the data is received, it will be identified by this Id. This is also used when canceling the historical data request.
contract	Contract	This class contains attributes used to describe the contract.
endDateTime	String	Use the format yyyymmdd hh:mm:ss tmz, where the time zone is allowed (optionally) after a space at the end.
durationStr	String	This is the time span the request will cover, and is specified using the format: <integer> <unit>, i.e., 1 D, where valid units are: • ” S (seconds) • ” D (days) • ” W (weeks) • ” M (months) • ” Y (years) If no unit is specified, seconds are used. Also, note “years” is currently limited to one.

Parameter	Type	Description
barSizeSetting	String	Specifies the size of the bars that will be returned (within IB/TWS limits). Valid bar size values include: • 1 sec • 5 secs • 15 secs • 30 secs • 1 min • 2 mins • 3 mins • 5 mins • 15 mins • 30 mins • 1 hour • 1 day
whatToShow	String	Determines the nature of data being extracted. Valid values include: • TRADES • MIDPOINT • BID • ASK • BID_ASK • HISTORICAL_VOLATILITY • OPTION_IMPLIED_VOLATILITY
useRTH	int	Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include: • 0 - all data is returned even where the market in question was outside of its regular trading hours. • 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.

Parameter	Type	Description
formatDate	int	Determines the date format applied to returned bars. Valid values include: • 1 - dates applying to bars returned in the format: yyyymmddspacespacehh:mm:dd • 2 - dates are returned as a long integer specifying the number of seconds since 1/1/1970 GMT.
chartOptions	List	For internal use only. Use default value XYZ.

10.1.2 cancelhistoricalData()

Call the cancelHistoricalData() method to stop receiving historical data results.

void cancelHistoricalData (int tickerId)

Parameter	Type	Description
tickerId	int	The Id that was specified in the call to reqHistoricalData().

10.2 Historical Data EWrapper Methods

10.2.1 historicalData()

This method receives the requested historical data results.

void historicalData (int reqId, String date, double open, double high, double low, double close, int volume, int count, double WAP, boolean hasGaps)

Parameter	Type	Description
reqId	int	The ticker Id of the request to which this bar is responding.
date	String	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	double	The bar opening price.
high	double	The high price during the time covered by the bar.
low	double	The low price during the time covered by the bar.
close	double	The bar closing price.
volume	int	The volume during the time covered by the bar.
count	int	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers
WAP	double	The weighted average price during the time covered by the bar.
hasGaps	boolean	The weighted average price during the time covered by the bar.

Chapter 11

Market Scanners Group

11.1 Market Scanners EClientSocket Methods

11.1.1 reqScannerParameters()

Call the reqScannerParameters() method to receive an XML document that describes the valid parameters that a scanner subscription can have.

void reqScannerParameters()

11.1.2 reqScannerSubscription()

Call the reqScannerSubscription() method to start receiving market scanner results through the scannerData() EWrapper method.

void reqScannerSubscription(int tickerId, ScannerSubscription subscription, Vector scannerSubscriptionOptions)

Parameter	Type	Description
tickerId	int	The Id for the subscription. Must be a unique value. When the subscription data is received, it will be identified by this Id. This is also used when canceling the scanner.
subscription	ScannerSubscription	Summary of the scanner subscription parameters including filters.
scannerSubscriptionOptions	Vector	For internal use only. Use default value XYZ.

11.1.3 cancelScannerSubscription()

Call the cancelScannerSubscription() method to stop receiving market scanner results. **void cancelScannerSubscription(int tickerId)**

Parameter	Description
tickerId	The Id that was specified in the call to reqScannerSubscription().

11.2 Market Scanners EWrapper Methods

11.2.1 scannerParameters()

This method receives an XML document that describes the valid parameters that a scanner subscription can have.

void scannerParameters(String xml)

Parameter	Type	Description
xml	String	A document describing available scanner subscription parameters.

11.2.2 scannerData()

This method receives the requested market scanner data results.

void scannerData(int reqId, int rank, ContractDetails contractDetails, String distance, String benchmark, String projection, String legsStr)

Parameter	Type	Description
reqId	int	The ID of the request to which this row is responding.
rank	int	The ranking within the response of this bar.
contractDetails	ContractDetails	This structure contains a full description of the contract that was executed.
distance	String	Varies based on query.
benchmark	String	Varies based on query.
projection	String	Varies based on query.
legsStr	String	Describes combo legs when scan is returning EFP.

11.2.3 scannerDataEnd()

This method is called when the snapshot is received and marks the end of one scan.

void scannerDataEnd(int reqId)

Parameter	Type	Description
reqId	int	The ID of the market data snapshot request being closed by this parameter.

Chapter 12

Real Time Bars Group

12.1 Real Time Bars EClientSocket Methods

12.1.1 reqRealTimeBars()

Call the reqRealTimeBars() method to start receiving real time bar results through the realtimeBar() EWrapper method.

void reqRealTimeBars(int tickerId, Contract contract, int barSize, String whatToShow, boolean useRTH, Vector realTimeBarOptions)

Parameter	Type	Description
tickerId	int	The Id for the request. Must be a unique value. When the data is received, it will be identified by this Id. This is also used when canceling the historical data request.
contract	Contract	This class contains attributes used to describe the contract.
barSize	int	Currently only 5 second bars are supported, if any other value is used, an exception will be thrown.
whatToShow	String	Determines the nature of the data extracted. Valid values include: • TRADES • BID • ASK • MIDPOINT
useRTH	boolean	Regular Trading Hours only. Valid values include: • 0 = all data available during the time span requested is returned, including time intervals when the market in question was outside of regular trading hours. • 1 = only data within the regular trading hours for the product requested is returned, even if the time time span falls partially or completely outside.
realTimeBarOptions	Vector	For internal use only. Use default value XYZ.

12.1.2 cancelRealTimeBars()

Call this method to stop receiving real time bar results.

void cancelRealTimeBars (int tickerId)

Parameter	Type	Description
tickerId	int	The Id that was specified in the call to reqRealTimeBars().

12.2 Real Time Bars EWrapper Methods

12.2.1 realtimeBar()

This method receives the real-time bars data results.

void realtimeBar(int reqId, long time, double open, double high, double low, double close, long volume, double wap, int count)

Parameter	Type	Description
reqId	int	The ticker ID of the request to which this bar is responding
time	long	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	double	The bar opening price.
high	double	The high price during the time covered by the bar.
low	double	The low price during the time covered by the bar.
close	double	The bar closing price.
volume	long	The volume during the time covered by the bar.
wap	double	The weighted average price during the time covered by the bar.
count	int	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers.

Chapter 13

Fundamental Data Group

13.1 Fundamental Data EClientSocket Methods

13.1.1 reqFundamentalData()

Call this method to receive Reuters global fundamental data for stocks. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data. reqFundamentalData() can handle conid specified in the Contract object, but not tradingClass or multiplier. This is because reqFundamentalData() is used only for stocks and stocks do not have a multiplier and trading class.
void reqFundamentalData(int reqId, Contract contract, String reportType)

Parameter	Type	Description
reqId	int	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contract	Contract	This structure contains a description of the contract for which Reuters Fundamental data is being requested.
reportType	String	One of the following XML reports: • ReportSnapshot (company overview) • ReportsFinSummary (financial summary) • ReportRatios (financial ratios) • ReportsFinStatements (financial statements) • RESC (analyst estimates) • CalendarReport (company calendar)

13.1.2 cancelFundamentalData()

Call this method to stop receiving Reuters global fundamental data.
void cancelFundamentalData(int reqId)

Parameter	Type	Description
reqId	int	The ID of the data request.

13.2 Fundamental Data EWrapper Methods

This method is called to receive Reuters global fundamental market data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data. **void fundamentalData(int reqId, String data)**

Parameter	Type	Description
reqId	int	The ID of the data request.
data	String	One of these XML reports: • Company overview • Financial summary • Financial ratios • Financial statements • Analyst estimates • Company calendar

Chapter 14

Display Groups

14.1 Display Groups EClientSocket Methods

14.1.1 queryDisplayGroups()

queryDisplayGroups(int reqId)

Parameter	Type	Description
reqId	int	The unique number that will be associated with the response

14.1.2 subscribeToGroupEvents()

subscribeToGroupEvents(int reqId, int groupId)

Parameter	Type	Description
reqId	int	The unique number associated with the notification.
groupId	int	The ID of the group, currently it is a number from 1 to 7. This is the display group subscription request sent by the API to TWS.

14.1.3 updateDisplayGroup()

updateDisplayGroup(int reqId, const String contractInfo)

Parameter	Type	Description
reqId	int	The unique number associated with the notification.
contractInfo	String	The encoded value that uniquely represents the contract in IB. Possible values include: • none = empty selection • contractID@exchange any non-combination contract. Examples: 8314@SMART for IBM SMART; 8314@ARCA for IBM@ARCA. • combo = if any combo is selected.

14.1.4 unsubscribeFromGroupEvents()

unsubscribeFromGroupEvents(int reqId)

Parameter	Type	Description
reqId	int	The requestId specified in subscribeToGroupEvents().

14.2 Display Groups EWrapper Methods

14.2.1 displayGroupList()

This callback is a one-time response to queryDisplayGroups().

displayGroupList(int reqId As Integer, String groups)

Parameter	Type	Description
reqtId	int	The requestId specified in queryDisplayGroups().
groups	String	A list of integers representing visible group ID separated by the | character, and sorted by most used group first. This list will not change during TWS session (in other words, user cannot add a new group; sorting can change though). Example: 3|1|2

14.2.2 displayGroupUpdated()

This is sent by TWS to the API client once after receiving the subscription request subscribeToGroupEvents(), and will be sent again if the selected contract in the subscribed display group has changed.

displayGroupUpdated(int reqId, String contractInfo)

Parameter	Type	Description
requestId	int	The requestId specified in subscribeToGroupEvents().
contractInfo	String	The encoded value that uniquely represents the contract in IB. Possible values include: • none = empty selection • contractID@exchange any non-combination contract. Examples: 8314@SMART for IBM SMART; 8314@ARCA for IBM @ARCA. • combo = if any combo is selected.

Chapter 15

Java SocketClient Properties

15.1 Execution Properties

15.1.1 Execution

Attribute	Type	Description
m_acctNumber	String	The customer account number.
m_avgPrice	double	Average price. Used in regular trades, combo trades and legs of the combo. Does not include commissions.
m_clientId	int	The id of the client that placed the order. Note: TWS orders have a fixed client id of ”0.”
m_cumQty	int	Cumulative quantity. Used in regular trades, combo trades and legs of the combo.
m_exchange	String	Exchange that executed the order.
m_execId	String	Unique order execution id.
m_liquidation	int	Identifies the position as one to be liquidated last should the need arise.
m_orderId	int	The order id. Note: TWS orders have a fixed order id of ”0.”
m_permId	int	The TWS id used to identify orders, remains the same over TWS sessions.
m_price	double	The order execution price, not including commissions.
m_shares	int	The number of shares filled.
m_side	String	Specifies if the transaction was a sale or a purchase. Valid values are: • BOT • SLD
m_time	String	The order execution time.
m_evRule	String	Contains the Economic Value Rule name and the respective optional argument. The two values should be separated by a colon. For example, aussieBond: YearsToExpiration=3. When the optional argument is not present, the first value will be followed by a colon.
m_evMultiplier	double	Tells you approximately how much the market value of a contract would change if the price were to change by 1. It cannot be used to get market value by multiplying the price by the approximate multiplier.

15.1.2 ExecutionFilter

Attribute	Type	Description
m_acctCode	String	Filter the results of the reqExecutions() method based on an account code. Note: this is only relevant for Financial Advisor (FA) accounts.
m_clientId	int	Filter the results of the reqExecutions() method based on the clientId.
m_exchange	String	Filter the results of the reqExecutions() method based on the order exchange.
m_secType	String	Filter the results of the reqExecutions() method based on the order security type. Note: Refer to the Contract struct for the list of valid security types.
m_side	String	Filter the results of the reqExecutions() method based on the order action. Note: Refer to the Order class for the list of valid order actions.
m_symbol	String	Filter the results of the reqExecutions() method based on the order symbol.
m_time	String	Filter the results of the reqExecutions() method based on execution reports received after the specified time. The format for timeFilter is ”yyyymmdd-hh:mm:ss”

15.2 CommissionReport

Attribute	Type	Description
m_commission()	double	The commission amount.
m_currency()	String	The currency.
m_execId()	String	Unique order execution id.
m_realizedPNL()	double	The amount of realized Profit and Loss.
m_yield()	double	The yield.
m_yieldRedemptionDate()	int	Takes the YYYYMMDD format.

15.3 Contract Properties

15.3.1 Contract

Attribute	Type	Description
m_comboLegs	Vector	Dynamic memory structure used to store the leg definitions for this contract.
m_comboLegsDescrip	String	Description for combo legs
m_conId	int	The unique contract identifier.
m_currency	String	Specifies the currency. Ambiguities may require that this field be specified, for example, when SMART is the exchange and IBM is being requested (IBM can trade in GBP or USD). Given the existence of this kind of ambiguity, it is a good idea to always specify the currency.
m_exchange	String	The order destination, such as Smart.
m_expiry	String	The expiration date. Use the format YYYYMM.
m_includeExpired	boolean	If set to true, contract details requests and historical data queries can be performed pertaining to expired contracts. Note: Historical data queries on expired contracts are limited to the last year of the contracts life, and are initially only supported for expired futures contracts.
m_localSymbol	String	This is the local exchange symbol of the underlying asset.
m_multiplier	String	Allows you to specify a future or option contract multiplier. This is only necessary when multiple possibilities exist.
m_primaryExch	String	Identifies the listing exchange for the contract (do not list SMART).
m_right	String	Specifies a Put or Call. Valid values are: P, PUT, C, CALL.
m_secId	String	Unique identifier for the secIdType.
m_secIdType	String	Security identifier, when querying contract details or when placing orders. Supported identifiers are: • SIN (Example: Apple: US0378331005) • CUSIP (Example: Apple: 037833100) • SEDOL (Consists of 6-AN + check digit. Example: BAE: 0263494) • RIC (Consists of exchange-independent RIC Root and a suffix identifying the exchange. Example: AAPL.O for Apple on NASDAQ.)
m_secType	String	This is the security type. Valid values are: • STK. Stocks • OPT. Options • FUT. Futures • IND. Index • FOP. Futures Options • CASH. FX • BAG • NEWS
m_strike	double	The strike price.
m_symbol	String	This is the symbol of the underlying asset.
m_tradingClass	String	The trading class name for this contract.

15.3.2 ContractDetails

Attribute	Type	Description
m_category	String	The industry category of the underlying. For example, InvestmentSvc.
m_contractMonth	String	The contract month. Typically the contract month of the underlying for a futures contract.
m_industry	String	The industry classification of the underlying/product. For example, Financial.
m_liquidHours	String	The regular trading hours of the product. For example, 20090507:0930-1600;20090508:CLOSED.
m_longName	String	Descriptive name of the asset.
m_marketName	String	The market name for this contract.
m_minTick	double	The minimum price tick.
m_orderTypes	String	The list of valid order types for this contract.
m_priceMagnifier	String	Allows execution and strike prices to be reported consistently with market data, historical data and the order price, i.e. Z on LIFFE is reported in index points and not GBP.
m_secIdList()	Vector	A list of contract identifiers that the customer is allowed to view (CUSIP, ISIN, etc.)
m_subcategory	String	The industry subcategory of the underlying. For example, Brokerage.
m_summary	Contract	A contract summary.
m_timeZoneId	String	The ID of the time zone for the trading hours of the product. For example, EST.
m_tradingHours	String	The total trading hours of the product. For example, 20090507:0700-1830,1830-2330;20090508:CLOSED.
m_underConId	String	The underlying contract ID.
m_validExchanges	String	The list of exchanges this contract is traded on.
m_evRule	String	Contains the Economic Value Rule name and the respective optional argument. The two values should be separated by a colon. For example, aussieBond:YearsToExpiration=3. When the optional argument is not present, the first value will be followed by a colon.
m_evMultiplier	double	Tells you approximately how much the market value of a contract would change if the price were to change by 1. It cannot be used to get market value by multiplying the price by the approximate multiplier.

Bond Values

Attribute	Type	Description
m_bondType	String	For Bonds. The type of bond, such as ”CORP.”
m_callable	boolean	For Bonds. Values are True or False. If true, the bond can be called by the issuer under certain conditions.
m_convertible	boolean	For Bonds. Values are True or False. If true, the bond can be converted to stock under certain conditions.
m_coupon	double	For Bonds. The interest rate used to calculate the amount you will receive in interest payments over the course of the year.
m_couponType	String	For Bonds. The type of bond coupon.
m_cusip	String	For Bonds. The nine-character bond CUSIP or the 12-character SEDOL.
m_descAppend	String	For Bonds. A description string containing further descriptive information about the bond.
m_issueDate	String	For Bonds. The date the bond was issued.
m_maturity	String	For Bonds. The date on which the issuer must repay the face value of the bond.
m_nextOptionDate	String	For Bonds, only if bond has embedded options.
m_nextOptionPartial	boolean	For Bonds, only if bond has embedded options.
m_putable	boolean	For Bonds. Values are True or False. If true, the bond can be sold back to the issuer under certain conditions.
m_ratings	String	For Bonds. Identifies the credit rating of the issuer. A higher credit rating generally indicates a less risky investment. Bond ratings are from Moody’s and S&P respectively.
m_nextOptionType	String	For Bonds, only if bond has embedded options.
m_notes	String	For Bonds, if populated for the bond in IB’s database

15.4 Orders

15.4.1 ComboLeg

Attribute	Type	Description
m_action	String	The side (buy or sell) for the leg you are constructing.
m_conId	int	The unique contract identifier specifying the security.
m_designatedLocation	String	If shortSaleSlot == 2, the designatedLocation must be specified. Otherwise leave blank or orders will be rejected.
m_exchange	String	The exchange to which the complete combination order will be routed.
m_openClose	int	Specifies whether the order is an open or close order. Valid values are: • 0 - Same as the parent security. This is the only option for retail customers. • 1 - Open. This value is only valid for institutional customers. • 2 - Close. This value is only valid for institutional customers. • 3 - Unknown
m_ratio	int	Select the relative number of contracts for the leg you are constructing. To help determine the ratio for a specific combination order, refer to the Interactive Analytics section of the User's Guide.
m_shortSaleSlot	int	For institutional customers only. • 0 - inapplicable (i.e. retail customer or not short leg) • 1 - clearing broker • 2 - third party. If this value is used, you must enter a designated location.

