

## API Reference Guide

of alternatives for connectivity directly with  
ary Application Program Interface (API)  
ader Workstation and does not require  
dedicated FIX server. This solution is  
l Basic. In addition, we offer an  
s already supporting a FIX  
re. Both solutions can  
ine. See the sub-level tabs  
tion types.

er the Internet. It is  
n and IB. The  
sing the same  
need to  
'Users'  
nd the



**Interactive Brokers**

*The Professional's Gateway to the World's Markets*

## **API Reference Guide**

**June 2013**

Updated through API Release 9.69

© 2013 Interactive Brokers LLC. All rights reserved.

Sun, Sun Microsystems, the Sun Logo and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Excel, Windows and Visual Basic (VB) are trademarks or registered trademarks of the Microsoft Corporation in the United States and/or in other countries.

Any symbols displayed within these pages are for illustrative purposes only, and are not intended to portray any recommendation.

# Contents

Contents .....	i
Overview .....	23
About the APIs .....	24
Run the API through TWS .....	25
Run the API through the IB Gateway .....	26
Recommendations .....	28
API Orders and TWS Precautionary Settings .....	29
API Order IDs .....	31
New Order Example .....	31
Modified Order Example .....	31
Trader Workstation API Settings .....	32
General .....	32
Trusted IP Addresses .....	33
Uninstalling and Re-installing the TWS API Software on Windows .....	34
DDE for Excel .....	35
Getting Started with the DDE for Excel API .....	36
Download the API Components and Spreadsheet .....	37
Configure Trader Workstation to Support API Components .....	38
Open the Sample Spreadsheet .....	39
Using the DDE for Excel Sample Spreadsheet .....	40
Tickers Page .....	41
Using the Tickers Page .....	41
Tickers Page Toolbar Buttons .....	43
Basic Orders Page .....	44
Placing Orders .....	45

Placing a Combination Order .....	46
Supported Order Types .....	47
Basic Orders Page Toolbar Buttons .....	47
Extended Order Attributes Page .....	48
Manually Program Extended Order Attributes .....	49
Apply Extended Order Attributes to Individual Orders and Groups of Orders .....	49
Extended Order Attributes .....	50
Conditional Orders Page .....	54
Setting Up Conditional Orders .....	55
Conditional Order Examples .....	56
If-Filled order .....	56
Price-change order .....	57
Conditional Orders Page Toolbar Buttons .....	58
Advanced Orders Page .....	58
Placing a Bracket Order .....	59
Placing a Volatility Order .....	60
Placing a Trailing Stop Limit Order .....	61
Placing a Scale Order .....	62
Placing a Relative Order .....	63
Advanced Orders Page Toolbar Buttons .....	63
Open Orders Page .....	63
Viewing Open Orders .....	64
Open Orders Tab Toolbar Buttons .....	65
Executions Page .....	65
Viewing Executions .....	66
Executions Page Toolbar Buttons .....	66
Executions Reporting Page .....	66

Running Execution Reports .....	67
Account Page .....	67
Using the Account Page .....	68
Account Page Toolbar Buttons .....	69
Account Page Values .....	69
Portfolio Page .....	73
Viewing Your Portfolio .....	73
Portfolio Page Toolbar Buttons .....	74
Historical Data Page .....	74
Viewing Historical Data .....	75
Historical Data Page Toolbar Buttons .....	76
Historical Data Page Query Specification Fields .....	77
Market Scanner Page .....	79
Starting a Market Scanner Subscription .....	80
Market Scanner Parameters .....	81
Market Scanner Page Toolbar Buttons .....	82
Available Market Scanners .....	82
Contract Details Page .....	86
Requesting Contract Details .....	87
Contract Details Page Toolbar Buttons .....	87
Bond Contract Details Page .....	88
Requesting Bond Contract Details .....	88
Bond Contract Details Page Toolbar Buttons .....	89
Market Depth Page .....	89
Using the Market Depth Page .....	90
Market Depth Page Toolbar Buttons .....	91
Advisors Page .....	91