15.4.2 OrderComboLeg

Attribute	Type	Description
m_price	double	Order-specific leg price.

15.4.3 Order

Order Identifiers		
Attribute	**Type**	**Description**
m_clientId	int	The id of the client that placed this order.
m_orderId	int	The id for this order.
m_permid	int	The TWS id used to identify orders, remains the same over TWS sessions.

Main Order Fields		
Attribute	**Type**	**Description**
m_action	String	Identifies the side. Valid values are: BUY, SELL, SHORT
m_auxPrice	double	This is the STOP price for stop-limit orders, and the offset amount for relative orders. In all other cases, specify zero.
m_lmtPrice	double	This is the LIMIT price, used for limit, stop-limit and relative orders. In all other cases specify zero. For relative orders with no limit price, also specify zero.
m_orderType	String	Identifies the order type. For more information about supported order types, see Supported Order Types.
m_totalQuantity	long	The order quantity.
Extended Order Fields		
Attribute	**Type**	**Description**
m_allOrNone	boolean	0 = no, 1 = yes
m_blockOrder	boolean	If set to true, specifies that the order is an ISE Block order.
m_displaySize	int	The publicly disclosed order size, used when placing Iceberg orders.
m_goodAfterTime	String	The trade's "Good After Time," format "YYYYMMDD hh:mm:ss (optional time zone)". Use an empty String if not applicable.
m_goodTillDate	String	You must enter GTD as the time in force to use this string. The trade's "Good Till Date," format "YYYYMMDD hh:mm: ss (optional time zone)". Use an empty String if not applicable.
hidden	boolean	If set to true, the order will not be visible when viewing the market depth. This option only applies to orders routed to the ISLAND exchange.
m_minQty	int	Identifies a minimum quantity order type.
m_ocaGroup	String	Identifies an OCA (one cancels all) group.
m_ocaType	int	Tells how to handle remaining orders in an OCA group when one order or part of an order executes. Valid values include: • 1 = Cancel all remaining orders with block • 2 = Remaining orders are proportionately reduced in size with block • 3 = Remaining orders are proportionately reduced in size with no block If you use a value "with block" gives your order has overfill protection. This means that only one order in the group will be routed at a time to remove the possibility of an overfill.

Attribute	Type	Description
m_orderRef	String	The order reference. Intended for institutional customers only, although all customers may use it to identify the API client that sent the order when multiple API clients are running.
m_outsideRth	boolean	If set to true, allows orders to also trigger or fill outside of regular trading hours.
m_parentId	int	The order ID of the parent order, used for bracket and auto trailing stop orders.
m_percentOffset	double	The percent offset amount for relative orders.
overridePercentageConstraints	boolean	Precautionary constraints are defined on the TWS Presets page, and help ensure that your price and size order values are reasonable. Orders sent from the API are also validated against these safety constraints, and may be rejected if any constraint is violated. To override validation, set this parameters value to True. Valid values include: • 0 = False • 1 = True
m_rule80A	string	Values include: • Individual = 'I' • Agency = 'A', • AgentOtherMember = 'W' • IndividualPTIA = 'J' • AgencyPTIA = 'U' • AgentOtherMemberPTIA = 'M' • IndividualPT = 'K' • AgencyPT = 'Y' • AgentOtherMemberPT = 'N'
m_sweepToFill	boolean	If set to true, specifies that the order is a Sweep-to-Fill order.
m_tif	String	The time in force. Valid values are: DAY, GTC, IOC, GTD.
m_transmit	boolean	Specifies whether the order will be transmitted by TWS. If set to false, the order will be created at TWS but will not be sent.

Attribute	Type	Description
m_triggerMethod	int	Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are: • 0 - The default value. The ”double bid/ask” function will be used for orders for OTC stocks and US options. All other orders will used the ”last” function. • 1 - use ”double bid/ask” function, where stop orders are triggered based on two consecutive bid or ask prices. • 2 - ”last” function, where stop orders are triggered based on the last price. • 3 - double last function. • 4 - bid/ask function. • 7 - last or bid/ask function. • 8 - mid-point function.
m_trailStopPrice	double	For TRAIL LIMIT orders only
m_trailingPercent	double	Specify the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field: • This field is mutually exclusive with the existing trailing amount. That is, the API client can send one or the other but not both. • This field is read AFTER the stop price (barrier price) as follows: – deltaNeutralAuxPrice – stopPrice – trailingPercent – scale order attributes • The field will also be sent to the API in the openOrder message if the API client version is $>=$ 56. It is sent after the stopPrice field as follows: – stopPrice – trailingPct – basisPoint
m_activeStartTime	String	For GTC orders.
m_activeStopTime	String	For GTC orders.
Financial Advisor Fields		
Attribute	**Type**	**Description**
m_faGroup	String	The Financial Advisor group the trade will be allocated to – use an empty String if not applicable.
m_faMethod	String	The Financial Advisor allocation function the trade will be allocated with – use an empty String if not applicable.
m_faPercentage	String	The Financial Advisor percentage concerning the trade’s allocation – use an empty String if not applicable.
m_faProfile	String	The Financial Advisor allocation profile the trade will be allocated to – use an empty String if not applicable.

Institutional (non-cleared) Only		
Attribute	**Type**	**Description**
m_designatedLocation	String	Used only when shortSaleSlot = 2.
m_openClose	String	For institutional customers only. Valid values are O, C.
m_origin	int	The order origin. For institutional customers only. Valid values are 0 = customer, 1 = firm
m_shortSaleSlot	int	Valid values are 1 or 2.
SMART Routing Only		
Attribute	**Type**	**Description**
m_discretionaryAmt	double	The amount off the limit price allowed for discretionary orders.
m_eTradeOnly	boolean	Trade with electronic quotes. 0 = no, 1 = yes
m_firmQuoteOnly	boolean	Trade with firm quotes. 0 = no, 1 = yes
m_nbboPriceCap	double	Maximum smart order distance from the NBBO.
m_optOutSmartRouting	boolean	Use to opt out of default SmartRouting for orders routed directly to ASX. This attribute defaults to false unless explicitly set to true. When set to false, orders routed directly to ASX will NOT use SmartRouting. When set to true, orders routed directly to ASX orders WILL use SmartRouting.
BOX or VOL Orders Only		
Attribute	**Type**	**Description**
m_auctionStrategy	int	Values include: • match = 1 • improvement = 2 • transparent = 3 For orders on BOX only.
BOX Exchange Orders Only		
Attribute	**Type**	**Description**
m_delta	double	The stock delta. For orders on BOX only.
m_startingPrice	double	The auction starting price. For orders on BOX only.
m_stockRefPrice	double	The stock reference price. The reference price is used for VOL orders to compute the limit price sent to an exchange (whether or not Continuous Update is selected), and for price range monitoring.

Pegged-to-Stock and VOL Orders Only		
Attribute	**Type**	**Description**
m_stockRangeLower	double	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
m_stockRangeUpper	double	The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
Volatility Orders Only		
Attribute	**Type**	**Description**
m_continuousUpdate	boolean	VOL orders only. Specifies whether TWS will automatically update the limit price of the order as the underlying price moves.
m_deltaNeutralOrderType	String	VOL orders only. Enter an order type to instruct TWS to submit a delta neutral trade on full or partial execution of the VOL order. For no hedge delta order to be sent, specify NONE.
m_deltaNeutralAuxPrice	int	VOL orders only. Use this field to enter a value if the value in the *deltaNeutralOrderType* field is an order type that requires an Aux price, such as a REL order.
m_referencePriceType	int	VOL orders only. Specifies how you want TWS to calculate the limit price for options, and for stock range price monitoring. Valid values include: • 1 = Average of NBBO • 2 = NBB or the NBO depending on the action and right.
m_volatility	double	The option price in volatility, as calculated by TWS' Option Analytics. This value is expressed as a percent and is used to calculate the limit price sent to the exchange.
m_volatilityType	int	Values include: • 1 = Daily volatility • 2 = Annual volatility
m_deltaNeutralOpenClose	String	Specifies whether the order is an Open or a Close order and is used when the hedge involves a CFD and the order is clearing away.
m_deltaNeutralShortSale	boolean	Used when the hedge involves a stock and indicates whether or not it is sold short.
m_deltaNeutralShortSaleSlot	int	Has a value of 1 (the clearing broker holds shares) or 2 (delivered from a third party). If you use 2, then you must specify a deltaNeutralDesignatedLocation.
m_deltaNeutralDesignatedLocation	String	Used only when deltaNeutralShortSaleSlot = 2.

Combo Orders Only		
Attribute	**Type**	**Description**
m_basisPoints	double	For EFP orders only
m_basisPointsType	int	For EFP orders only
Scale Orders Only		
Attribute	**Type**	**Description**
m_scaleAutoReset()	boolean	For extended Scale orders.
m_scaleInitFillQty()	int	For extended Scale orders.
m_scaleInitLevelSize	int	For Scale orders: Defines the size of the first, or initial, order component.
m_scaleInitPosition()	int	For extended Scale orders.
m_scalePriceAdjustInterval()	int	For extended Scale orders.
m_scalePriceAdjustValue()	double	For extended Scale orders.
m_scalePriceIncrement	double	For Scale orders: Defines the price increment between scale components. This field is required.
m_scaleProfitOffset()	double	For extended Scale orders.
m_scaleRandomPercent()	boolean	For extended Scale orders.
m_scaleSubsLevelSize	int	For Scale orders: Defines the order size of the subsequent scale order components. Used in conjunction with scaleInitLevelSize().
m_scaleTable	String	Manual table for Scale orders.
Hedge Orders Only		
Attribute	**Type**	**Description**
m_hedgeParam	String	Beta = x for Beta hedge orders, ratio = y for Pair hedge order
m_hedgeType	String	For hedge orders. Possible values are: • D = Delta • B = Beta • F = FX • P = Pair

Clearing Information		
Attribute	**Type**	**Description**
m_account	String	The account. For institutional customers only.
m_clearingAccount	String	For IBExecution customers: Specifies the true beneficiary of the order. This value is required for FUT/FOP orders for reporting to the exchange.
m_clearingIntent	String	For IBExecution customers: Valid values are: IB, Away, and PTA (post trade allocation).
m_settlingFirm	String	Institutional only.
Algo Orders Only		
Attribute	**Type**	**Description**
m_algoStrategy	String	For information about API Algo orders, see IBAlgo Parameters.
m_algoParams	Vector	Support for IBAlgo parameters.
m_algoId	String	Identifies an order generated by algorithmic trading.
Solicited Orders		
Attribute	**Type**	**Description**
m_solicited	boolean	True = solicited (orders initiated by a broker through the brokers research and design) False = unsolicited (those instigated by a broker's customer either through their actions or by the broker at their direction)
What If		
Attribute	**Type**	**Description**
m_whatIf	boolean	Use to request pre-trade commissions and margin information. If set to true, margin and commissions data is received back via the OrderState() object for the openOrder() callback.
Smart Combo Routing		
Attribute	**Type**	**Description**
m_smartComboRoutingParams	Vector	Support for Smart Combo Routing.
Order Combo Legs		
Attribute	**Type**	**Description**
OrderComboLegs()	Object	Holds attributes for all legs in a combo order.
Not Held		
Attribute	**Type**	**Description**
m_notHeld	boolean	For IBDARK orders only. Orders routed to IBDARK are tagged as post only and are held in IB's order book, where incoming SmartRouted orders from other IB customers are eligible to trade against them.
Internal use only		
Attribute	**Type**	**Description**
m_orderMiscOptions	Vector	For internal use only. Use the default value XYZ.

15.4.4 OrderState

Attribute	Type	Description
m_commission	double	Shows the commission amount on the order.
m_commissionCurrency	String	Shows the currency of the commission value.
m_equityWithLoan	String	Shows the impact the order would have on your equity with loan value.
m_initMargin	String	Shows the impact the order would have on your initial margin.
m_maintMargin	String	Shows the impact the order would have on your maintenance margin.
m_maxCommission	double	Used in conjunction with the minCommission field, this defines the highest end of the possible range into which the actual order commission will fall.
m_minCommission	double	Used in conjunction with the maxCommission field, this defines the lowest end of the possible range into which the actual order commission will fall.
m_status	String	Displays the order status.
m_warningText	String	Displays a warning message if warranted.

15.5 ScannerSubscription

Attribute	Type	Description
m_abovePrice	double	Filter out contracts with a price lower than this value. Can be left blank.
m_aboveVolume	int	Filter out contracts with a volume lower than this value. Can be left blank.
m_averageOptionVolumeAbove	int	Can leave empty.
m_belowPrice	double	Filter out contracts with a price higher than this value. Can be left blank.
m_couponRateAbove	double	Filter out contracts with a coupon rate lower than this value. Can be left blank.
m_couponRateBelow	double	Filter out contracts with a coupon rate higher than this value. Can be left blank.

Attribute	Type	Description
m_excludeConvertible	String	Filter out convertible bonds. Can be left blank.
m_instrument	String	Defines the instrument type for the scan.
m_locationCode	String	The location.
m_maturityDateAbove	String	Filter out contracts with a maturity date earlier than this value. Can be left blank.
m_maturityDateBelow	String	Filter out contracts with a maturity date later than this value. Can be left blank.
m_marketCapAbove	double	Filter out contracts with a market cap lower than this value. Can be left blank.
m_marketCapBelow	double	Filter out contracts with a market cap above this value. Can be left blank.
m_moodyRatingAbove	String	Filter out contracts with a Moody rating below this value. Can be left blank.
m_moodyRatingBelow	String	Filter out contracts with a Moody rating above this value. Can be left blank.
m_numberOfRows	int	Defines the number of rows of data to return for a query.
m_scanCode	String	Can be left blank.
m_scannerSettingPairs	String	Can leave empty. For example, a pairing ”Annual, true” used on the ”top Option Implied Vol % Gainers” scan would return annualized volatilities.
m_spRatingAbove	String	Filter out contracts with an S&P rating below this value. Can be left blank.
m_spRatingBelow	String	Filter out contracts with an S&P rating above this value. Can be left blank.
m_stockTypeFilter	String	Valid values are: • CORP = Corporation • ADR = American Depositary Receipt • ETF = Exchange Traded Fund • REIT = Real Estate Investment Trust • CEF = Closed End Fund

15.6 UnderComp

Attribute	Type	Description
m_conId	int	The unique contract identifier specifying the security. Used for Delta-Neutral Combo contracts.
m_delta	double	The underlying stock or future delta. Used for Delta-Neutral Combo contracts.
m_price	double	The price of the underlying. Used for Delta-Neutral Combo contracts.

Chapter 16

Placing a Combination Order

A combination order is a special type of order that is constructed of many separate legs but executed as a single transaction. Submit combo orders such as calendar spreads, conversions and straddles using the **BAG** security type (defined in the Contract object). The key to implementing a successful API combination order using the API is to knowing how to place the same order using Trader Workstation. If you are familiar with placing combination orders in TWS, then it will be easier to place the same order using the API, because the API only imitates the behavior of TWS.

Example

In this example, a customer places a BUY order on a calendar spread for GOOG. To buy one calendar spread means:

Leg 1: Sell 1 GOOG OPT SEP 18 '09 150.0 CALL (100)

Leg 2: Buy 1 GOOG OPT JAN 21 '11 150.0 CALL (100)

Here is a summary of the steps required to place a combo order using the API:

- Obtain the contract id (conId) for each leg. Get this number by invoking the reqContractDetails() method.
- Include each leg on the ComboLeg object by populating the related fields.
- Implement the placeOrder() method with the Contract and Order socket client properties.

To place this combo order:

1. Get the Contract IDs for both leg definitions:

```
//First leg
Contract con1 = new Contract();
con1.m_symbol = "GOOG";
con1.m_secType = "OPT";
con1.m_expiry =   200909  ;
con1.m_strike = 150.0
con1.m_right =    C
con1.m_multiplier =    100
con1.m_exchange = "SMART  ;
con1.m_currency = "USD";
.reqContractDetails(1, con1);
//Second leg
Contract con2 = new Contract();
con2.m_symbol = "GOOG";
con2.m_secType = "OPT";
con2.m_expiry =   201101  ;
```

```
con2.m_strike = 150.0
con2.m_right =    C
con2.m_multiplier =    1 0 0
con2.m_exchange = "S M A R T  ;
con2.m_currency = "USD";
.reqContractDetails(2, con2);
//All conId numbers are delivered by the ContractDetail()
static public String contractDetails(int reqId, ContractDetails contractDetails)
{
Contract contract = contractDetails.m_summary;
/*Base on the request above,
reqId = 1 is corresponding to the first request or first leg
reqId = 2 is corresponding to the second request or second leg*/
if (reqId == 1)
{ Leg1_conId = contract.m_conId;} // to obtain conId for first leg
if (reqId == 2)
{ Leg2_conId = contract.m_conId;} // to obtain conId for second leg
}
```

2. Once the program has acquired the conId value for each leg, include it in the ComboLeg object:

```
ComboLeg leg1 = new ComboLeg(); // for the first leg
ComboLeg leg2 = new ComboLeg(); // for the second leg
Vector addAllLegs = new Vector();
leg1.m_conId = Leg1_conId;
leg1.m_ratio = 1;
leg1.m_action = "SELL";
leg1.m_exchange = "SMART";
leg1.m_openClose = 0;
leg1.m_shortSaleSlot = 0;
leg1.m_designatedLocation = "";
leg2.m_conId = Leg2_conId;
leg2.m_ratio = 1;
leg2.m_action = "BUY";
leg2.m_exchange = "SMART";
leg2.m_openClose = 0;
leg2.m_shortSaleSlot = 0;
leg2.m_designatedLocation = "";
addAllLegs.add(leg1);
addAllLegs.add(leg2);
```

3. Invoke the placeOrder() method with the appropriate contract and order objects:

```
Order order = new Order();
contract.m_symbol = "USD"; // For combo order use    U S D    as the symbol value
all the time
contract.m_secType = "BAG"; // BAG is the security type for COMBO order
contract.m_exchange = "SMART";
contract.m_currency = "USD";
contract.m_comboLegs = addAllLegs; //including combo order in contract object
order.m_action =    B U Y   ;
order.m_totalQuantity = 1;
order.m_orderType =    M K T
.placeOrder(OrderId, contract, order);
```

Note: For more information on combination orders, see the TWS Users Guide topic About Combination Orders[1].

[1] https://individuals.interactivebrokers.com/en/software/tws/usersguidebook/specializedorderentry/about_combination_orders.htm

Chapter 17

Java Code Samples: Contract Parameters

This section includes the following Java code samples:

- How to Determine an Option Contract
- How to Determine a Futures Contract
- How to Determine a Stock

17.1 How to Determine an Option Contract

Example 1 - Standard Method of Determining an Option Contract

```
void onHowToDetermineOption(){
        Contract contract = new Contract();
        Order order = new Order();
        contract.m_symbol = "IBKR";
        contract.m_secType = "OPT";
        contract.m_expiry = "20120316";
        contract.m_strike = 20.0;
        contract.m_right = "P";
                contract.m_multiplier = "100";
        contract.m_exchange = "SMART";
        contract.m_currency = "USD";
        order.m_action = "BUY";
        order.m_totalQuantity = 1;
        order.m_orderType = "LMT";
        order.m_lmtPrice = enteredLmtPrice;
        m_client.placeOrder(GlobalOrderId, contract, order);
}
```

Example 2 - Determining an Option Contract Using OCC Option Symbology Initiative

```
void inUsingOptionSymbologyInitiative(){
        Contract contract = new Contract();
        Order order = new Order();
        contract.m_localSymbol = "IBKR 120317P00020000"; //OSI
        contract.m_secType = "OPT";
        contract.m_exchange = "SMART";
        contract.m_currency = "USD";
        order.m_action = "BUY";
        order.m_totalQuantity = 1;
        order.m_orderType = "LMT";
```

```
        order.m_lmtPrice = enteredLmtPrice;
        m_client.placeOrder(GlobalOrderId, contract, order);
}
```

17.2 How to Determine a Futures Contract

Example 1 - Standard Method of Determining a Futures Contract

```
void onHowtoDetermineFuture(){
        Contract contract = new Contract();
        Order order = new Order();
        contract.m_symbol = "ES";
        contract.m_secType = "FUT";
        contract.m_expiry = "201109";
        contract.m_exchange = "GLOBEX";
        contract.m_currency = "USD";
        order.m_action = "BUY";
        order.m_totalQuantity = 1;
        order.m_orderType = "LMT";
        order.m_lmtPrice = enteredLmtPrice;
        m_client.placeOrder(GlobalOrderId, contract, order);
}
```

Example 2 - Determining a Futures Contract Using the Local Symbol

```
void inUsingLocalSymbolForFuture(){
        Contract contract = new Contract();
        Order order = new Order();
        contract.m_localSymbol = "ESU1";
        contract.m_secType = "FUT";
        contract.m_exchange = "GLOBEX";
        contract.m_currency = "USD";
        order.m_action = "BUY";
        order.m_totalQuantity = 1;
        order.m_orderType = "LMT";
        order.m_lmtPrice = enteredLmtPrice;
        m_client.placeOrder(GlobalOrderId, contract, order);
}
```

How to Determine a Stock

```
void onHowToDetermineStock(){
        Contract contract = new Contract();
        Order order = new Order();
        contract.m_symbol = "IBKR";
        contract.m_secType = "STK";
        contract.m_exchange = "SMART";
        contract.m_currency = "USD";
        order.m_action = "BUY";
        order.m_totalQuantity = 100;
        order.m_orderType = "LMT";
        order.m_lmtPrice = enteredLmtPrice;
        m_client.placeOrder(GlobalOrderId, contract, order);
}
```

Chapter 18

Financial Advisor Orders and Account Configuration

This section assumes familiarity on the part of the reader with TWS Financial Advisor account configuration and order placement.

API FA functionality became significantly more powerful in TWS release 821 and higher, in that the now deprecated ”allocation string” method was replaced by the much more powerful Financial Advisor order allocation methods. Prior to those new methods being used, TWS had to be configured to understand the desired FA order groups, profiles, and account aliases. This can be done manually in TWS, or via the API, or via both.

18.1 Support by Other API Technologies

For all ActiveX, Java, or C++ based API technologies, TWS Financial Advisor account configuration is done via two new methods and one new event. The methods are called replaceFA, and requestFA. The event is called receiveFA. These methods and that event pertain to the following three parts of TWS FA account configuration: creating groups, profiles, and account aliases.

- **requestFA(int faDataType)** is a method that is called by an API application to request one of those types of FA configuration data.
- **receiveFA(int faDataType, string XML)** receives the requested data from TWS, via an event that TWS sends that contains the data requested. The event includes an XML string containing the requested information.
- **replaceFA(int faDataType, string XML)** can be called from the API if the API application wishes to replace the previous FA configuration information with a new XML string.

In accordance with the existence of this new functionality, all placeOrder methods, whether ActiveX, Java, or C++ based, have four new parameters pertaining to Financial Advisor order placement: faGroup, faMethod, faPercentage, and faProfile. When one or more of these new values is not relevant to an order, simply pass in an empty string.

18.2 Improved Financial Advisor Execution Reporting

When using TWS version 823 or higher, the execution messages resulting from a new FA order will report both the initial execution of the order, as well as its being allocated to its various subaccounts.

The following example helps explains Advisor execution reporting.

Assume that 100 shares of IBM is being bought on the NYSE by a Financial Advisor who has three sub-accounts, and who wants them allocated with Equal Quantity to each. The following seven execution messages will occur:

- FA Account: Order filled on NYSE to BUY 100 IBM
- FA Account: allocation of 34 shares out of FA account and into sub account 1. Message says ”BUY -34 IBM.” The negative quantity reflects the fact that the execution being reported is reducing the purchase.
- SUB1 Account: BUY 34 IBM.
- FA Account: allocation of 33 shares out of FA account and into sub account 2. Message says ”BUY -33 IBM.”
- SUB2 Account: BUY 33 IBM.
- FA Account: allocation of 33 shares out of FA account and into sub account 3. Message says ”BUY -33 IBM.”
- SUB3 Account: BUY 33 IBM.”

18.3 Allocation Methods for Account Groups

Note that you must type the method name in exactly as appears here, or your order won't work.

EqualQuantity Method

Requires you to specify an order size. This method distributes shares equally between all accounts in the group.

Example: You transmit an order for 400 shares of stock ABC. If your Account Group includes four accounts, each account receives 100 shares. If your Account Group includes six accounts, each account receives 66 shares, and then 1 share is allocated to each account until all are distributed.

NetLiq Method

Requires you to specify an order size. This method distributes shares based on the net liquidation value of each account. The system calculates ratios based on the Net Liquidation value in each account and allocates shares based on these ratios.

Example: You transmit an order for 700 shares of stock XYZ. The account group includes three accounts, A, B and C with Net Liquidation values of $25,000, $50,000 and $100,000 respectively. The system calculates a ratio of 1:2:4 and allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

AvailableEquity Method

Requires you to specify an order size. This method distributes shares based on the amount of equity with loan value currently available in each account. The system calculates ratios based on the Equity with Loan value in each account and allocates shares based on these ratios.

Example: You transmit an order for 700 shares of stock XYZ. The account group includes three accounts, A, B and C with available equity in the amounts of $25,000, $50,000 and $100,000 respectively. The system calculates a ratio of 1:2:4 and allocates 100 shares to Client A, 200 shares to Client B, and 400 shares to Client C.

PctChange Method

This method only works when you already hold a position in the selected instrument. Do not specify an order size. Since the quantity is calculated by the system, the order size is displayed in the Quantity field after the order is acknowledged. This method increases or decreases an already existing position. Positive percents will increase a position,

negative percents will decrease a position.

Example 1: Assume that three of the six accounts in this group hold long positions in stock XYZ. Client A has 100 shares, Client B has 400 shares, and Client C has 200 shares. You want to increase their holdings by 50%, so you enter ”50” in the percentage field. The system calculates that your order size needs to be equal to 350 shares. It then allocates 50 shares to Client A, 200 shares to Client B, and 100 shares to Client C.

Example 2: You want to close out all long positions for three of the five accounts in a group. You create a sell order and enter ”-100” in the Percentage field. The system calculates 100% of each position for every account in the group that holds a position, and sells all shares to close the positions.

These handy charts make it easy to see how negative and positive percent values will affect long and short positions for both buy and sell orders. Phew, that was a mouthful!

BUY ORDER	Positive Percent	Negative Percent
Long Position	Increases position	No effect
Short Position	No effect	Decreases position

SELL ORDER	Positive Percent	Negative Percent
Long Position	No effect	ecreases position
Short Position	Increases position	No effect

18.4 Java Code Samples for Financial Advisor API Orders

There are generally three methods for placing an order in the API from a Financial Advisor (FA) account:

- Place an order for a single managed account.
- Place an order for an allocation profile.
- Place an order for an account group.

Place an Order for a Single Managed Account

As an FA, you can place an order for any one of your managed accounts. The following code sample performs this task.

```
Contract m_contract = new Contract();
Order m_order = new Order();
/** Stocks */
m_contract.m_symbol = "IBM";
m_contract.m_secType = "STK";
m_contract.m_exchange = "SMART";
m_contract.m_currency = "USD";
m_order.m_orderType = "MKT";
m_order.m_action = "BUY";
m_order.m_totalQuantity = 100;
m_order.m_transmit = true;
// allocate the order for this particular account
m_order.m_account = "DU74649";
m_client.placeOrder(orderId++, m_contract, m_order);
```

Place an Order for an Allocation Profile

As an FA, you can place an order for accounts that share an allocation profile. The following code sample performs the task.

Note: Before trying this yourself, you must be familiar with setting up an allocation profile and placing an order in TWS.

```
Contract m_contract = new Contract();
Order m_order = new Order();
.
.
// allocate the order for this profile
m_order.m_faProfile = "USClients";
m_client.placeOrder(orderId++, m_contract, m_order);
```

Place an Order for an Account Group

As an FA, you can place an order for accounts in an account group. Note that the method attribute is a mandatory field when placing an order for account groups. The following code sample performs the task.

Note: Before trying this yourself, you must be familiar with setting up account groups and placing an order in TWS.

```
Contract m_contract = new Contract();
Order m_order = new Order();
.
.
// allocate the order for this group
m_order.m_faGroup = "USGroup";
// using the percent change method
m_order.m_faMethod = "PctChange";
m_order.m_faPercentage = "100";
m_client.placeOrder(orderId++, m_contract, m_order);
```

Changing/Updating Allocation Information

As an FA, you can retrieve allocation information in XML format, and change or update allocation information by passing an XML formatted configuration back.

The following code sample changes one of the groups' name by replacing the first occurrence of "TestGroup" in the configuration file with "MyTestGroup" and passing it back.

```
public void receiveFA(int faDataType, String xml) {
        switch (faDataType) {
                case EClientSocket.GROUPS:
                        faGroupXML = xml ;
                        String test = xml.replaceFirst("TestGroup", "MyTestGroup");
                        m_client.replaceFA(1, test);
                        break ;
                case EClientSocket.PROFILES:
                        faProfilesXML = xml ;
                        break ;
                case EClientSocket.ALIASES:
                        faAliasesXML = xml ;
                        break ;
        }
```

Part II

IB Reference

Chapter 19

Message

19.1 API Message Codes

This section lists all of the API Error, System and Warning message codes and their descriptions.

Message codes shown below that end with a colon (:) display additional information.

Error Codes

Code	Description
100	Max rate of messages per second has been exceeded.
101	Max number of tickers has been reached.
102	Duplicate ticker ID.
103	Duplicate order ID.
104	Can't modify a filled order.
105	Order being modified does not match original order.
106	Can't transmit order ID:
107	Cannot transmit incomplete order.
109	Price is out of the range defined by the Percentage setting at order defaults frame. The order will not be transmitted.
110	The price does not conform to the minimum price variation for this contract.
111	The TIF (Tif type) and the order type are incompatible.
113	The Tif option should be set to DAY for MOC and LOC orders.
114	Relative orders are valid for stocks only.
115	Relative orders for US stocks can only be submitted to SMART, SMART_ECN, INSTINET, or PRIMEX.
116	The order cannot be transmitted to a dead exchange.
117	The block order size must be at least 50.
118	VWAP orders must be routed through the VWAP exchange.
119	Only VWAP orders may be placed on the VWAP exchange.
120	It is too late to place a VWAP order for today.
121	Invalid BD flag for the order. Check "Destination" and "BD" flag.
122	No request tag has been found for order:
123	No record is available for conid:
124	No market rule is available for conid:
125	Buy price must be the same as the best asking price.
126	Sell price must be the same as the best bidding price.
129	VWAP orders must be submitted at least three minutes before the start time.
131	The sweep-to-fill flag and display size are only valid for US stocks routed through SMART, and will be ignored.
132	This order cannot be transmitted without a clearing account.
133	Submit new order failed.
134	Modify order failed.
135	Can't find order with ID =
136	This order cannot be cancelled.
137	VWAP orders can only be cancelled up to three minutes before the start time.
138	Could not parse ticker request:
139	Parsing error:
140	The size value should be an integer:

Code	Description
141	The price value should be a double:
142	Institutional customer account does not have account info
143	Requested ID is not an integer number.
144	Order size does not match total share allocation. To adjust the share allocation, right-click on the order and select Modify ¿. Share Allocation.
145	Error in validating entry fields -
146	Invalid trigger method.
147	The conditional contract info is incomplete.
148	A conditional order can only be submitted when the order type is set to limit or market
151	This order cannot be transmitted without a user name.
152	The ”hidden” order attribute may not be specified for this order.
153	EFPs can only be limit orders.
154	Orders cannot be transmitted for a halted security.
155	A sizeOp order must have a username and account.
156	A SizeOp order must go to IBSX
157	An order can be EITHER Iceberg or Discretionary. Please remove either the Discretionary amount or the Display size.
158	You must specify an offset amount or a percent offset value.
159	The percent offset value must be between 0% and 100%.
160	The size value cannot be zero.
161	Cancel attempted when order is not in a cancellable state. Order permId =
162	Historical market data Service error message.
163	The price specified would violate the percentage constraint specified in the default order settings.
164	There is no market data to check price percent violations.
165	Historical market Data Service query message.
166	HMDS Expired Contract Violation.
167	VWAP order time must be in the future.
168	Discretionary amount does not conform to the minimum price variation for this contract.

Code	Description
200	No security definition has been found for the request.
201	Order rejected - Reason:
202	Order cancelled - Reason:
203	The security <security> is not available or allowed for this account.

Code	Description
300	Can't find EId with ticker Id:
301	Invalid ticker action:
302	Error parsing stop ticker string:
303	Invalid action:
304	Invalid account value action:
305	Request parsing error, the request has been ignored.
306	Error processing DDE request:
307	Invalid request topic:
308	Unable to create the 'API' page in TWS as the maximum number of pages already exists.
309	Max number (3) of market depth requests has been reached. Note: TWS currently limits users to a maximum of 3 distinct market depth requests. This same restriction applies to API clients, however API clients may make multiple market depth requests for the same security
310	Can't find the subscribed market depth with tickerId:
311	The origin is invalid.
312	The combo details are invalid.
313	The combo details for leg '¡leg number¿' are invalid.
314	Security type 'BAG' requires combo leg details.
315	Stock combo legs are restricted to SMART order routing.
316	Market depth data has been HALTED. Please re-subscribe.
317	Market depth data has been RESET. Please empty deep book contents before applying any new entries.
319	Invalid log level ¡log level¿
320	Server error when reading an API client request.
321	Server error when validating an API client request.
322	Server error when processing an API client request. If you get this error when submitting a reqAccountSummary() request, unsubscribe from one reqAccountSummary() request and then resubmit the request. reqAccountSummary() only allows two concurrent requests.
323	Server error: cause - %s
324	Server error when reading a DDE client request (missing information).
325	Discretionary orders are not supported for this combination of exchange and order type
326	Unable to connect as the client id is already in use. Retry with a unique client id.
327	Only API connections with clientId set to 0 can set the auto bind TWS orders property
328	Trailing stop orders can be attached to limit or stop-limit orders only.
329	Order modify failed. Cannot change to the new order type.
330	Only FA or STL customers can request managed accounts list.
331	Internal error. FA or STL does not have any managed accounts.
332	The account codes for the order profile are invalid.
333	Invalid share allocation syntax.
334	Invalid Good Till Date order
335	Invalid delta: The delta must be between 0 and 100.