Allocating Shares to a Single Account .....	92
Placing an Order using an FA Account Group and Method .....	93
Placing an Order using an Allocation Profile .....	93
Advisors Page Toolbar Buttons .....	94
DDE for Excel API Reference .....	95
Viewing the Code .....	95
Modules .....	96
Macros .....	96
Named Ranges .....	97
DDE Syntax for Excel .....	98
Active X .....	105
Linking to the Application using ActiveX .....	106
Registering Third-Party ActiveX Controls .....	107
Running the ActiveX API on 64-bit Windows XP Systems .....	108
Using the Visual Basic Sample Program .....	109
ActiveX Methods .....	110
connect() .....	111
disconnect() .....	111
reqCurrentTime() .....	111
setServerLogLevel() .....	111
reqMktDataEx() .....	112
cancelMktData() .....	112
calculateImpliedVolatility() .....	112
cancelCalculateImpliedVolatility() .....	112
calculateOptionPrice() .....	113
cancelCalculateOptionPrice() .....	113
reqMarketDataType() .....	113

placeOrderEx() .....	113
cancelOrder() .....	114
reqOpenOrders() .....	114
reqAllOpenOrders() .....	114
reqAutoOpenOrders() .....	114
reqIds() .....	115
exerciseOptionsEx() .....	115
reqExecutionsEx() .....	116
reqContractDetailsEx() .....	116
reqMktDepthEx() .....	116
cancelMktDepth() .....	116
reqAccountUpdates() .....	117
reqNewsBulletins() .....	117
cancelNewsBulletins() .....	117
reqManagedAccts() .....	117
requestFA() .....	117
replaceFA() .....	118
reqAccountSummary() .....	118
cancelAccountSummary() .....	120
reqPositions() .....	120
cancelPositions() .....	120
reqHistoricalDataEx() .....	120
cancelHistoricalData() .....	122
reqScannerParameters() .....	122
reqScannerSubscriptionEx() .....	122
cancelScannerSubscription() .....	123
reqRealTimeBarsEx() .....	123

cancelRealTimeBars()	124
createComboLegList()	124
createContract()	124
createExecutionFilter()	124
createOrder()	124
createScannerSubscription()	125
createTagValueList	125
createUnderComp()	125
reqFundamentalData()	125
cancelFundamentalData()	126
ActiveX Events	127
connectionClosed()	128
currentTime()	128
errMsg()	128
tickPrice()	128
tickSize()	129
tickOptionComputation()	129
tickGeneric()	130
tickString()	130
tickEFP()	131
tickSnapshotEnd()	131
marketDataType()	132
orderStatus()	132
openOrderEx()	134
nextValidId()	134
permId()	134
updateAccountValue()	134



updatePortfolioEx() .....	137
updateAccountTime() .....	138
updateNewsBulletin() .....	138
contractDetailsEx() .....	139
contractDetailsEnd() .....	139
bondContractDetails() .....	139
execDetailsEx() .....	141
execDetailsEnd() .....	141
commissionReport() .....	141
updateMktDepth() .....	141
updateMktDepthL2() .....	142
managedAccounts() .....	143
receiveFA() .....	143
accountSummary() .....	143
accountSummaryEnd .....	145
position() .....	145
positionEnd() .....	146
historicalData() .....	146
scannerParameters() .....	146
scannerDataEx() .....	146
scannerDataEnd() .....	147
realtimeBar() .....	147
fundamentalData() .....	148
ActiveX COM Objects .....	149
IExecution .....	149
IExecutionFilter .....	150
ICommissionReport .....	151

IContract .....	151
IContractDetails .....	153
IComboLeg .....	154
IComboLegList .....	155
IOrder .....	155
OrderComboLeg .....	163
IOrderState .....	163
IScannerSubscription .....	164
ITagValueList .....	165
ITagValue .....	165
IUnderComp .....	165
ActiveX Properties .....	167
Placing a Combination Order .....	168
Example .....	168
C++ .....	171
Linking to TWS using the TwsocketClient.dll .....	172
Using the C++ TestSocketClient Sample Program .....	178
To run the pre-built sample application .....	178
To run the TestSocketClient program from Microsoft Visual Studio 2008 .....	178
Class EClientSocket Functions .....	179
EClientSocket() .....	180
eConnect() .....	180
eDisconnect() .....	180
isConnected() .....	180
reqCurrentTime() .....	180
serverVersion() .....	181
setLogLevel() .....	181

TwscConnectionTime()	181
checkMessages()	181
reqMktData()	181
cancelMktData()	182
calculateImpliedVolatility()	182
cancelCalculateImpliedVolatility()	182
calculateOptionPrice()	182
cancelCalculateOptionPrice()	183
reqMarketDataType()	183
placeOrder()	183
cancelOrder()	183
reqOpenOrders()	184
reqAllOpenOrders()	184
reqAutoOpenOrders()	184
reqIDs()	184
exerciseOptions()	185
reqAccountUpdates()	185
reqExecutions()	185
reqContractDetails()	186
reqMktDepth()	186
cancelMktDepth()	186
reqNewsBulletins()	186
cancelNewsBulletins()	187
reqManagedAccts()	187
requestFA()	187
replaceFA()	187
reqAccountSummary()	188

cancelAccountSummary() .....	190
reqPositions() .....	190
cancelPositions() .....	190
reqHistoricalData() .....	190
cancelHistoricalData() .....	192
reqScannerParameters() .....	192
reqScannerSubscription() .....	192
cancelScannerSubscription() .....	192
reqRealTimeBars() .....	193
cancelRealTimeBars() .....	193
reqFundamentalData() .....	194
cancelFundamentalData() .....	194
Class EWrapper Functions .....	195
winError() .....	196
error() .....	196
connectionClosed() .....	196
currentTime() .....	196
tickPrice() .....	196
tickSize() .....	197
tickOptionComputation() .....	197
tickGeneric() .....	198
tickString() .....	198
tickEFP() .....	199
tickSnapshotEnd() .....	199
marketDataType() .....	200
orderStatus() .....	200
openOrder() .....	202

nextValidId() .....	202
updateAccountValue() .....	203
updatePortfolio() .....	203
updateAccountTime() .....	204
updateNewsBulletin() .....	204
contractDetails() .....	205
contractDetailsEnd() .....	205
bondContractDetails() .....	205
execDetails() .....	205
execDetailsEnd() .....	206
commissionReport .....	206
updateMktDepth() .....	206
updateMktDepthL2() .....	207
managedAccounts() .....	207
receiveFA() .....	207
accountSummary() .....	208
accountSummaryEnd .....	210
position() .....	210
positionEnd() .....	210
historicalData() .....	211
scannerParameters() .....	211
scannerData() .....	211
scannerDataEnd() .....	212
realtimeBar() .....	212
fundamentalData() .....	213
SocketClient Properties .....	214
Execution .....	214

ExecutionFilter .....	215
Contract .....	216
ContractDetails .....	217
ComboLeg .....	219
Order .....	220
OrderState .....	227
ScannerSubscription .....	227
UnderComp .....	229
CommissionReport .....	229
Placing a Combination Order .....	230
Example .....	230
Java .....	233
Linking to TWS using the Java API .....	234
Running the Java Test Client Sample Program .....	238
Running the Java Test Client Program with Eclipse .....	242
Java Test Client Overview .....	244
Package .....	244
TestJavaClient Classes .....	244
Java API Overview .....	245
Java EClientSocket Methods .....	247
EClientSocket() .....	248
eConnect() .....	248
eDisconnect() .....	248
isConnected() .....	248
setServerLogLevel() .....	248
reqCurrentTime() .....	249
serverVersion() .....	249

TwscConnectionTime()	249
reqMktData()	249
cancelMktData()	249
calculateImpliedVolatility()	250
cancelCalculateImpliedVolatility()	250
calculateOptionPrice()	250
cancelCalculateOptionPrice()	250
reqMarketDataType()	251
placeOrder()	251
cancelOrder()	251
reqOpenOrders()	251
reqAllOpenOrders	252
reqAutoOpenOrders()	252
reqIDs()	252
exerciseOptions()	252
reqAccountUpdates()	253
reqExecutions()	253
reqContractDetails()	254
reqMktDepth()	254
cancelMktDepth()	254
reqNewsBulletins()	254
cancelNewsBulletins()	255
reqManagedAccts()	255
requestFA()	255
replaceFA()	255
reqAccountSummary()	256
cancelAccountSummary()	258

reqPositions()	258
cancelPositions()	258
reqScannerParameters()	258
reqScannerSubscription()	259
cancelScannerSubscription()	259
reqHistoricalData()	259
cancelHistoricalData()	261
reqRealTimeBars()	261
cancelRealTimeBars()	262
reqFundamentalData()	262
cancelFundamentalData()	263
Java EWrapper Methods	264
currentTime()	264
error()	265
connectionClosed()	265
tickPrice()	265
tickSize()	266
tickOptionComputation()	267
tickGeneric()	267
tickString()	268
tickEFP()	268
tickSnapshotEnd()	269
marketDataType()	269
orderStatus()	269
openOrder()	271
nextValidId()	271
updateAccountValue()	271