Code	Description
336	The time or time zone is invalid. The correct format is hh:mm:ss xxx where xxx is an optionally specified time-zone. E.g.: 15:59:00 EST Note that there is a space between the time and the time zone. If no time zone is specified, local time is assumed.
337	”The date, time, or time-zone entered is invalid. The correct format is yyyymmdd hh:mm:ss xxx where yyyymmdd and xxx are optional. E.g.: 20031126 15:59:00 EST Note that there is a space between the date and time, and between the time and time-zone. If no date is specified, current date is assumed. If no time-zone is specified, local time-zone is assumed.”
338	Good After Time orders are currently disabled on this exchange.
339	Futures spread are no longer supported. Please use combos instead.
340	Invalid improvement amount for box auction strategy.
341	”Invalid delta. Valid values are from 1 to 100. You can set the delta from the ””Pegged to Stock”” section of the Order Ticket Panel, or by selecting Page/Layout from the main menu and adding the Delta column.”
342	Pegged order is not supported on this exchange.
343	”The date, time, or time-zone entered is invalid. The correct format is yyyymmdd hh:mm:ss xxx where yyyymmdd and xxx are optional. E.g.: 20031126 15:59:00 EST Note that there is a space between the date and time, and between the time and time-zone. If no date is specified, current date is assumed. If no time-zone is specified, local time-zone is assumed.”
344	The account logged into is not a financial advisor account.
345	Generic combo is not supported for FA advisor account.
346	Not an institutional account or an away clearing account.
347	Short sale slot value must be 1 (broker holds shares) or 2 (delivered from elsewhere).
348	Order not a short sale – type must be SSHORT to specify short sale slot.
349	Generic combo does not support ”Good After” attribute.
350	Minimum quantity is not supported for best combo order.
351	The ”Regular Trading Hours only” flag is not valid for this order.
352	Short sale slot value of 2 (delivered from elsewhere) requires location.
353	Short sale slot value of 1 requires no location be specified.
354	Not subscribed to requested market data.
355	Order size does not conform to market rule.
356	Smart-combo order does not support OCA group.
357	Your client version is out of date.
358	Smart combo child order not supported.
359	Combo order only supports reduce on fill without block(OCA).
360	No whatif check support for smart combo order.
361	Invalid trigger price.
362	Invalid adjusted stop price.
363	Invalid adjusted stop limit price.
364	Invalid adjusted trailing amount.
365	No scanner subscription found for ticker id:
366	No historical data query found for ticker id:
367	Volatility type if set must be 1 or 2 for VOL orders. Do not set it for other order types
368	Reference Price Type must be 1 or 2 for dynamic volatility management. Do not set it for non-VOL orders.
369	Volatility orders are only valid for US options.
370	Dynamic Volatility orders must be SMART routed, or trade on a Price Improvement Exchange

Code	Description
371	VOL order requires positive floating point value for volatility. Do not set it for other order types.
372	Cannot set dynamic VOL attribute on non-VOL order.
373	Can only set stock range attribute on VOL or RELATIVE TO STOCK order.
374	If both are set, the lower stock range attribute must be less than the upper stock range attribute.
375	Stock range attributes cannot be negative.
376	The order is not eligible for continuous update. The option must trade on a cheap-to-reroute exchange.
377	Must specify valid delta hedge order aux. price.
378	Delta hedge order type requires delta hedge aux. price to be specified.
379	Delta hedge order type requires that no delta hedge aux. price be specified.
380	This order type is not allowed for delta hedge orders.
381	Your DDE.dll needs to be upgraded.
382	The price specified violates the number of ticks constraint specified in the default order settings.
383	The size specified violates the size constraint specified in the default order settings.
384	Invalid DDE array request.
385	Duplicate ticker ID for API scanner subscription.
386	Duplicate ticker ID for API historical data query.
387	Unsupported order type for this exchange and security type.
388	Order size is smaller than the minimum requirement.
389	Supplied routed order ID is not unique.
390	Supplied routed order ID is invalid.
391	”The time or time-zone entered is invalid. The correct format is hh:mm:ss xxx where xxx is an optionally specified time-zone. E.g.: 15:59:00 EST. Note that there is a space between the time and the time zone. If no time zone is specified, local time is assumed.”
392	Invalid order: contract expired.
393	Short sale slot may be specified for delta hedge orders only.
394	”Invalid Process Time: must be integer number of milliseconds between 100 and 2000. Found:”
395	Due to system problems, orders with OCA groups are currently not being accepted.
396	Due to system problems, application is currently accepting only Market and Limit orders for this contract.
397	Due to system problems, application is currently accepting only Market and Limit orders for this contract.
398	$<>$ cannot be used as a condition trigger.
399	Order message error

Code	Description
400	Algo order error.
401	Length restriction.
402	Conditions are not allowed for this contract.
403	Invalid stop price.
404	Shares for this order are not immediately available for short sale. The order will be held while we attempt to locate the shares.
405	The child order quantity should be equivalent to the parent order size.
406	The currency <> is not allowed.
407	The symbol should contain valid non-unicode characters only.
408	Invalid scale order increment.
409	Invalid scale order. You must specify order component size.
410	Invalid subsequent component size for scale order.
411	The ”Outside Regular Trading Hours” flag is not valid for this order.
412	The contract is not available for trading.
413	What-if order should have the transmit flag set to true.
414	Snapshot market data subscription is not applicable to generic ticks.
415	Wait until previous RFQ finishes and try again.
416	RFQ is not applicable for the contract. Order ID:
417	Invalid initial component size for scale order.
418	Invalid scale order profit offset.
419	Missing initial component size for scale order.
420	Invalid real-time query.
421	Invalid route.
422	The account and clearing attributes on this order may not be changed.
423	”Cross order RFQ has been expired. THI committed size is no longer available. Please open order dialog and verify liquidity allocation.”
424	FA Order requires allocation to be specified.
425	”FA Order requires per-account manual allocations because there is no common clearing instruction. Please use order dialog Adviser tab to enter the allocation.”
426	None of the accounts have enough shares.
427	Mutual Fund order requires monetary value to be specified.
428	Mutual Fund Sell order requires shares to be specified.
429	Delta neutral orders are only supported for combos (BAG security type).
430	We are sorry, but fundamentals data for the security specified is not available.
431	What to show field is missing or incorrect.
432	Commission must not be negative.
433	Invalid ”Restore size after taking profit” for multiple account allocation scale order
434	The order size cannot be zero.
435	You must specify an account.
436	You must specify an allocation (either a single account, group, or profile).
437	Order can have only one flag Outside RTH or Allow PreOpen.
438	The application is now locked.
439	Order processing failed. Algorithm definition not found.
440	Order modify failed. Algorithm cannot be modified.
441	Algo attributes validation failed:
442	Specified algorithm is not allowed for this order.
443	Order processing failed. Unknown algo attribute.
444	Volatility Combo order is not yet acknowledged. Cannot submit changes at this time
445	The RFQ for this order is no longer valid.

Code	Description
446	Missing scale order profit offset.
447	Missing scale price adjustment amount or interval.
448	Invalid scale price adjustment interval.
449	Unexpected scale price adjustment amount or interval.
450	Dividend schedule query failed.
501	Already connected.
502	Couldn't connect to TWS. Confirm that API is enabled in TWS via the Configure> API menu command.
503	Your version of TWS is out of date and must be upgraded.
504	Not connected.
505	Fatal error: Unknown message id.
506	Unsupported version. For Java clients only.
507	Bad message length. For Java clients only.
508	Bad message. For Java clients only.
510	Request market data - sending error:
511	Cancel market data - sending error:
512	Order - sending error:
513	Account update request - sending error:
514	Request for executions - sending error:
515	Cancel order - sending error:
516	Request open order - sending error:
517	Unknown contract. Verify the contract details supplied.
518	Request contract data - sending error:
519	Request market depth - sending error:
520	Cancel market depth - sending error:
521	Set server log level - sending error:
522	FA Information Request - sending error:
523	FA Information Replace - sending error:
524	Request Scanner subscription - sending error:
525	Cancel Scanner subscription - sending error:

Code	Description
526	Request Scanner parameter - sending error:
527	Request Historical data - sending error:
528	Cancel Historical data - sending error:
529	Request real-time bar data - sending error:
530	Cancel real-time bar data - sending error:
531	Request Current Time - Sending error:

Code	Description
Code	Description
10000	Cross currency combo error.
10001	Cross currency vol error.
10002	Invalid non-guaranteed legs.
10003	IBSX not allowed.
10005	Read-only models.
10006	Missing parent order.
10007	Invalid hedge type.
10008	Invalid beta value.
10009	Invalid hedge ratio.
10010	Invalid delta hedge order.
10011	Currency is not supported for Smart combo.
10012	Invalid allocation percentage
10013	Smart routing API error (Smart routing opt-out required).
10014	PctChange limits.
10015	Trading is not allowed in the API.
10016	Contract is not visible.
10017	Contracts are not visible.
10018	Orders use EV warning.
10019	Trades use EV warning.
10020	Display size should be smaller than order size.
10021	Invalid leg2 to Mkt Offset API.
10022	Invalid Leg Prio API.
10023	Invalid combo display size API.
10024	Invalid dont start next legin API.
10025	Invalid leg2 to Mkt time1 API.
10026	Invalid leg2 to Mkt time2 API.
10027	Invalid combo routing tag API.

System Message Codes

Code	Description
Code	Description
1100	Connectivity between IB and TWS has been lost.
1101	Connectivity between IB and TWS has been restoreddata lost.*
1102	Connectivity between IB and TWS has been restoreddata maintained.
1300	TWS socket port has been reset and this connection is being dropped. Please reconnect on the new port - <port_num>

Market and account data subscription requests must be resubmitted

Warning Message Codes

Code	Description
Code	Description
2100	New account data requested from TWS. API client has been unsubscribed from account data.
2101	Unable to subscribe to account as the following clients are subscribed to a different account.
2102	Unable to modify this order as it is still being processed.
2103	A market data farm is disconnected.
2104	A market data farm is connected.
2105	A historical data farm is disconnected.
2106	A historical data farm is connected.
2107	A historical data farm connection has become inactive but should be available upon demand.
2108	A market data farm connection has become inactive but should be available upon demand.
2109	Order Event Warning: Attribute Outside Regular Trading Hours is ignored based on the order type and destination. PlaceOrder is now processed.
2110	Connectivity between TWS and server is broken. It will be restored automatically.

Chapter 20

Historical Data

20.1 Historical Data Limitations

Historical data requests are subject to the following limitations:

- Historical data requests that use a bar size below 30 seconds can only go back six months.
- Historical data requests can go back one full calendar year or more, depending on the number of concurrent realtime market data lines:

Number of Market Data Lines	Historical Data Request Limit
Less than 499	One year
500 - 749	Two years
750 - 999	Three years
1000	Four years

Market data lines can be increased based on monthly commission amounts, amount of equity and Quote Booster subscriptions.

- For more information on how market data is affected by commissions and equity, see the second-to-the-last Note (the long note with the examples) at the bottom of the Market Data page on our website.
- For more information on Quote Boosters, see the Quote Boosters page on our website. You subscribe to Quote Boosters on the Market Data Subscriptions page in Account Management.

Pacing Violations

All of the API technologies support historical data requests. However, requesting the same historical data in a short period of time can cause extra load on the backend and subsequently cause pacing violations. The error code and message that indicates a pacing violation is:

162 - Historical Market Data Service error message: Historical data request pacing violation

The following conditions can cause a pacing violation:

- Making identical historical data requests within 15 seconds;
- Making six or more historical data requests for the same Contract, Exchange and Tick Type within two seconds.

Also, observe the following limitation when requesting historical data:

- Do not make more than 60 historical data requests in any ten-minute period.
- If the whatToShow parameter in reqHistoricalData() is set to BID_ASK, then this counts as two requests and we will call BID and ASK historical data separately.

Note: For more information about historical data requests, see Viewing Historical Data in the DDE for Excel chapter, reqHistoricalDataEx() in the ActiveX chapter, reqHistoricalData() in the C++ chapter, and reqHistoricalData() in the Java chapter.

Minimum Bar Size Settings for Historical Data Requests

The following table lists the minimum bar size settings for API historical data requests.

Duration	Minimum Bar Size
1 min	1 second
5 mins	1 second
15 mins	1 second
1 hour	5 seconds
2 hours	5 seconds
4 hours	10 seconds
1 day	30 seconds
2 days	1 minute
1 week	10 minutes
2 weeks	15 minutes
1 month	30 minutes
3 months	1 day
Everything else	1 day

Valid Duration and Bar Size Settings for Historical Data Requests

The following table lists the minimum bar size settings for API historical data requests.

The following table lists valid duration and bar size settings for API historical data requests. Please note that these are only guidelines.

Duration	Bar Size
1 Y	1 day
6 M	1 day
3 M	1 day
1 M	1 day, 1 hour
1 W	1 day, 1 hour, 30 mins, 15 mins
2 D	1 hour, 30 mins, 15 mins, 3 mins, 2 mins, 1 min
1 D	1 hour, 30 mins, 15 mins, 5 mins, 3 mins, 2 mins, 1 min, 30 secs
14400 S (4 hrs)	1 hour, 30 mins, 15 mins, 5 mins, 3 mins, 2 mins, 1 min, 30 secs, 15 secs
7200 S (2 hrs)	1 hour, 30 mins, 15 mins, 5mins, 3 mins, 2 mins, 1 min, 30 secs, 15 secs, 5 secs
3600 S (1 hr)	15 mins, 5 mins, 3 mins, 2 mins, 1 min, 30 secs, 15 secs, 5 secs,
1800 S (30 mins)	15 mins, 5 mins, 3 mins, 2 mins, 1 min, 30 secs, 15 secs, 5 secs, 1 secs
960 S (15 mins.)	5 mins, 3 mins, 2 mins, 1 min, 30 secs, 15 secs, 5 secs, 1 secs
300 S (5 mins)	3 mins, 2 mins, 1 min, 30 secs, 15 secs, 5 secs, 1 secs
60 S (1 min)	30 secs, 15 secs, 5 secs, 1 secs

20.2 Tick Types

The following table lists all possible values for the tickType parameter, which is used in the following ActiveX events, C++ EWrapper functions, and Java EWrapper methods:

- tickPrice()
- tickSize()
- tickOptionComputation()
- tickGeneric()
- tickString()
- tickEFP

Tick Value	Description	Event/Function/Method
-1	Not applicable.	–
0	BID_SIZE	tickSize()
1	BID_PRICE	tickPrice()
2	ASK_PRICE	tickPrice()
3	ASK_SIZE	tickSize()
4	LAST_PRICE	tickPrice()
5	LAST_SIZE	tickSize()
6	HIGH	tickPrice()
7	LOW	tickPrice()
8	VOLUME	tickSize()
9	CLOSE_PRICE	tickPrice()
10	BID_OPTION_COMPUTATION	tickOptionComputation() See Note 1 below
11	ASK_OPTION_COMPUTATION	tickOptionComputation() See Note 1 below
12	LAST_OPTION_COMPUTATION	tickOptionComputation() See Note 1 below
13	MODEL_OPTION_COMPUTATION	tickOptionComputation() See Note 1 below
14	OPEN_TICK	tickPrice()
15	LOW_13_WEEK	tickPrice()
16	HIGH_13_WEEK	tickPrice()
17	LOW_26_WEEK	tickPrice()
18	HIGH_26_WEEK	tickPrice()
19	LOW_52_WEEK	tickPrice()
20	HIGH_52_WEEK	tickPrice()
21	AVG_VOLUME	tickSize()
22	OPEN_INTEREST	tickSize()
23	OPTION_HISTORICAL_VOL	tickGeneric()
24	OPTION_IMPLIED_VOL	tickGeneric()
25	OPTION_BID_EXCH	NOT USED
26	OPTION_ASK_EXCH	NOT USED
27	OPTION_CALL_OPEN_INTEREST	tickSize()
28	OPTION_PUT_OPEN_INTEREST	tickSize()
29	OPTION_CALL_VOLUME	tickSize()

Tick Value	Description	Event/Function/Method
30	OPTION_PUT_VOLUME	tickSize()
31	INDEX_FUTURE_PREMIUM	tickGeneric()
32	BID_EXCH	tickString()
33	ASK_EXCH	tickString()
34	AUCTION_VOLUME	tickSize()
35	AUCTION_PRICE	tickPrice()
36	AUCTION_IMBALANCE	tickSize()
37	MARK_PRICE	tickPrice()
38	BID_EFP_COMPUTATION	tickEFP()
39	ASK_EFP_COMPUTATION	tickEFP()
40	LAST_EFP_COMPUTATION	tickEFP()
41	OPEN_EFP_COMPUTATION	tickEFP()
42	HIGH_EFP_COMPUTATION	tickEFP()
43	LOW_EFP_COMPUTATION	tickEFP()
44	CLOSE_EFP_COMPUTATION	tickEFP()
45	LAST_TIMESTAMP	tickString()
46	SHORTABLE	tickString()
47	FUNDAMENTAL_RATIOS	tickString()
48	RT_VOLUME	tickString()
49	HALTED	See Note 2 below.
50	BIDYIELD	tickPrice() See Note 3 below
51	ASKYIELD	tickPrice() See Note 3 below
52	LASTYIELD	tickPrice() See Note 3 below
53	CUST_OPTION_COMPUTATION	tickOptionComputation()
54	TRADE_COUNT	tickGeneric()
55	TRADE_RATE	tickGeneric()
56	VOLUME_RATE	tickGeneric()

1. Tick types BID_OPTION_COMPUTATION, ASK_OPTION_COMPUTATION, LAST_OPTION_COMPUTATION, and MODEL_OPTION_COMPUTATION return all Greeks (delta, gamma, vega, theta), the underlying price and the stock and option reference price when requested. MODEL_OPTION_COMPUTATION also returns model implied volatility.

2. Prior to TWS Version 939, when trading is halted for a contract, TWS receives a special tick: haltedLast=1. When trading is resumed, TWS receives haltedLast=0. A tick type, HALTED, tick ID = 49, is available in regular market data via the API to indicate this halted state. Possible values for this tick type are:

   (a) 0 = Not halted

   (b) 1 = Halted.

   Beginning with TWS Version 939, possible values for the HALTED tick type are:

   (a) 0 = Not halted

   (b) 1 = General halt (trading halt is imposed for purely regulatory reasons) with/without volatility halt.

   (c) 2 = Volatility only halt (trading halt is imposed by the exchange to protect against extreme volatility).

3. Applies to bond contracts only.

20.3 Generic Tick Types

For all socket-based API technologies, including the socket client library, ActiveX and Java, we provide several types of market data ticks that can be requested as a part of the market data request.

Starting with API version 9.0 (client version 30), version 6 of the REQ_MKT_DATA message is sent containing a new field that specifies the requested ticks as a list of comma-delimited integer Ids (generic tick types). Requests for these ticks will be answered if the tick type requested pertains to the contract at issue.

Note that Generic Tick Tags cannot be specified if you elect to use the Snapshot market data subscription.