updatePortfolio()	272
updateAccountTime()	273
contractDetails()	273
contractDetailsEnd()	273
bondContractDetails()	273
execDetails()	273
execDetailsEnd()	274
commissionReport()	274
updateMktDepth()	274
updateMktDepthL2()	275
updateNewsBulletin()	275
managedAccounts()	276
receiveFA()	276
accountSummary()	276
accountSummaryEnd	279
position()	279
positionEnd()	279
historicalData()	280
scannerParameters()	280
scannerData()	280
scannerDataEnd()	281
realtimeBar()	281
fundamentalData()	282
Java SocketClient Properties	283
Execution	283
ExecutionFilter	284
CommissionReport	285

Contract .....	285
ContractDetails .....	286
ComboLeg .....	288
OrderComboLeg .....	289
Order .....	289
OrderState .....	297
ScannerSubscription .....	297
UnderComp .....	299
Placing a Combination Order .....	300
Example .....	300
Java Code Samples: Contract Parameters .....	303
How to Determine an Option Contract .....	303
How to Determine a Futures Contract .....	304
How to Determine a Stock .....	304
Advisors .....	307
Financial Advisor Orders and Account Configuration .....	308
Excel DDE Support .....	309
Support by Other API Technologies .....	310
Improved Financial Advisor Execution Reporting .....	311
Allocation Methods for Account Groups .....	312
EqualQuantity Method .....	312
NetLiq Method .....	312
AvailableEquity Method .....	312
PctChange Method .....	312
Java Code Samples for Financial Advisor API Orders .....	314
Place an Order for a Single Managed Account .....	314
Place an Order for an Allocation Profile .....	314

Place an Order for an Account Group .....	315
Changing/Updating Allocation Information .....	315
ActiveX for Excel .....	317
Getting Started with the ActiveX for Excel API .....	318
Download the API Components and Spreadsheet .....	318
Running the ActiveX for Excel API on 64-bit Windows XP Systems .....	318
Open the Sample Spreadsheet .....	319
Using the ActiveX for Excel Sample Spreadsheet .....	320
General Page .....	321
General Page Toolbar Buttons .....	323
Tickers Page .....	323
Using the Tickers Page .....	324
Tickers Page Toolbar Buttons .....	325
Bulletins Page .....	326
Bulletins Page Toolbar Buttons .....	327
Market Depth Page .....	327
Using the Market Depth Page .....	328
Market Depth Page Toolbar Buttons .....	328
Basic Orders Page .....	328
Placing Orders .....	329
Placing a Combination Order .....	330
Supported Order Types .....	332
Basic Orders Page Toolbar Buttons .....	332
Conditional Orders Page .....	333
Setting Up Conditional Orders .....	334
Conditional Order Examples .....	335
If-Filled order .....	335

Price-change order .....	335
Conditional Orders Page Toolbar Buttons .....	336
Advanced Orders Page .....	336
Placing a Bracket Order .....	338
Placing a Volatility Order .....	338
Placing a Trailing Stop Limit Order .....	340
Placing a Scale Order .....	341
Placing a Relative Order .....	341
Advanced Orders Page Toolbar Buttons .....	342
Extended Order Attributes Page .....	342
Manually Program Extended Order Attributes .....	343
Apply Extended Order Attributes to Individual Orders and Groups of Orders .....	343
Open Orders Page .....	344
Viewing Open Orders .....	345
Open Orders Tab Toolbar .....	346
Account Page .....	346
Using the Account Page .....	347
Account Page Toolbar Buttons .....	348
Portfolio Page .....	349
Viewing Your Portfolio .....	349
Exercising Options .....	350
Portfolio Page Toolbar Buttons .....	350
Executions Page .....	350
Viewing Executions .....	351
Executions Page Toolbar Buttons .....	352
Historical Data Page .....	352
Viewing Historical Data .....	353

Historical Data Page Query Specification Fields .....	354
Historical Data Page Toolbar Buttons .....	356
Contract Details Page .....	357
Requesting Contract Details .....	357
Contract Details Page Toolbar Buttons .....	358
Bond Contract Details Page .....	359
Requesting Bond Contract Details .....	359
Bond Contract Details Page Toolbar Buttons .....	360
Real Time Bars Page .....	361
Real Time Bars Page Toolbar Buttons .....	362
Market Scanner Page .....	362
Starting a Market Scanner Subscription .....	363
Market Scanner Parameters .....	364
Market Scanner Page Toolbar Buttons .....	364
Fundamentals Page .....	365
Fundamentals Page Toolbar Buttons .....	366
Advisors Page .....	366
Allocating Shares to a Single Account .....	367
Placing an Order using an FA Account Group and Method .....	368
Placing an Order using an Allocation Profile .....	369
Advisors Page Toolbar Buttons .....	369
Log Page .....	370
POSIX .....	371
Running the POSIX Client on a Windows Machine .....	372
Reference .....	373
API Message Codes .....	374
Error Codes .....	374

System Message Codes .....	385
Warning Message Codes .....	385
Historical Data Limitations .....	387
Pacing Violations .....	387
Valid Duration and Bar Size Settings for Historical Data Requests .....	388
Tick Types .....	389
Generic Tick Types .....	392
Using the SHORTABLE Tick .....	393
TAG Values for FUNDAMENTAL_RATIOS tickType .....	394
IBDividends Tick Example .....	399
Example .....	400
RTVolume .....	400
Order Types and IBAlgos .....	402
Supported Order Types .....	402
IBAlgo Parameters .....	404
Arrival Price (ArrivalPx) .....	405
Dark Ice (DarkIce) .....	406
Percentage of Volume (PctVol) .....	406
TWAP (Twap) .....	407
VWAP (Vwap) .....	407
Balance Impact and Risk (BalanceImpactRisk) .....	407
Minimize Impact (MinImpact) .....	408
Accumulate/Distribute (AD) .....	409
CSFB Algo Parameters .....	410
Crossfinder (CROS) .....	411
Crossfinder (CROS) Java Code Sample .....	411
Float (FLT) .....	412

Float (FLT) Java Code Sample .....	412
Guerilla (GRRL) .....	413
Guerilla (GRRL) Java Code Sample .....	413
Work It IW (INIW) .....	414
Work It IW (INIW) Java Code Sample .....	414
Work It (INLN) .....	415
Work It (INLN) Java Code Sample .....	415
Pathfinder (PTHF) .....	416
Pathfinder (PTHF) Java Code Sample .....	416
Reserve (RSRV) .....	417
Reserve (RSRV) Java Code Sample .....	417
Strike (SNPR) .....	418
Strike (SNPR) Java Code Sample .....	418
10B 18 (TENB) Java Code Sample .....	418
10B 18 (TENB) Java Code Sample .....	419
Tex (TEX) .....	419
Tex (TEX) Java Code Sample .....	420
TWAP (TWAP) .....	420
TWAP (TWAP) Java Code Sample .....	421
VWAP (VWAP) .....	421
VWAP (VWAP) Java Code Sample .....	421
Extended Order Attributes .....	423
Order Status for Partial Fills .....	427
Available Market Scanners .....	428
Instruments and Location Codes for Market Scanners .....	432
Supported Time Zones .....	433
Smart Combo Routing .....	434

API Logging .....	435
Example Log Entry .....	435
API Request/Server Response Message Identifiers .....	436
Requests for Quotes (RFQs) .....	437
Submitting RFQs using the API .....	437
Delta-Neutral RFQs .....	437
RFQ Samples .....	437
Support for Mini Options .....	438
Support for Mini Options - ActiveX, Java and C++ APIs .....	438
Support for Mini Options - DDE for Excel .....	439
Requirements .....	439
DDE Syntax Examples .....	439
Requests That Receive Contract Data from TWS .....	440
Requesting Real-Time Index Premium Data .....	441
Troubleshooting FAQs .....	442
Index .....	445



# Overview

This chapter provides an overview of the APIs (Application Programming Interfaces) available, including the following topics:

- [About the APIs](#)
- [Run the API through TWS](#)
- [Run the API through the IB Gateway](#)
- [Recommendations](#)
- [API Orders and TWS Precautionary Settings](#)
- [API Order IDs](#)
- [Trader Workstation API Settings](#)
- [Uninstalling and Re-installing the TWS API Software on Windows](#)

## About the APIs

We provide several APIs which you can use to link to our system and trade your IB account. The API allows you to connect through either the TWS or the IB Gateway. Connecting through the TWS requires that you have the application running, but also allows you to test and confirm that your API orders are working correctly. Connecting through the IB Gateway allows you to use the API without a large GUI application running, but does not provide an interface for you to test and confirm API activity.