The generic market data tick types are:

Integer ID Value	Tick Type	Resulting Tick Value
100	Option Volume (currently for stocks)	29, 30
101	Option Open Interest (currently for stocks)	27, 28
104	Historical Volatility (currently for stocks)	23
106	Option Implied Volatility (currently for stocks)	24
162	Index Future Premium	31
165	Miscellaneous Stats	15, 16, 17, 18, 19, 20, 21
221	Mark Price (used in TWS P&L computations)	37
225	Auction values (volume, price and imbalance)	34, 35, 36
233	RTVolume - contains the last trade price, last trade size, last trade time, total volume, VWAP, and single trade flag.	48
236	Shortable	46
256	Inventory	
258	Fundamental Ratios	47
292	Receive top news for underlying contracts from TWS for news feeds to which you have subscribed (in Account Management). Use secType = NEWS in the Contract object. See Requesting News for examples. You will receive at least one news tick regardless of its timeliness (it could be new or it could be weeks old). Some news providers limit us to maximum news retention of 30 days, so those limitations may affect which news you see. Otherwise you will receive a maximum of five of the most recent news items in the last 24 hours, and you will receive additional news items as they come in.	62
411	Realtime Historical Volatility	58
456	IBDividends	59

Using the SHORTABLE Tick

In TWS, there is a SHORTABLE column, as shown below. The column describes number of shares with which the security can be sold short.

The color GREEN indicates that at least 1000 shares are available to sell short. DARK GREEN indicates that this contract can be sold short but that at the moment there are no shares available for short sale, and that the system is searching for shares. RED indicates that no shares are available for short sale.

With API 9.30 or higher, the shorting indicator is supported for all socket connections. The functionality equates to the SHORTABLE column in the TWS workstation.

When invoking the reqMktDataEx()/reqMktData() methods, you must include generic ticktype 236 in the argument to obtain the shortable value. For example:

The Shortable tick determines if SHORT SELL orders for a contract will be accepted. Analyze the value returned from tickGeneric(int tickerId, int tickType, double value) as follows:

```
if (value > 2.5) { // 3.0
// There are at least 1000 shares available for a short sale
// In TWS, this is identical to GREEN status
}
else if (value > 1.5) { // 2.0
// This contract will be available for short sale if shares can be // located
// In TWS, this is identical to DARK GREEN status
}
else if (value > 0.5) { // 1.0
// Not available for short sale
// In TWS, this is identical to RED status
}
else {
// unknown value
}
```

Note: This feature is supported as of server version 33 (872 release of TWS).

20.4 TAG Values for FUNDAMENTAL_RATIOS tickType

The FUNDAMENTAL_RATIOS tickType (Tick Value 47) lets you request fundamental ratios in the form TAGG=VALUE, TAG2=VALUE2, and so on. This ratios are sent using the tickGeneric() callback. The following table lists all the TAG values for FUNDAMENTAL_RATIOS.

TAG	Description
NPRICE	**Closing Price** This is the closing price for the issue from the day it last traded. It is also referred to as the Current Price. Note that some issues may not trade every day, and therefore it is possible for this price to come from a date prior to the last business day.
Three_Year_TTM_Growth	3 year trailing twelve months growth.
TTM_over_TTM	Trailing twelve months over trailing twelve months.
NHIG	**High Price** This price is the highest price the stock traded at in the last 12 months. This could be an intra-day high.
NLOW	**Low Price** This price is the lowest price the stock traded at in the last 12 months. This could be an intra-day low.
PDATE	**Pricing date** The pricing date is the date at which the issue was last priced.
VOL10DAVG	**Volume** This is the daily average of the cumulative trading volume for the last ten days.
MKTCAP	**Market capitalization** This value is calculated by multiplying the current Price by the current number of shares outstanding.
TTMEPSXCLX	**EPS excluding extraordinary items** This is the Adjusted Income Available to Common Stockholders for the trailing twelve months divided by the trailing twelve month Diluted Weighted Average Shares Outstanding.
AEPSNORM	**EPS Normalized** This is the Normalized Income Available to Common Stockholders for the most recent annual period divided by the same period's Diluted Weighted Average Shares Outstanding.
TTMREVPS	**Revenue/share** This value is the trailing twelve month Total Revenue divided by the Average Diluted Shares Outstanding for the trailing twelve months. Note: Most banks and insurance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the trailing twelve month values will not be available (NA).
QBVPS	**Book value (Common Equity) per share** This is defined as the Common Shareholder's Equity divided by the Shares Outstanding at the end of the most recent interim period. Book Value is the Total Shareholder's Equity minus Preferred Stock and Redeemable Preferred Stock.
QTANBVPS	**Book value (tangible) per share** This is the interim Tangible Book Value divided by the Shares Outstanding at the end of the most recent interim period. Tangible Book Value is the Book Value minus Goodwill and Intangible Assets for the same period.

TAG	Description
QCSHPS	**Cash per share** This is the Total Cash plus Short Term Investments divided by the Shares Outstanding at the end of the most recent interim period. Note: This does NOT include cash equivalents that may be reported under long term assets.
TTMCFSHR	**Cash Flow per share** This value is the trailing twelve month Cash Flow divided by the trailing twelve month Average Shares Outstanding. Cash Flow is defined as the sum of Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
TTMDIVSHR	**Dividends per share** This is the sum of the Cash Dividends per share paid to common stockholders during the last trailing twelve month period.
IAD	**Dividend rate** This value is the total of the expected dividend payments over the next twelve months. It is generally the most recent cash dividend paid or declared multiplied by the dividend payment frequency, plus any recurring extra dividends.
PEEXCLXOR	**P/E excluding extraordinary items** This ratio is calculated by dividing the current Price by the sum of the Diluted Earnings Per Share from continuing operations BEFORE Extraordinary Items and Accounting Changes over the last four interim periods.
APENORM	**P/E Normalized** This is the Current Price divided by the latest annual Normalized Earnings Per Share value.
TMPR2REV	**Price to sales** This is the current Price divided by the Sales Per Share for the trailing twelve months. If there is a preliminary earnings announcement for an interim period that has recently ended, the revenue (sales) values from this announcement will be used in calculating the trailing twelve month revenue per share. NOTE: Most Banks and Finance companies do not report revenues when they announce their preliminary interim financial results in the press. When this happens, the trailing twelve month values will not be available (NA) until the complete interim filing is released.
PR2TANBK	**Price to Tangible Book** This is the Current Price divided by the latest annual Tangible Book Value Per Share. Tangible Book Value Per Share is defined as Book Value minus Goodwill and Intangible Assets divided by the Shares Outstanding at the end of the fiscal period.
TTMPRCFPS	**Price to Cash Flow per share** This is the current Price divided by Cash Flow Per Share for the trailing twelve months. Cash Flow is defined as Income After Taxes minus Preferred Dividends and General Partner Distributions plus Depreciation, Depletion and Amortization.
PRICE2BK	**Price to Book** This is the Current Price divided by the latest interim period Book Value Per Share.
QCURRATIO	**Current ratio** This is the ratio of Total Current Assets for the most recent interim period divided by Total Current Liabilities for the same period. NOTE: This item is Not Available (NA) for Banks, Insurance companies and other companies that do not distinguish between current and long term assets and liabilities.

TAG	Description
QQUICKRATI	**Quick ratio** Also known as the Acid Test Ratio, this ratio is defined as Cash plus Short Term Investments plus Accounts Receivable for the most recent interim period divided by the Total Current Liabilities for the same period. NOTE: This item is Not Available (NA) for Banks, Insurance companies and other companies that do not distinguish between current and long term assets and liabilities.
QLTD2EQ	**LT debt/equity** This ratio is the Total Long Term Debt for the most recent interim period divided by Total Shareholder Equity for the same period.
QTOTD2EQ	**Total debt/total equity** This ratio is Total Debt for the most recent interim period divided by Total Shareholder Equity for the same period. NOTE: This is Not Meaningful (NM) for banks.
TTMPAYRAT	**Payout ratio** This ratio is the percentage of the Primary/Basic Earnings Per Share Excluding Extraordinary Items paid to common stockholders in the form of cash dividends during the trailing twelve months.
TTMREV	**Revenue** This is the sum of all revenue (sales) reported for all operating divisions for the most recent TTM period. NOTE: Most banks and Insurance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the quarterly value will not be available (NA).
TTMEBITD	**EBITD** Earnings Before Interest, Taxes, Depreciation and Amortization (EBITDA) is EBIT for the trailing twelve months plus the same period's Depreciation and Amortization expenses (from the Statement of Cash Flows). NOTE: This item is only available for Industrial and Utility companies.
TTMEBT	**Earnings before taxes** Also known as Pretax Income and Earnings Before Taxes, this is Total Revenue for the most recent TTM period minus Total Expenses plus Non-operating Income (Expenses) for the same period.
TTMNIAC	**Net Income available to common** This is the trailing twelve month dollar amount accruing to common shareholders for dividends and retained earnings. Income Available to Common Shareholders is calculated as trailing twelve month Income After Taxes plus Minority Interest and Equity in Affiliates plus Preferred Dividends, General Partner Distributions and US GAAP Adjustments. NOTE: Any adjustment that is negative (ie. Preferred Stock Dividends) would be subtracted from Income After Taxes.
AEBTNORM	**Earnings before taxes Normalized** This is the Income Before Tax number excluding the impact of all unusual/one-time/special charges items for the most recent annual period.
ANIACNORM	**Net Income Available to Common, Normalized** This is the annual dollar amount accruing to common shareholders for dividends and retained earnings excluding the impact of all unusual/one-time/special charges items.

TAG	Description
TTMGROSMGN	**Gross Margin** This value measures the percent of revenue left after paying all direct production expenses. It is calculated as the trailing 12 months Total Revenue minus the trailing 12 months Cost of Goods Sold divided by the trailing 12 months Total Revenue and multiplied by 100. NOTE: This item is only available for Industrial and Utility companies.
TTMNPMGN	**Net Profit Margin %** Also known as Return on Sales, this value is the Income After Taxes for the trailing twelve months divided by Total Revenue for the same period and is expressed as a percentage. NOTE: Most Banks and Finance companies do not report revenues when they announce their preliminary quarterly financial results in the press. When this happens, the trailing twelve month value will not be available (NA).
TTMOPMGN	**Operating margin** This value measures the percent of revenues remaining after paying all operating expenses. It is calculated as the trailing 12 months Operating Income divided by the trailing 12 months Total Revenue, multiplied by 100. Operating Income is defined as Total Revenue minus Total Operating Expenses.
APTMGNPCT	**Pretax margin** This value represents Income Before Taxes for the most recent fiscal year expressed as a percent of Total Revenue for the most recent fiscal year.
TTMROAPCT	**Return on average assets** This value is the Income After Taxes for the trailing twelve months divided by the Average Total Assets, expressed as a percentage. Average Total Assets is calculated by adding the Total Assets for the 5 most recent quarters and dividing by 5.
TTMROEPCT	**Return on average equity** This value is the Income Available to Common Stockholders for the trailing twelve months divided by the Average Common Equity and is expressed as a percentage. Average Common Equity is calculated by adding the Common Equity for the 5 most recent quarters and dividing by 5.
TTMROIPCT	**Return on investment** This value is the trailing twelve month Income After Taxes divided by the average Total Long Term Debt, Other Long Term Liabilities and Shareholders Equity, expressed as a percentage.
REVCHNGYR	**Revenue Change %** This value is calculated as the most recent interim period Sales minus the Sales for the same interim period 1 year ago divided by the Sales for the same interim period one year ago, multiplied by 100.
TTMREVCHG	**Revenue Change %** This is the percent change in the trailing twelve month Sales as compared to the same trailing twelve month period one year ago. It is calculated as the trailing twelve month Sales minus the trailing twelve month Sales one year ago divided by the trailing twelve month Sales one year ago, multiplied by 100.
REVTRENDGR	**Revenue growth rate** The Five Year Revenue Growth Rate is the annual compounded growth rate of Revenues over the last 5 years.

TAG	Description
EPSCHNGYR	**EPS Change %** This value is calculated as the most recent interim period EPS minus the EPS for the same interim period 1 year ago divided by the EPS for the same interim period one year ago, multiplied by 100. NOTE: EPS must be positive for both periods. If either EPS value is negative, the result in Not Meaningful (NM).
TTMEPSCHG	**EPS Change %** This is the percent change in the trailing twelve month EPS as compared to the same trailing twelve month period one year ago. It is calculated as the trailing twelve month EPS minus the trailing twelve month EPS one year ago divided by the trailing twelve month EPS one year ago, multiplied by 100. NOTE: If either value has a negative value, the resulting value will be Not Meaningful (NM).
EPSTRENDGR	**EPS growth rate** This growth rate is the compound annual growth rate of Earnings Per Share Excluding Extraordinary Items and Discontinued Operations over the last 5 years. NOTE: If the value for either the most recent year or the oldest year is zero or negative, the growth rate cannot be calculated and a 'NA' (Not Available) code will be used.
DIVGRPCT	**Growth rate % - dividend** The Dividend Growth Rate is the compound annual growth rate in dividends per share. DIVGR% is calculated for 3 years whenever 4 years of dividends are available

IBDividends Tick Example

The IBDividends generic tick returns a comma-separated list of dividends in the following order:

1. sum of dividends for the past 12 months
2. sum of dividends for the next 12 months
3. next dividend date
4. next single dividend amount

Example

Here is an example of an IBDividends tick update for the symbol MSFT:

```
0.83,0.92,20130219,0.23
```

Where

```
0.83 = sum of dividends for the past 12 months
0.92 - sum of dividends for the next 12 months
20130219 - next dividend date
0.23 - next single dividend amount
```

RTVolume

RTVolume is one of the generic tick tags that can be requested as part of a market data request. RTVolume returns the following:

- Last trade price
- Last trade size

- Last trade time
- Total volume
- VWAP
- Single trade flag - True indicates the trade was filled by a single market maker; False indicates multiple marketmakers helped fill the trade

RTVolume is the API equivalent to opening the Time and Sales Window in Trader Workstation and viewing the updates in real time. To implement this, you must include 233 in the genericTicklist parameter in your market data request.

You will receive the RTVolume update through the tickString() event within field value 48.

Example

Here is an example of the RTVolume formatting for AAPL:

```
RTVolume=701.28;1;1348075471534;67854;701.46918464;true
RTVolume=701.26;3;1348075476533;67857;701.46917554;false
RTVolume=701.27;3;1348075482034;67860;701.46916674;true
RTVolume=701.27;3;1348075482336;67863;701.46915809;false
RTVolume=701.25;1;1348075483534;67864;701.46915486;true
RTVolume=701.24;1;1348075487029;67865;701.46915151;true
RTVolume=701.25;1;1348075489787;67866;701.46914828;true
RTVolume=701.32;4;1348075490787;67870;701.46913949;true
RTVolume=701.32;2;1348075493802;67872;701.46913497;true
RTVolume=701.29;1;1348075494789;67873;701.46913233;true
```

Chapter 21

Orders

21.1 Order Types and IBAlgos

21.1.1 Order Types

Order Type	Abbreviation
Limit	LMT
Limit Risk	
Bracket	
Market-to-Limit	MTL
Market with Protection	MKT PRT
Request for Quote	QUOTE
Stop	STP
Stop Limit	STP LMT
Trailing Limit if Touched	TRAIL LIT
Trailing Market If Touched	TRAIL MIT
Trailing Stop	TRAIL
Trailing Stop Limit	TRAIL LIMIT
Speed of Execution	
At Auction	
Discretionary	
Market	MKT
Market-if-Touched	MIT
Market-on-Close	MOC
Market-on-Open	MOO
Pegged-to-Market	PEG MKT
Relative	REL
Sweep-to-Fill	
Price Improvement	
Box Top	BOX TOP
Price Improvement Auction	
Block	
Limit-on-Close	LOC
Limit-on-Open	LOO
Limit if Touched	LIT
Pegged-to-Midpoint	PEG MID
Privacy	
Hidden	
Iceberg/Reserve	
VWAP - Guaranteed	VWAP
Time to Market	
All-or-None	
Fill-or-Kill	
Good-after-Time/Date	GAT
Good-till-Date/Time	GTD
Good-till-Canceled	GTC

21.1.2 IBAlgos

Trader Workstation[1]

Beginning with TWS API Release 9.6, the ActiveX, C++ and Java APIs support the following IBAlgo orders for US Stocks and US Options:

US Stocks

- Arrival Price (ArrivalPx)
- Dark Ice (DarkIce)
- Percentage of Volume (PctVol)
- TWAP (Twap)
- VWAP (Vwap)

US Options

- Balance Impact and Risk (BalanceImpactRisk)
- Minimize Impact (MinImpact)

US Products

- Accumulate/Distribute (AD)

The following image lists all of the IBAlgo strategies and parameters supported by the API, except Accumulate/Distribute, which is documented in Accumulate/Distribute (AD).