Regardless of the connection method you use, the API allows you to:

To view syntax for specific functionality, see the [DDE for Excel](#), [ActiveX](#), [C++](#) or [Java](#) topics in this guide. Customers with no programming expertise should begin with the DDE for Excel section, which uses an everyday Excel® spreadsheet to link to TWS via the API.

**Note:** API topics are written for experienced programmers and provide little guidance for non-technical users.

To develop and test your API program, we recommend that you use the sample application and connect via TWS. Once you are satisfied that the API works as designed, you can use the GUI-less IB Gateway to connect, if you desire.

### To use the API components and view sample source code and spreadsheets

1. Install or upgrade the latest API and sample files from the IB website:
  - From the IB website menu, click **Trading Technology > API Solutions**.
  - Click **IB API**, then click the **API Software** button.
  - In popup window, click the button corresponding to the production or beta API version you want to install. The installation program is downloaded to your computer.
  - Run the downloaded installation program to install the API software on your computer.
2. Log in to Trader Workstation (TWS) and configure it to support the API. You do this in Global Configuration > API > Settings in TWS.
3. Use the sample application to learn how to request market data, submit orders, etc.
4. Customize the sample applications to meet your needs, or create your own application using described syntax and functionality.

## Run the API through TWS

To run the API through TWS, you must always have your system running and it must be configured to use any of the API components.

### To enable API connection through TWS

1. Log into TWS.
2. On the **Edit** menu, select *Global Configuration*.
3. Select *API* in the left pane, then click *Settings*.
4. In the right pane, click the check box for *Enable ActiveX and Socket Clients* (ActiveX, C++ and Java API connections), and/or *Enable DDE Clients* (for DDE for Excel API connections only) to configure TWS for the appropriate API connection. You must have these settings enabled to connect to the API through TWS.

**Note:** Multiple API clients with different client IDs can access a single instance of TWS on the same computer. With the exception of DDE for Excel, the API application does not need to be running on the same computer as TWS.

For a complete description of all Trader Workstation's API settings, see the [TWS Users' Guide](#).

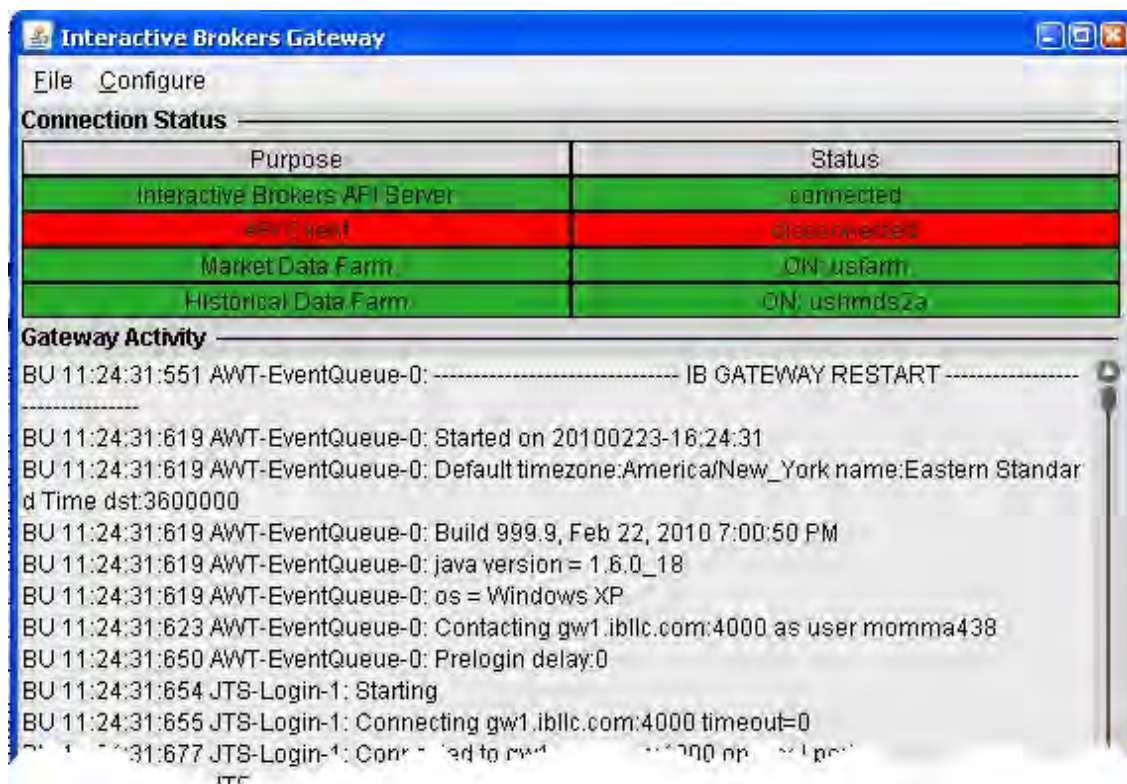
## Run the API through the IB Gateway

The IB Gateway provides a low-resource alternative to TWS for connecting to the IB trading system via the API. The gateway uses approximately 40% fewer system resources than TWS. However, the gateway is GUI-less, which means that you cannot view the API activity as you can when running TWS.



### To log into the IB Gateway

1. From the **Login** menu on the IB web site, select *IB Gateway*.
2. Select the API radio button.
3. Log in using your IB username and password, just as you would when logging into TWS.
4. Click **Login**. The Interactive Brokers Gateway box opens, displaying the connection status and gateway activity.



You must have the IB Gateway running while connected to the API.

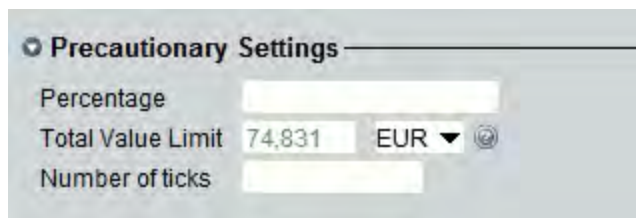
## Recommendations

Before you use our TWS API to create your own customized trading application, you should consider the following important recommendations:

- Placing Orders by Conid - When you place an order by conid, you must provide the conid AND the exchange. If you provide extra fields when placing an order by conid, the order may not work.
- Order IDs - Each order you place must have a unique Order ID. We recommend that you increment your own Order IDs to avoid conflicts between orders placed from your API application. To resolve issues with Order IDs, click the **Reset API order ID sequence** button on the API - Settings panel in TWS Global Configuration.
- Please test your API application with an IB Paper Trading account to catch and avoid any errors. You can request a Paper Trading account from Account Management.
- While the API supports up to eight simultaneous API connections using the same login to a single running TWS, we recommend that you avoid this scenario. If possible, use a single API connection for your application to avoid performance overhead.

## API Orders and TWS Precautionary Settings

By default, Trader Workstation includes precautionary settings as part of its Order Presets on the TWS Configuration page. Precautionary settings are safety checks and include percentage, size limit, total value and number of ticks. They can be modified in TWS for most instrument types (stocks, options, and so on) or for specific tickers.