	Corresponding Parameters									
	maxPctVol	pctVol	strategyType	startTime	endTime	allowPastEndTiime	noTakeLiq	riskAversion	forceCompletion	displaySize
Algo Strategy										
For US Stocks										
Arrival Price	X			X	X	X		X	X	
Dark Ice				X	X	X				X
Percentage of Volume		X		X	X		X			
TWAP			X	X	X	X				
VWAP	X			X	X		X			
For US Options										
Balance Impact and Risk	X							X	X	
Minimize Impact	X									

Arrival Price (ArrivalPx)

Arrival Price Algo[2]

Parameter	Description	Syntax
maxPctVol	Maximum percentage	range: 0.01 0.5
riskAversion	Urgency/Risk aversion	Get Done, Aggressive, Neutral, Passive
startTime	Start time	9:00:00 EST
endTime	End time	15:00:00 EST
forceCompletion	Attempt completion by EOD	0 or 1
allowPastEndTime	Allow trading past end time	0 or 1

[1] https://www.interactivebrokers.com/en/software/tws/twsguide_Left.htm
[2] http://www.interactivebrokers.com/en/trading/orders/arrivalprice.php?ib_entity=llc

Arrival Price Java Code Example:

```
Contract m_contract = new Contract();
Order m_order = new Order();
Vector<TagValue> m_algoParams = new Vector<TagValue>();
/** Stocks */
m_contract.m_symbol = "MSFT";
m_contract.m_secType = "STK";
m_contract.m_exchange = "SMART";
m_contract.m_currency = "USD";
/** Arrival Price */
m_algoParams.add( new TagValue("maxPctVol","0.01") );
m_algoParams.add( new TagValue("riskAversion","Passive") );
m_algoParams.add( new TagValue("startTime","9:00:00 EST") );
m_algoParams.add( new TagValue("endTime","15:00:00 EST") );
m_algoParams.add( new TagValue("forceCompletion","0") );
m_algoParams.add( new TagValue("allowPastEndTime","1") );
m_order.m_action = "BUY";
m_order.m_totalQuantity = 1;
m_order.m_orderType = "LMT";
m_order.m_lmtPrice = 0.14
m_order.m_algoStrategy = "ArrivalPx";
m_order.m_algoParams = m_algoParams;
m_order.m_transmit = false;
m_client.placeOrder(40, m_contract, m_order);
```

Dark Ice (DarkIce)

Dark Ice Algo[3]

Parameter	Description	Syntax
displaySize	Display size	
startTime	Start time	9:00:00 EST
endTime	End time:	15:00:00 EST
allowPastEndTime	Allow trading past end time:	0 or 1

Percentage of Volume (PctVol)

Percentage of Volume Algo[4]

Parameter	Description	Syntax
pctVol	Percentage of volume	range:0.01 0.5
startTime	Start time	9:00:00 EST
endTime	End time:	15:00:00 EST
noTakeLiq	Attempt to never take liquidity	0 or 1

TWAP (Twap)

TWAP Algo[5]

[3] http://www.interactivebrokers.com/en/trading/orders/darkIce.php?ib_entity=llc
[4] http://www.interactivebrokers.com/en/trading/orders/percentofvolume.php?ib_entity=llc
[5] http://www.interactivebrokers.com/en/trading/orders/twapAlgo.php?ib_entity=llc

Parameter	Description	Syntax
strategyType	Trade strategy	Marketable, Matching Midpoint, Matching Same Side, Matching Last
startTime	Start time	9:00:00 EST
endTime	End time	15:00:00 EST
allowPastEndTime	Allow trading past end time:	0 or 1

VWAP (Vwap)

VWAP Algo[6]

Parameter	Description	Syntax
maxPctVol	Maximum percentage of volume	range:0.01 0.5
startTime	Start time	9:00:00 EST
endTime	End time:	15:00:00 EST
allowPastEndTime	Allow trading past end time	0 or 1
noTakeLiq	Attempt to never take liquidity	0 or 1

Balance Impact and Risk (BalanceImpactRisk)

Balance Impact and Risk Algo[7]

Parameter	Description	Syntax
maxPctVol	Maximum percentage of volume	range:0.01 0.5
riskAversion	Urgency/Risk aversion	Get Done, Aggressive, Neutral, Passive
forceCompletion	Attempt completion by EOD	0 or 1
allowPastEndTime	Allow trading past end time	0 or 1

Balance Impact and Risk Java Code Example:

```
Contract m_contract = new Contract();
Order m_order = new Order();
Vector<TagValue> m_algoParams = new Vector<TagValue>();
/** Options */
m_contract.m_symbol = "C";
m_contract.m_secType = "OPT";
m_contract.m_exchange = "SMART";
m_contract.m_localSymbol = "C 110304C00004500";
/** Balance Impact and Risk (OPT) */
m_algoParams.add( new TagValue("maxPctVol","0.1") );
m_algoParams.add( new TagValue("riskAversion","Aggressive") );
m_algoParams.add( new TagValue("forceCompletion","1") );
m_order.m_action = "BUY";
m_order.m_totalQuantity = 1;
m_order.m_orderType = "LMT";
m_order.m_lmtPrice = 0.14;
m_order.m_algoStrategy = "BalanceImpactRisk";
m_order.m_algoParams = m_algoParams;
m_order.m_transmit = false;
m_client.placeOrder(45, m_contract, m_order);
```

Minimize Impact (MinImpact) Minimize Impact[8]

Parameter	Description	Syntax
maxPctVol	Maximum percentage of volume	range:0.01 0.5

[6] http://www.interactivebrokers.com/en/trading/orders/vwapAlgo.php?ib_entity=llc
[7] http://www.interactivebrokers.com/en/trading/orders/balanceimpactrisk.php?ib_entity=llc
[8] http://www.interactivebrokers.com/en/trading/orders/minimizeimpact.php?ib_entity=llc

Accumulate/Distribute (AD) Accumulate Distribute Order Type[9]

TWS Accumulate Distribute[10]

Parameter	Description	Syntax
componentSize	Quantity of increment	Cannot exceed the amount of the initial order
timeBetweenOrders	Time interval	
randomizeTime20	Randomize time period by +/- 20%	”0” or ”1”
randomizeSize55	Randomize size by +/- 55%	”0” or ”1”
giveUp	Number associated with the clearing	
catchUp	Catch up in time:	”0” or ”1”
waitForFill	Wait for current order to fill before submitting next order	”0” or ”1”
startTime	Start time	9:00:00 EST
endTime	End time:	15:00:00 EST

Accumulate Distribute Java Code Example

```
Contract m_contract = newContract();
Order m_order = newOrder();
Vector<TagValue>m_algoParams = new Vector<TagValue>();
/** Stocks */
m_contract.m_symbol = "IBM";
m_contract.m_secType = "STK";
m_contract.m_exchange = "SMART";
m_contract.m_currency = "USD";
/** Accumulate/Distribute (All) */
m_algoParams.add(newTagValue("componentSize", "100"));
m_algoParams.add(newTagValue("timeBetweenOrders", "60"));
m_algoParams.add(newTagValue("randomizeTime20", "1"));
m_algoParams.add(newTagValue("randomizeSize55", "1"));
m_algoParams.add(newTagValue("giveUp", "1"));
m_algoParams.add(newTagValue("catchUp", "1"));
m_algoParams.add(newTagValue("waitForFill", "1"));
m_algoParams.add(newTagValue("startTime", "20110302-14:30:00 GMT"));
m_algoParams.add(newTagValue("endTime", "20110302-21:00:00 GMT"));
m_order.m_action = "BUY";
m_order.m_totalQuantity = 700;
m_order.m_orderType = "LMT";
m_order.m_lmtPrice = 140.0;
m_order.m_algoStrategy = "AD";
m_order.m_tif = "DAY";
m_order.m_algoParams = m_algoParams;
m_order.m_transmit = false;
m_client.placeOrder(orderId++, m_contract, m_order);
```

[9] http://www.interactivebrokers.com/en/trading/orders/accumulatedistribute.php?ib_entity=llc
[10] https://www.interactivebrokers.com/en/software/tws/usersguidebook/algos/accumulate_distribute.htm

21.1.3 CSFB Algo Parameters

The ActiveX, C++ and Java APIs support the following CSFB algo strategies:

- Crossfinder
- Float
- Guerilla
- Work It IW
- Work It
- Pathfinder
- Reserve
- Strike
- 10B 18
- Tex
- TWAP
- VWAP

The following image lists all of the CSFB algo strategies and parameters supported by the API.

Trader Workstation[11]

Look under Algos > IB Algos

Look under Algos > CSFB Algos

Look under Algos > Jefferies Algos

[11] https://www.interactivebrokers.com/en/software/tws/twsguide_Left.htm

21.2 Extended Order Attributes

The extended order attributes below can be used in all placeOrder functions and Open_Order events.

Attribute	Key	Possible Values
m_tif	string	Day, GTC, IOC, GTD
m_ocaGroup	string	Identifies a member of a one-cancels-all group.
m_account	string	Institutional only.
m_openClose	string	Institutional only.
m_origin	int	Institutional only.
m_orderRef	string	Customer defined order ID tag.
m_transmit	boolean	Specifies whether the order will be transmitted by TWS. If set to false, order is created by not transmitted.
m_parentId	int	The order ID of the parent, used for bracket, auto stop and trailing stop orders.
m_blockOrder	boolean	If set to true, specifies that the order is a block order.
m_sweep-ToFill	boolean	If set to true, specifies that the order is a Sweep-to-fill order.
m_displaySize	int	The publicly disclosed order size to be used when placing iceberg orders.
m_triggerMethod	int	Specifies how simulated Stop, Stop-Limit, and Trailing Stop orders are triggered: • 0 - the default value. The ”double bid/ask” method will be used for orders for OTC stocks and US options. All other orders will use the ”last” method. • 1 - use ”double bid/ask” method, where stop orders are triggered based on two consecutive bid or ask prices. • 2 - ”last” method, where stop orders are triggered based on the last price. • 3 - ”double-last” method, where stop orders are triggered based on last two prices. • 4 - bid-ask method. For a buy order, a single occurrence of the bid price must be at or above the trigger price. For a sell order, a single occurrence of the ask price must be at or below the trigger price. • 7 - last-or-bid-ask method. For a buy order, a single bid price or the last price must be at or above the trigger price. For a sell order, a single ask price or the last price must be at or below the trigger price. • 8 - mid-point method, where the midpoint must be at or above (for a buy) or at or below (for a sell) the trigger price, and the spread between the bid and ask must be less than 0.1% of the midpoint. For a complete description of Trigger Methods, see Modify the Trigger Method[12] in the Trader Workstation Users’ Guide.
m_ignoreRth	boolean	If set to true, allows triggering of orders outside of regular trading hours.
m_hidden	boolean	If set to true, the order will not be visible when viewing the market depth. The only applies to orders routed to INet.
m_goodAfter-Time	string	Indicates that the trade should be submitted after the time and date set, with format YYYYMMDD HH:MM:SS (seconds are optional). Use an empty string if not applicable.
m_goodTillDate	string	Indicates that the trade should remain working until the time and date set, with format YYYYMMDD HH:MM:SS (seconds are optional). You must set the tif to GTD when using this string. Use an empty string if not applicable.
m_faGroup	string	The advisor group to which the trade will be allocated. Use an empty string if not applicable.
m_faProfile	string	The advisor allocation profile to which the trade will be allocated. Use an empty string if not applicable.
m_faMethod	string	The advisor allocation method with which the trade will be allocated. Use an empty string if not applicable.

Attribute	Key	Possible Values
m_faPercentage	string	The advisor percentage concerning the trade's allocation. Use an empty string if not applicable.
m_primaryExch	string	To clarify any ambiguity for Smart-routed contracts, include the primary exchange, along with the Smart designation, for the destination.
m_shortSaleSlot	int	For institutional customers only. • 0 - unapplicable (i.e. retail customer or not sshort leg) • 1 - clearing broker • 2 - third party. If this value is used, you must enter a designated location.
m_designatedLocation	string	Only valid when shortSaleSlot value = 2. Otherwise leave blank or orders will be rejected.
ocaType	long	Cancel on Fill with Block = 1 Reduce on Fill with Block = 2 Reduce on Fill without Block = 3
rthOnly	int	Regular trading hours only. yes=1, no=0
rule80A	String	Individual = 'I' Agency = 'A', AgentOtherMember = 'W' IndividualPTIA = 'J' AgencyPTIA = 'U' AgentOtherMemberPTIA = 'M' IndividualPT = 'K' AgencyPT = 'Y' AgentOtherMemberPT = 'N'
settlingFirm	String	Institutional only
clearingAccount	String	The true beneficiary of the order. This value must be sent on FUT/FOP orders for reporting the exchange.
clearingIntent	String	IB, Away, or PTA
allOrNone	int	yes=1, no=0
minQty	long	Identifies a minimum quantity order type.
percentOffset	double	The percent offset for relative orders.
eTradeOnly	int	Trade with electronic quotes. yes=1, no=0
firmQuoteOnly	int	Trade with firm quotes. yes=1, no=0
nbboPriceCap	double	Maximum SMART order distance from the NBBO.
auctionStrategy	long	match = 1 improvement = 2 transparent = 3 For BOX exchange only.
startingPrice	double	Starting price. For BOX exchange only.
stockRefPrice	double	The stock reference price. For BOX exchange only.
delta	double	For BOX exchange only.
stock-RangeLower	double	The lower value of the acceptable stock range. For BOX exchange only.
stock-RangeUpper	double	The upper value of the acceptable stock range. For BOX exchange only.
m_volatility	double	The option price in volatility, as calculated by TWS' Option Analytics. This value is expressed as a percent and is used to calculate the limit price sent to the exchange.
m_volatilityType	int	1 = Daily; 2 = Annual
m_continuousUpdate	int	0 = false; 1 = True
m_referencePriceType	int	1 = Average; 2 = BidorAsk
m_deltaNeutralOrderType	String	Enter an accepted order type such as: MKT, LMT, REL etc.
m_deltaNeutralAuxPrice	double	Enter the Aux Price for Hedge Delta order types that require one.

Attribute	Key	Possible Values
m_scaleNumComponents	int	For Scale orders: Defines the number of component orders into which the parent order will be split, thereby backing into the number of units within each component.
m_scaleComponentSize	int	For Scale orders: Defines the number of units per component, backing into the number of components into which the parent order is split.
m_scalePriceIncrement	double	For Scale orders: Defines the price increment per scale component.
m_basisPoints	double	EFP orders
basisPointsType	int	EFP orders

21.3 Order Status for Partial Fills

The following example demonstrates how the orderStatus() event behaves when there is a partial fill of an order.

Partial Fill Example

You place an order for 1000 shares of XYZ stock. There are four separate executions before the order for 1000 total shares is completed. The first partial fill executes with 200, then the second partial fill executes with 200 shares. The third partial fill executes with another 200 shares and finally, the last partial fill executes with 400 shares.

- First execution: 200
- Second execution: 200
- Third execution: 200
- Fourth execution: 400

In the orderStatus() event, here is the sequence of status messages that will be received by the API based on this example:

Market Scanner (Scan Code)	Status Message
1st Execution	Status = Submitted Filled qnty = 200 Remaining qnty = 800
2nd Execution	Status = Submitted Filled qnty = 400 Remaining qnty = 600
3rd Execution	Status = Submitted Filled qnty = 600 Remaining qnty = 400
4th Execution	Status = Filled Filled qnty = 1000 Remaining qnty = 0

Chapter 22

Misc

22.1 Available Market Scanners

The following table shows a list (current as of July 2008) of the available scanners.

Execution	Description
Low Opt Volume P/C Ratio (LOW_OPT_VOL_PUT_CALL_RATIO)*	Put option volumes are divided by call option volumes and the top underlying symbols with the lowest ratios are displayed.
High Option Imp Vol Over Historica (HIGH_OPT_IMP_VOLAT_OVER_HIST)*	Shows the top underlying contracts (stocks or indices) with the largest divergence between implied and historical volatilities.
Low Option Imp Vol Over Historical (LOW_OPT_IMP_VOLAT_OVER_HIST)*	Shows the top underlying contracts (stocks or indices) with the smallest divergence between implied and historical volatilities.
Highest Option Imp Vol (HIGH_OPT_IMP_VOLAT)*	Shows the top underlying contracts (stocks or indices) with the highest vega-weighted implied volatility of near-the-money options with an expiration date in the next two months.
Top Option Imp Vol % Gainers (TOP_OPT_IMP_VOLAT_GAIN)*	Shows the top underlying contracts (stocks or indices) with the largest percent gain between current implied volatility and yesterday's closing value of the 15 minute average of implied volatility.
Top Option Imp Vol % Losers (TOP_OPT_IMP_VOLAT_LOSE)*	Shows the top underlying contracts (stocks or indices) with the largest percent loss between c urrent implied volatility and yesterday's closing value of the 15 minute average of implied volatility.
High Opt Volume P/C Ratio (HIGH_OPT_VOLUME_PUT_CALL_RATIO)	Put option volumes are divided by call option volumes and the top underlying symbols with the highest ratios are displayed.
Low Opt Volume P/C Ratio (LOW_OPT_VOLUME_PUT_CALL_RATIO)	Put option volumes are divided by call option volumes and the top underlying symbols with the lowest ratios are displayed.
Most Active by Opt Volume (OPT_VOLUME_MOST_ACTIVE)	Displays the most active contracts sorted descending by options volume.
Hot by Option Volume (HOT_BY_OPT_VOLUME)	Shows the top underlying contracts for highest options volume over a 10-day average.
High Option Open Interest P/C Ratio (HIGH_OPT_OPEN_INTEREST_PUT_CALL_RATIO)	Returns the top 50 contracts with the highest put/call ratio of outstanding option contracts.
Low Option Open Interest P/C Ratio (LOW_OPT_OPEN_INTEREST_PUT_CALL_RATIO)	Returns the top 50 contracts with the lowest put/call ratio of outstanding option contracts.
Top % Gainers (TOP_PERC_GAIN)	Contracts whose last trade price shows the highest percent increase from the previous night's closing price.
Most Active (MOST_ACTIVE)	Contracts with the highest trading volume today, based on units used by TWS (lots for US stocks; contract for derivatives and non-US stocks). The sample spreadsheet includes two Most Active scans: Most Active List, which displays the most active contracts in the NASDAQ, NYSE and AMEX markets, and Most Active US, which displays the most active stocks in the United States.
Top % Losers (TOP_PERC_LOSE)	Contracts whose last trade price shows the lowest percent increase from the previous night's closing price.
Hot Contracts by Volume (HOT_BY_VOLUME)	Contracts where: • today's Volume/avgDailyVolume is highest. • avgDailyVolume is a 30-day exponential moving average of the contract's daily volume.
Top % Futures Gainers (TOP_PERC_GAIN)	Futures whose last trade price shows the highest percent increase from the previous night's closing price.
Hot Contracts by Price (HOT_BY_PRICE)	Contracts where: • (lastTradePrice-prevClose)/avgDailyChange is highest in absolute value (positive or negative). • The avgDailyChange is defined as an exponential moving average of the contract's (dailyClose-dailyOpen)

Execution	Description
Top Trade Count (TOP_TRADE_COUNT)	The top trade count during the day.
Top Trade Rate (TOP_TRADE_RATE)	Contracts with the highest number of trades in the past 60 seconds (regardless of the sizes of those trades).
Top Price Range (TOP_PRICE_RANGE)	The largest difference between today's high and low, or yesterday's close if outside of today's range.
Hot by Price Range (HOT_BY_PRICE_RANGE)	The largest price range (from Top Price Range calculation) over the volatility.
Top Volume Rate (TOP_VOLUME_RATE)	The top volume rate per minute.
Lowest Option Imp Vol (LOW_OPT_IMP_VOLAT)	Shows the top underlying contracts (stocks or indices) with the lowest vega-weighted implied volatility of near-the-money options with an expiration date in the next two months.
Most Active by Opt Open Interest (OPT_OPEN_INTEREST_MOST_ACTIVE)	Returns the top 50 underlying contracts with the (highest number of outstanding call contracts) + (highest number of outstanding put contracts)
Not Open (NOT_OPEN)	Contracts that have not traded today.
Halted (HALTED)	Contracts for which trading has been halted.
Top % Gainers Since Open (TOP_OPEN_PERC_GAIN)	Shows contracts with the highest percent price INCREASE between the last trade and opening prices.
Top % Losers Since Open (TOP_OPEN_PERC_LOSE)	Shows contracts with the highest percent price DECREASE between the last trade and opening prices.
Top Close-to-Open % Gainers (HIGH_OPEN_GAP)	Shows contracts with the highest percent price INCREASE between the previous close and today's opening prices.
Top Close-to-Open % Losers (LOW_OPEN_GAP)	Shows contracts with the highest percent price DECREASE between the previous close and today's opening prices.
Lowest Option Imp Vol (LOW_OPT_IMP_VOLAT)	Shows the top underlying contracts (stocks or indices) with the lowest vega-weighted implied volatility of near-the-money options with an expiration date in the next two months.
Top Option Imp Vol % Gainers (TOP_OPT_IMP_VOLAT_GAIN)	Shows the top underlying contracts (stocks or indices) with the largest percent gain between current implied volatility and yesterday's closing value of the 15 minute average of implied volatility.
Top Option Imp Vol % Losers (TOP_OPT_IMP_VOLAT_LOSE)*	Shows the top underlying contracts (stocks or indices) with the largest percent loss between current implied volatility and yesterday's closing value of the 15 minute average of implied volatility.
13-Week High (HIGH_VS_13W_HL)	The highest price for the past 13 weeks.
13-Week Low (LOW_VS_13W_HL)	The lowest price for the past 13 weeks.
26-Week High (HIGH_VS_26W_HL)	The highest price for the past 26 weeks.
26-Week Low (LOW_VS_26W_HL)	The lowest price for the past 26 weeks.
52-Week High (HIGH_VS_52W_HL)	The highest price for the past 52 weeks.
52-Week Low (LOW_VS_52W_HL)	The lowest price for the past 52 weeks.
EFP - High Synth Bid Rev Yield (HIGH_SYNTH_BID_REV_NAT_YIELD)	Highlights the highest synthetic EFP interest rates available. These rates are computed by taking the price differential between the SSF and the underlying stock and netting dividends to calculate an annualized synthetic implied interest rate over the period of the SSF. The High rates may present an investment opportunity.
EFP - Low Synth Bid Rev Yield (LOW_SYNTH_BID_REV_NAT_YIELD)	Highlights the lowest synthetic EFP interest rates available. These rates are computed by taking the price differential between the SSF and the underlying stock and netting dividends to calculate an annualized synthetic implied interest rate over the period of the SSF. The Low rates may present a borrowing opportunity.

Instruments and Location Codes for Market Scanners

Market scanners in the TWS API support the following instruments and location codes:

Instruments

- STK - US stocks
- STOCK.HK - Asian stocks
- STOCK.EU - European stocks

Location Codes

- STK.US - US stocks
- STK.US.MAJOR - US stocks (without pink sheet)
- STK.US.MINOR - US stocks (only pink sheet)
- STK.HK.SEHK - Hong Kong stocks
- STK.HK.ASX - Australian Stocks
- STK.EU - European stocks

22.2 Supported Time Zones

The following table shows a list of time zones supported by the TWS API.

Time Zone	Description
GMT	Greenwich Mean Time
EST	Eastern Standard Time
MST	Mountain Standard Time
PST	Pacific Standard Time
AST	Atlantic Standard Time
JST	Japan Standard Time
AET	Australian Standard Time

22.3 Smart Combo Routing

These features are for both guaranteed and non-guaranteed combination orders routed to Smart, and are available based on the combo type and order type. For example, users can specify the maximum size to submit at one time (Maximum leg-in combo size) and which leg should be legged in first.

Smart Combo Routing is also supported in the Active X, C++ and Java APIs through the use of **smartComboRoutingParams**, which requires TWS server version 57 or higher. **smartComboRoutingParams** is similar to AlgoParams in that it makes use of tag/value pairs to add parameters to combo orders. The parameters cover the following capabilities:

- **Priority** - User can specify which leg to be executed first.
  Tag = LeginPrio
  Values = -1, 0 or 1
- **Discretionary Amount** - When one leg is executed, we can adjust the other leg by up to a discretionary amount.
  Tag = MaxSegSize
  Value = An amount
- **Market-If-Touched Timeout** - For Market-If-Touched combo orders, we record the firstTradeTime of the first fill of the first leg to execute, and the lastTradeTime of the last partial fill. For these kinds of orders, you can now specify timeout values of the last fill and the timeout since the first fill, in seconds.
  Tags = ChangeToMktTime1 is the timeout after the last fill, and ChangeToMktTime2 is the timeout after the first fill.
  Value = Number of seconds
- **Market-If-Touched Stop-Loss** - Specify an absolute stop-loss amount per combo. If specified and if the implied execution price of the combo (based on a leg that has already been executed and current market data) exceeds the combo price plus the stop-loss amount, we convert the order from LMT to MKT immediately in order to finish executing the combo order. If the stop-loss amount is specified but timeouts have not been specified, we will continue to try to execute the second leg at the calculated LMT price until it either executes or the stop-loss amount is reached.
  Tag = ChangeToMktOffset
  Value = An amount.
- **Maximum Leg-In Size** - Specify the maximum allowed leg-in size per segment.
  Tag = MaxSegSize
  Value = Unit of combo size
- **Discretionary Percentage** - Specify a percentage of the combo price. This applies to scale combos in which the discretionary amount is calculated from the current scale level. When the discretionary amount is entered as a percentage, the API converts it to a dollar amount based on the combo. This amount will be updated when the order price changes or for scale orders for each level. You can enter a value for this parameter or for the Discretionary Amt extended attribute one at a time, but not both at the same time.
  Tag = DiscretionaryPct
  Value = A value between 0 and 100.

22.4 API Logging

As client requests are processed (both system and API clients) it logs certain information to its 'log.txt' log file, which is located in the installation directory. The purpose of this file is to help resolve problems by providing some insight into the state of the program before the problem occurred.

API clients can specify how detailed they want these log entries to be by setting the log level. Log levels are:

- 1 = SYSTEM
- 2 = ERROR
- 3 = WARNING
- 4 = INFORMATION
- 5 = DETAIL

Note: Setting the log level to 5 will increase performance overhead. You should only use log level 5 when you are trying to resolve an issue.

The log entries for API requests have the format:

[ClientID: ClientVersion:ServerVersion:ClientType:Request:Response:Version:LogEntryType]

where:

- **ClientID** is the clientId used when connecting.
- **ClientVersion** identifies the client's request stream (for internal use).
- **ServerVersion** identifies the server's response stream (for internal use).
- **ClientType** is the type of API connection: DDE = 0, Socket = 1.
- **Request:** If greater than 0, indicates that the log entry is the result of an API client request. The number shown is the request identifier as listed in the "Outgoing Request Identifiers" section below.
- **Response:** If greater than 0, indicates that the log entry is the result of a server response to the API. The number shown is the response identifier as listed in the "Incoming Response Identifiers" section below.
- **Version** identifies the version of the request or response message. The version changes when the message format changes.
- **LogEntryLevel** identifies the type of log entry (i.e. the log level as listed above)

Example Log Entry

[0:9:9:1:1:0:3:DET]Socket request - [3;52;IBM;STK;null;0.0;2;SMART;null;null]

From this example, we can tell that a socket client with clientId=0 connected and made a request for market data. The version of the market data request, which was 3, implies what data should have been sent.

API Request/Server Response Message Identifiers

Outgoing Request Identifiers	Incoming Response Identifiers
1 = Request Market Data	1 = Ticker Price
2 = Cancel Market Data	2 = Ticker Size
3 = Place Order	3 = Order Status
4 = Cancel Order	4 = Error Message
5 = Request Open Orders	5 = Open Order
6 = Request Account Data	6 = Account Value
7 = Request Execution Reports	7 = Portfolio Value
8 = Request Next Order Id	8 = Account Update Time
9 = Request Contract Details	9 = Next Valid Order Id
10 = Request Market Depth	10 = Contract Details
11 = Cancel Market Depth	11 = Execution Report Details
12 = Request News Bulletins	12 = NYSE Open Book Row Entry
13 = Cancel News Bulletins	13 = Level II Quotes Row Entry
14 = Set Server Log Level	14 = News Bulletin

Note: This information, along with the various request/response message versions, can be found in the EClientSocket implementation file supplied with the API installation.

22.5 Requests for Quotes (RFQs)

RFQs from the IB Options Trading Desk allow you to get quotes for large orders from IB affiliate Timber Hill. Quotes are available for US equity and index options, and major European and Asian index options and combinations. For a complete list, please contact the IB Options Trading Desk.

RFQs from the IB Options Trading Desk are available only to users who have access to these specific areas. Please contact the IB Options Trading Desk if you are interested in participating.

Submitting RFQs using the API

Submit an RFQ by submitting an order with an order type of QUOTE. In the response, tickPrice()/tickSize() are called with the tickerId matching the orderId of the RFQ. Use orderId's with a relatively high number to avoid clashes. Additional space is required for non-RFQ tickerIds.

Market data for an RFQ is received until the user cancels the RFQ or the RFQ is canceled by the server. The server normally cancels an RFQ when it expires (approximately 1 minute) or if the RFQ request is invalid and/or for an unsupported product.

Delta-Neutral RFQs

Submit Delta-Neutral RFQs by creating a combo order, even if a single contract must be hedged, and filling up and attaching an UnderComp structure to a contract underComp field.

In the UnderComp structure, you must specify the conId of the hedge contract. The price and delta fields can be left empty (0).

Upon accepting a Delta-Neutral DN RFQ, the server sends a deltaNeutralValidation() message with the UnderComp structure. If the delta and price fields are empty in the original request, the confirmation will contain the current values from the server. These values are locked when the RFQ is processed and remain locked until the RFQ is canceled.

RFQ Samples

To learn more about submitting RFQs with the TWS API, look at the RFQ samples included in the 9.6 release of the API software. The samples are located in the samples/rfq folder in your API software installation folder. The SampleRFQ.java sample implements a small-state machine and shows how to submit RFQ's for:

- EU Stocks
- US Futures
- US Stock Options
- EU Stock Options
- Calendar Spread for Index Option (Delta-Neutral)
- US Stock Option (Delta-Neutral)
- US Index Option (Delta-Neutral)
- EU Index Option (Delta-Neutral)

22.6 Support for Mini Options

Support for Mini Options - ActiveX, Java and C++ APIs

You can identify mini options in the Active X, Java and C++ APIs by providing the multiplier or trading class in both requests to TWS and responses from TWS.

The following requests and callbacks that include the Contract structure as a parameter can now use the tradingClass and multiplier attributes to identify mini options. Also, some of these requests can now use the conId attribute to identify a security (these are indicated below).

Requests

- reqMktData
- reqHistoricalData - also conId
- reqRealTimeBars - also conId
- reqContractDetails
- reqMktDepth - also conId
- exerciseOptions - also conId
- placeOrder
- calculateImpliedVolatility
- calculateOptionPrice

Callbacks

- openOrder
- updatePortfolio
- execDetails
- position

Note the following:

- multiplier is encoded/decoded after contract.right and before contract.exchange in all requests and callbacks.
- conId is encoded after tickerId/reqId and before contract.symbol in all requests and callbacks.
- tradingClass is encoded/decoded after contract.localSymbol in all requests and callbacks.

The **reqFundamentalData** request, available for stocks only, can also handle the conid attribute in the Contract structure but not tradingClass or multiplier.

22.7 Requesting Real-Time Index Premium Data

You can request real-time Index Premium market data using the following APIs and API sample applications:

- ActiveX (including the ActiveX API sample application)
- C++ (including the C++ API sample application)
- Java (including the Java API sample application)
- ActiveX for Excel

To request real-time Index Premium data, you must do the following:

- Specify the Symbol, Security Type and Exchange.
- For example, INDU, IND and NYSE would get you Index Premium data for the Dow Jones Industrial Average.
- The exchange must match the index for which you want data.
- You must use the generic tick type 162 (for Index Future Premium).

22.8 Requesting News

You can receive news top news for underlying contracts from TWS for news feeds to which you have subscribed (in Account Management) using generic tick type 292. The following examples illustrate the different ways you can request news from TWS.

Note that we use the Java API method/parameters in the examples below, but you can substitute the correct method/ parameters for your API language of choice.

To request news for IBM

Request market data with the following parameters:

```
symbol=IBM
secType=STK
exchange=SMART
currency=USD
genericTicklist="mdoff,292"
```

Note: mdoff indicates that top market data will not tick.

or

Request market data with the following parameters:

```
conId=8314
secType=[empty]
exchange=SMART
currency=[empty]
genericTicklist="mdoff,292"
```

To request only Fly On The Wall (FLY) News for IBM

Request market data with the following parameters:

```
symbol=IBM
secType=STK
exchange=SMART
currency=USD
genericTicklist="mdoff,292:FLY"
```

To request only Fly On The Wall and Briefing.com (BRF) news for IBM

Request market data with the following parameters:

```
symbol=IBM
secType=STK
exchange=SMART
currency=USD
genericTicklist="mdoff,292:FLY+BRF"
```

To request top data and news for IBM

Request market data with the following parameters:

```
symbol=IBM
secType=STK
exchange=SMART
currency=USD
genericTicklist="292"
```

To request a list of news topics

Request contract data with the following parameters:

```
symbol=*
secType=NEWS
exchange=FLY or NEWS:FLY
currency=[empty]
```

To request Reuters European Union News

Request market data with the following parameters:

```
symbol=FLY:EU
secType=NEWS
exchange=FLY or NEWS:FLY
currency=[empty]
genericTicklist="292"
```

or

Request market data with the following parameters:

```
conId=6145497
symbol=[empty]
secType=[empty]
exchange=NEWS
currency=[empty]
genericTicklist="292"
```

Part III

Interactive Brokers Customer Interface

Chapter 23

The Account Windows

23.1 The Account Window

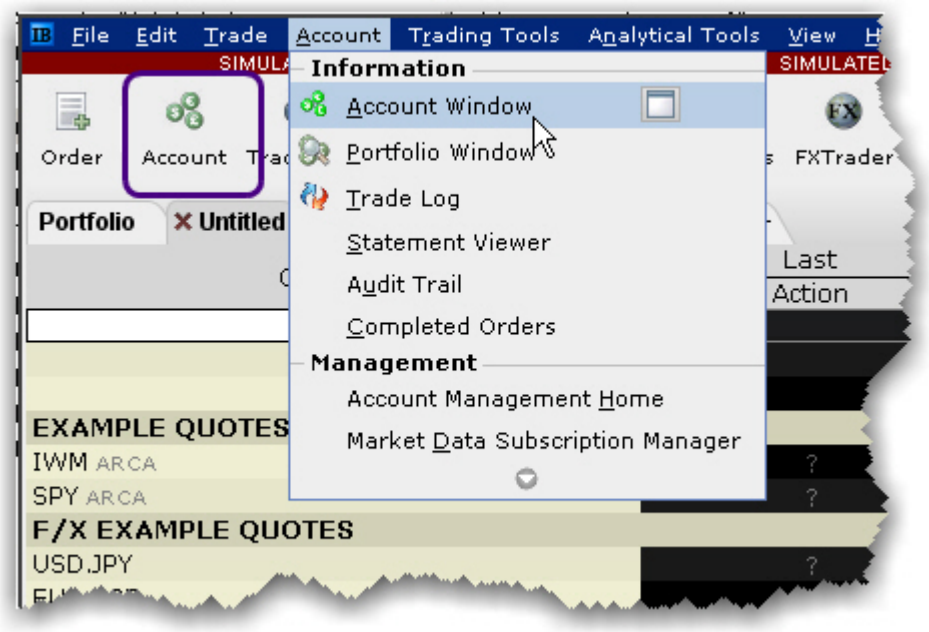
The Account window lets you monitor every aspect of your account activity. We present ”key” account values as the default when you first open the Account window (this default view is shown in the illustration below). The account window conveys in real-time values the funds you have available for additional trades and current margin projections.

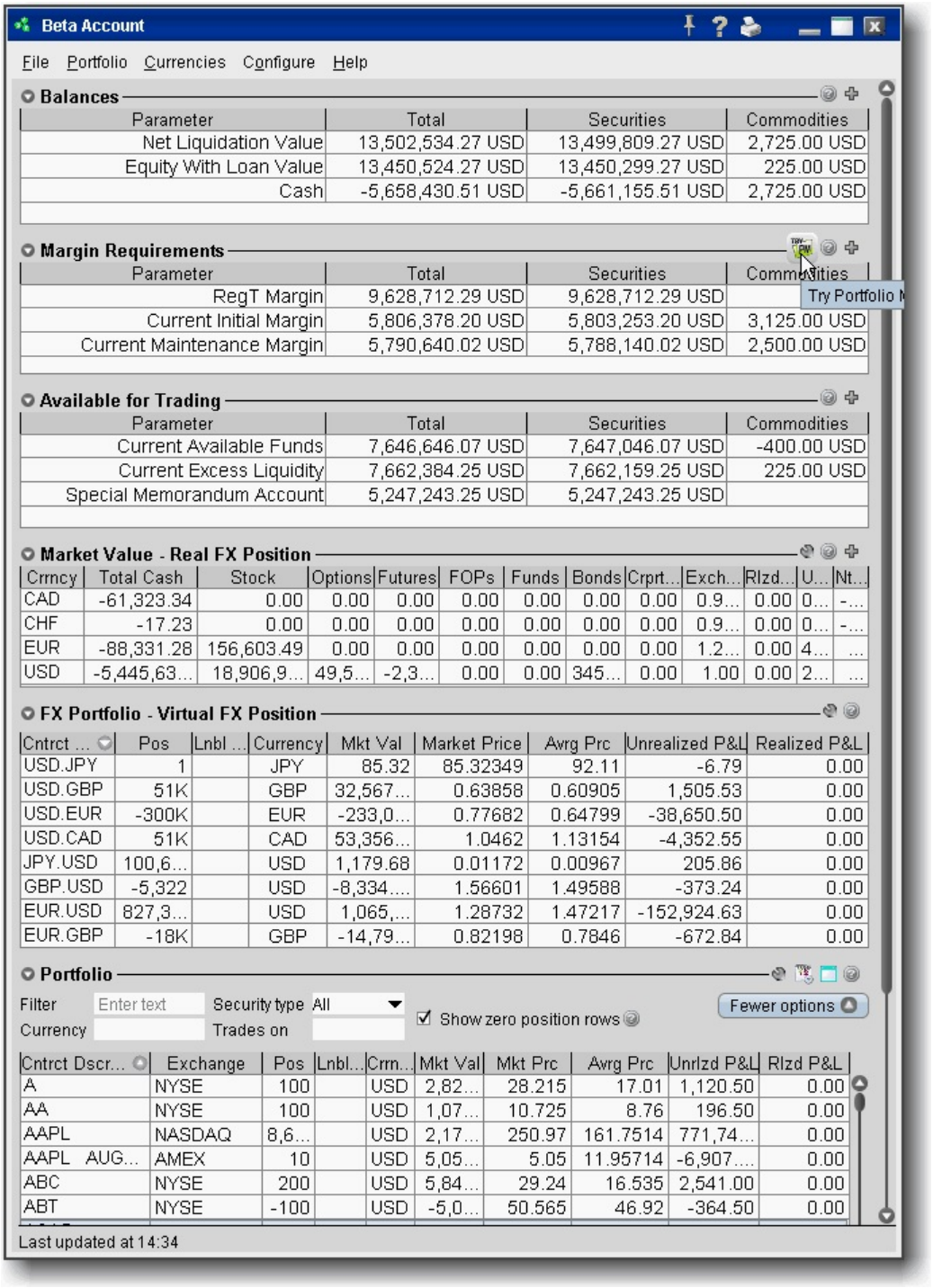
The Account screen is divided into the following sections:

- **Balances**, which shows your account balances for securities, commodities and in total. These balances don’t include margin requirements.
- **Margin Requirements**, which shows your current initial and maintenance margin requirements based on your current positions. Customers under the Reg T margin model can compare their current margin to current projected requirements under the Portfolio Margin model by clicking the Try PM icon.
- **Available for Trading**, which tells you what you have for additional trades and cushion before liquidation, based on your balance information and margin requirements.
- **Market Value - Real FX Position**, which shows the total value of all positions, sorted by currency.
- **FX Portfolio - Virtual FX Position**, which shows activity for currency pair trades only. The value in the Position field only reflects trades executed in the FX market. Please note that these may not reflect real-time FX positions, since non-base currency conversions may also be included.
- **Portfolio**, which lists all current positions sorted by underlying, and displays the total current market value for each position.

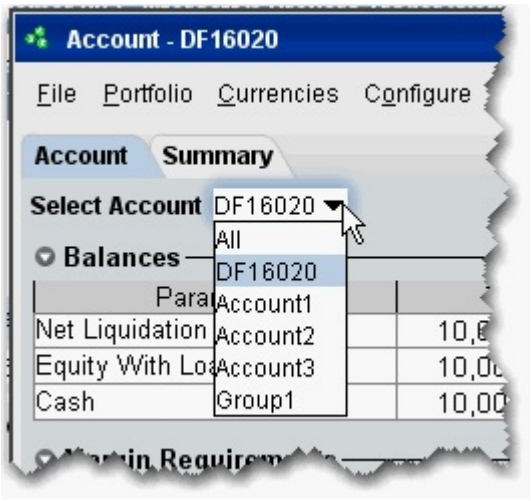
To open the Account window

- On the main trading Toolbar, click the Account icon, or from the Account menu select Account Window.

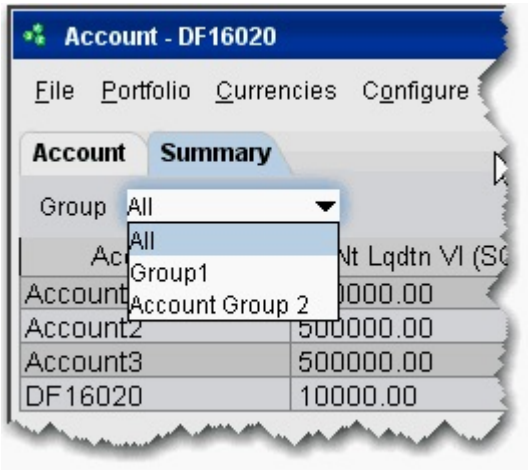

File
Edit
Trade
Account
Trading Tools
Analytical Tools
View
Information
Account Window
Portfolio Window
Trade Log
Statement Viewer
Audit Trail
Completed Orders
Management
Account Management Home
Market Data Subscription Manager
Order
Account
FXTrader
Portfolio
Untitled
Last
Action
EXAMPLE QUOTES
IWM ARCA
SPY ARCA
F/X EXAMPLE QUOTES
USD.JPY



Advisors and other multi-client account holders will see a drop-down selection on the Account and Summary tabs to select either an individual account or an account group.



On the Account page, elect to display account information for All accounts, any individual account including the Master, or a user-defined Account Group, which includes a subset of accounts (in the image above, **Group 1** is a user-defined account groups).



On the Summary page, choose to view the account summary for All accounts, or for any Account Group.

Advisors can also print the summary page by selecting *Print Summary* from the File menu.

23.2 Balances

The Balances section shows your account balances for securities, commodities and in total. These balances don't include margin requirements.

To view all Balances values, expand the section using the green plus (+) sign. To customize this view to show only the values you want to see, put a check next to these parameters and click the green minus (-) sign to contract the section.

Balances

Parameter	Total	Securities	Commodities	
Net Liquidation Value	10,089,200.77 USD	10,088,634.09 USD	566.68 USD	☑
Equity With Loan Value	10,088,554.09 USD	10,088,487.41 USD	66.68 USD	☑
Previous Day Equity with Loan Value	10,084,049.16 USD	10,084,049.16 USD		☐
Reg T Equity with Loan Value	10,088,487.41 USD	10,088,487.41 USD		☐
Special Memorandum Account	9,994,052.41 USD	9,994,052.41 USD		☐
Securities Gross Position Value	605,440.37 USD	207,752.32 USD		☐
Cash	9,922,148.99 USD	9,921,582.31 USD	566.68 USD	☑
Settled Cash				☐
Accrued Interest	17,391.10 USD	17,391.10 USD	0.00 USD	☐

The following table defines the fields available in the Balances section.

Parameter	Securities	Commodities
Net Liquidation Value	Total cash value + stock value + securities options value + bond value + fund value.	Total cash value (which includes futures P&L) + commodities options value.
Equity with Loan Value	**Cash Account**: Settled Cash. **Margin Account**: Total cash value + stock value + bond value + fund value + European & Asian options value.	**Cash Account**: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures P&L). **Margin Account**: Total cash value + commodities option value - futures maintenance margin requirement.
Previous Day Equity with Loan Value	Marginable Equity with Loan Value as of 16:00 ET the previous day.	Not applicable.
Reg T Equity with Loan Value	**Cash Account**: Settled Cash. **Margin Account**: Total cash value + stock value + bond value + fund value + non-U.S. & Canadian securities options value + accrued interest + accrued dividend.	**Cash Account**: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures P&L). **Margin Account**: Total cash value - futures maintenance margin requirement.

Parameter	Securities	Commodities
Special Memorandum Account. If this values falls below zero at the end of the trading day, positions will be liquidated.	A special account associated with a Reg T Margin account that is maintained for the purpose of applying Federal Regulation T initial margin requirements at the end of the trading day. Max ((EWL - US initial margin requirements)*, (Prior Day SMA +/- change in day's cash +/- US initial margin requirements** for trades made during the day.)) *calculated end of day under US Stock rules, regardless of country of trading. **at the time of the trade	Not applicable.
Securities Gross Position Value (GPV)	Long Stock Value + Short Stock Value + Long Option Value + Short Option Value + Fund Value	
Cash	Settled cash + sales at the time of trade + futures P&L	Settled cash + sales at the time of trade + futures P&L
Settled Cash	Cash recognized at the time of settlement - purchases at the time of trade - commissions - taxes - fees. Stock Settlement: Trade date + 3 days Options Settlement: Trade date + 1 day	Same as for Securities. Futures Settlement: Trade date + 1 day
Accrued Interest	Interest that has accumulated but has not been paid or charged.	Same as for Securities.
Dividend Payables	Dividends in Lieu that must be paid by the holder of a short position.	

23.3 Margin Requirements

Use real-time margin monitoring to see your current margin requirements at a glance, and to understand the margin implications of any transaction before you transmit an order.

To view real-time margin requirements

1. From the Account menu, select Account Window.
2. The Account Information window shows your account details, including the current market value and portfolio.
3. Use the Margin Requirements section to monitor your margin.

23.4 Available for Trading

The Available for Trading values show you the state of your account at a glance. Your Available Funds tells you if you can put on a trade. Your Excess Liquidity tells you whether you have sufficient cushion to maintain your current positions, your Buying Power tells you how much you have at your disposal including your equity and IB's margin.

To view all Available for Trading values, expand the section using the green plus (+) sign. To customize this view to show only the values you want to see, put a check next to these parameters and click the green minus (-) sign to contract the section.

23.5 Market Value - Real FX Position

The Market Value section shows you total value for all assets sorted by currency. Accounts with the ability to transfer between brokers will also have an In Transit tab to monitor DVPs and other position transfers.

23.6 FX Portfolio - Virtual FX Position

This section is designed to show activity for currency pair trades for FX traders so that they can track average cost and running P&L on their currency trades. However the ”Position” value may reflect the sum of trades executed in the FX market along with currency conversions to change non-base funds into your base currency.
Consequently these “virtual” positions do not necessarily reflect an actual cash balance in any currency.

For actual, real-time currency balance, check the Total Cash field of the Market Value section.

- To avoid having currency conversions reflected in this panel, use the FXCONV destination on the order line.
- When the FX Portfolio panel is displayed, currency positions will also be displayed in the trading window. To remove currency positions from display, condense the FX Portfolio panel by clicking the arrow to the left of the panel title.

23.7 Portfolio

The Portfolio section shows the current value of each position and allows you to mark any position to be liquidated last in the event of a margin call. Accounts with the ability to make transfers between brokers will also have an In Transit tab to monitor DVPs and other position transfers.

Please note that TWS displays all FA customer positions in the Account window on a net/aggregate basis.

Part IV

Python

Chapter 24

Quck Reference

How do I look inside a Python object? [1]

Python has a strong set of introspection features. Take a look at the following built-in functions:

- `type()`
- `dir()`
- `id()`
- `getattr()`
- `hasattr()`
- `globals()`
- `locals()`
- `callable()`

type() and dir() are particularly useful for inspecting the type of an object and its set of attributes, respectively.

```
object.__dict__
dir()
help()
```

Python `inspect()`

[1] `http://stackoverflow.com/questions/1006169/how-do-i-look-inside-a-python-object`

Part V

IbPy Summary EClientSocket and EWrapper Methods

Chapter 25

Summary

Groups	Methods	Specs
Account and Portfolio	EClientSocket	**reqAccountUpdates**(boolean subscribe, String acctCode) **reqAccountSummary**(int reqId, String group, String tags) **cancelAccountSummary**(int messageId, int version As Integer, int reqId As Integer) **reqPositions**() **cancelPositions**()
Account and Portfolio	EWrapper	**updateAccountValue**(String key, String value, String currency, String accountName) **updatePortfolio**(Contract contract, int position, double marketPrice, double marketValue, double averageCost,double unrealizedPNL, double realizedPNL, String accountName) **updateAccountTime**(String timeStamp) **accountDownloadEnd**(String accountName) **accountSummary**(int reqId, String account, String tag, String value, String currency) **accountSummaryEnd**(int reqId) **position**(String account, Contract contract, int pos) **positionEnd**()
Orders	EClientSocket	**placeOrder**(int id, Contract contract, Order order) **cancelOrder**(int id) **reqOpenOrders**() **reqAllOpenOrders**() **reqAutoOpenOrders**(boolean bAutoBind) **reqIds**(int numIds) **exerciseOptions**(int tickerId, Contract contract, int exerciseAction, int exerciseQuantity, String account, int override) **reqGlobalCancel**()
Orders	EWrapper	**orderStatus**(int orderId, String status, int filled, int remaining, double avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, String whyHeld) **openOrder**(int orderId, Contract contract, Order order, OrderState orderState) **openOrderEnd**() **nextValidId**(int orderId) **deltaNeutralValidation**(int reqId, UnderComp underComp)

Groups	Methods	Specs
Market Data	EClientSocket	**reqMktData**(int tickerId, Contract contract, String genericTicklist, boolean snapshot, List mktDataOptions) **cancelMktData**(int tickerId) **calculateImpliedVolatility**(int reqId, Contract optionContract, double optionPrice, double underPrice) **cancelcalculateImpliedVolatility**(int reqId) **calculateOptionPrice**(int reqId, Contract contract, double volatility, double underPrice) **cancelCalculateOptionPrice**(int reqId) **reqMarketDataType**(int type)
Market Data	EWrapper	**tickPrice**(int tickerId, int field, double price, int canAutoExecute) **tickSize**(int tickerId, int field, int size) **tickOptionComputation**(int tickerId, int field, double impliedVol, double delta, double optPrice, double pvDividend, double gamma, double vega, double theta, double undPrice) **tickGeneric**(int tickerId, int tickType, double value) **tickString**(int tickerId, int tickType, String value) **tickEFP**(int tickerId, int tickType, double basisPoints, String formattedBasisPoints, double impliedFuture, int holdDays, String futureExpiry, double dividendImpact, double dividendsToExpiry) **tickSnapshotEnd**(int reqId) **marketDataType**(int reqId, int marketDataType)
Connection and Server	EClientSocket	**EClientSocket**(AnyWrapper anyWrapper) **eConnect**(String host, int port, int clientId) **eDisconnect**() **isConnected**() **setServerLogLevel**(int logLevel) **reqCurrentTime**() **serverVersion**() **TwsConnectionTime**()
Connection and Server	EWrapper	**currentTime**(long time) **error**(int id, int errorCode, String errorString) **error**(Exception e) **error**(String str) **connectionClosed**()
Executions	EClientSocket	**reqExecutions**(ExecutionFilter filter)
Executions	EWrapper	**execDetails**(int reqId, Contract contract, Execution execution) **execDetailsEnd**(int reqId) **commissionReport**(CommissionReport commissionReport)
Contract Details	EClientSocket	**reqContractDetails**(int reqId, Contract contract)
Contract Details	EWrapper	**contractDetails**(int ReqId, ContractDetails contractDetails) **contractDetailsEnd**(int reqId) **bondContractDetails**(int reqId, ContractDetails contractDetails)
Market Depth	EClientSocket	**reqMktDepth**(int tickerId, Contract contract, int numRows, Vector mktDepthOptions) **cancelMktDepth**(int TickerId)
Market Depth	EWrapper	**updateMktDepth**(int tickerId, int position, int operation, int side, double price, int size) **updateMktDepthL2**(int tickerId, int position, String marketMaker, int operation, int side, double price, int size)
New Bulletins	EClientSocket	**reqNewsBulletins**(boolean allMsgs) **cancelNewsBulletins**()
New Bulletins	EWrapper	**updateNewsBulletin**(int msgId, int msgType, String message, String origExchange)

Groups	Methods	Specs
Financial Advisors	EClientSocket	**reqManagedAccts**() **requestFA**(int faDataType) **replaceFA**(int faDataType, String xml)
Financial Advisors	EWrapper	**managedAccounts**(String accountsList) **receiveFA**(int faDataType, String xml)
Historical Data	EClientSocket	**reqHistoricalData**(int id, Contract contract, String endDateTime, String durationStr, String barSizeSetting, String whatToShow, int useRTH, int formatDate, List chartOptions) **cancelHistoricalData**(int tickerId)
Historical Data	EWrapper	**historicalData**(int reqId, String date, double open, double high, double low, double close, int volume, int count, double WAP, boolean hasGaps)
Market Scanners	EClientSocket	**reqScannerParameters**() **reqScannerSubscription**(int tickerId, ScannerSubscription subscription, Vector scannerSubscriptionOptions) **cancelScannerSubscription**(int tickerId)
Market Scanners	EWrapper	**scannerParameters**(String xml) **scannerData**(int reqId, int rank, ContractDetails contractDetails, String distance, String benchmark, String projection, String legsStr) **scannerDataEnd**(int reqId)
Real Time Bars	EClientSocket	**reqRealTimeBars**(int tickerId, Contract contract, int barSize, String whatToShow, boolean useRTH, Vector realTimeBarOptions) **cancelRealTimeBars**(int tickerId)
Real Time Bars	EWrapper	**realtimeBar**(int reqId, long time, double open, double high, double low, double close, long volume, double wap,int count)
Fundamental Data	EClientSocket	**reqFundamentalData**(int reqId, Contract contract, String reportType) **cancelFundamentalData**(int reqId)
Fundamental Data	EWrapper	**fundamentalData**(int reqId, String data)
Display Groups	EClientSocket	**queryDisplayGroups**(int reqId) **subscribeToGroupEvents**(int reqId, int groupId) **updateDisplayGroup**(int reqId, const String contractInfo) **unsubscribeFromGroupEvents**(int reqId)
Display Groups	EWrapper	**displayGroupList**(int reqId As Integer, String groups) **displayGroupUpdated**(int reqId, String contractInfo)