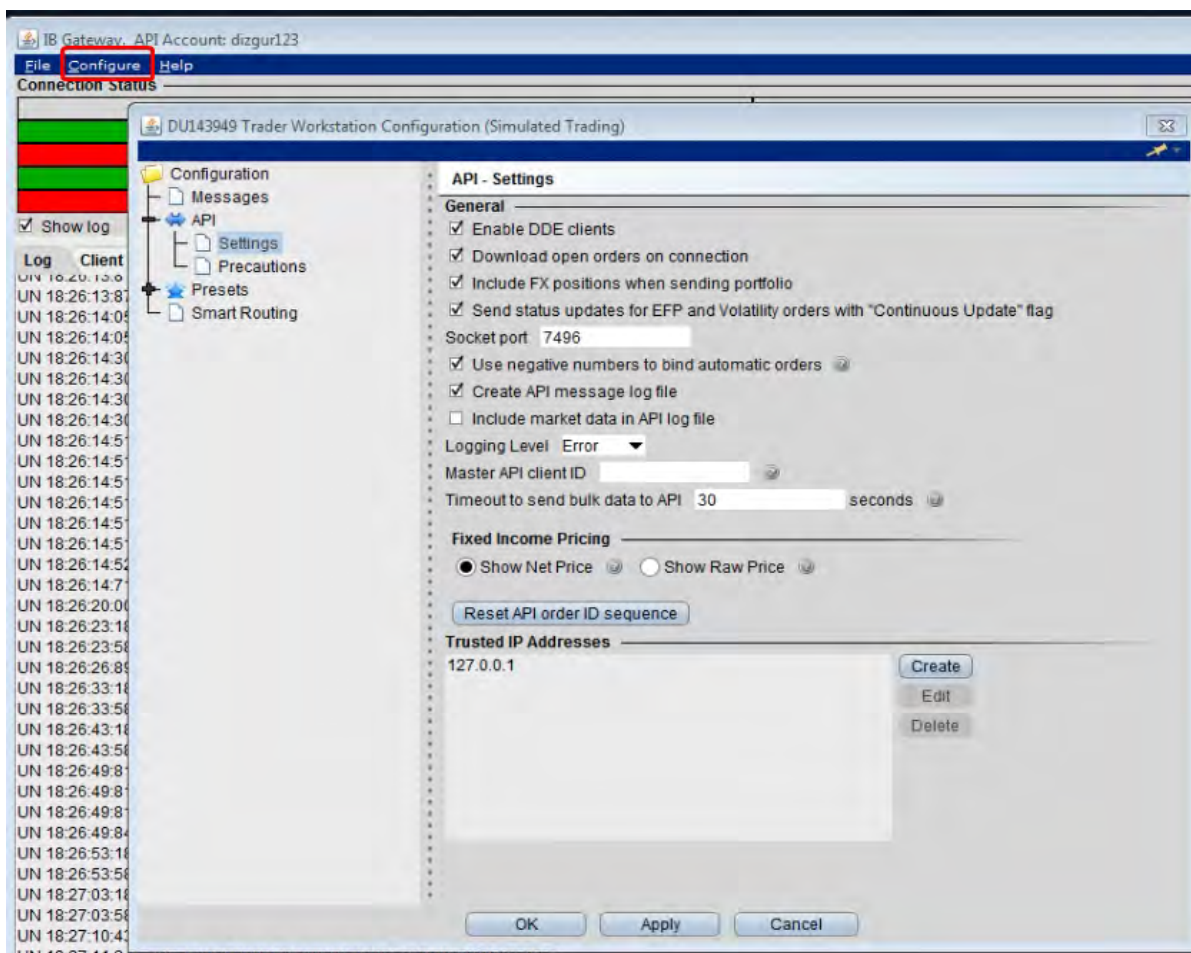


If your API order violates these settings, you will receive an error message. For example, the default precautionary setting for order size is 500. If you place an order for 1000 shares of stock, you will receive an error message indicating that the size specified violates the constraints specified in the default order settings. TWS precautionary settings apply to API orders placed from ALL API technologies.

You can override the precautionary settings by doing one of the following:

In TWS:

- On the Configure menu, select *API* then *All API Settings*. Select the *Bypass Order Precautions for API Orders* check box, then click **OK**. All of your API orders will ignore the precautionary settings in TWS.
- In the Order Presets, enter higher precautionary setting limits for the desired instrument types and or tickers. On the Configure menu, select *Order* then select *Order Presets*. Select the instrument type or ticker on the left, enter the desired limits in the Precautionary Settings section of the page, then click **OK**.



In the IB Gateway:

- From the Configure menu, select *Settings*. Select the *Bypass Order Precautions for API Orders* checkbox and click **OK**. All of your API orders will ignore the precautionary settings you had set via a TWS session.



## API Order IDs

When you place a new order using the API, the order id number must be greater than the previously used numbers. For example, if you place an order with an Order ID of 11, the next order you place should have an Order ID of at least 12. So when you place a new order, the order id must be greater than the previously used order id number.

### New Order Example

In this example, a user is going to place two orders for IBM stock. The first order is a BUY order for 200 shares and set the limit price to \$85.25. The second order will be an order to sell 100 shares with the limit price set to \$84.25.

In this example, the first order is tagged with an Order ID of 1:

```
.placeOrder(1, IBM, BUY, $85.25, 200...)
```

Now, user can place a second order. This order is assigned an Order ID of 2:

```
.placeOrder(2, IBM, SELL, $84.25, 100...)
```

### Modified Order Example

To modify an order using the API, resubmit the order you want to modify using the same order id, but with the price or quantity modified as required. Only certain fields such as price or quantity can be altered using this method. If you want to change the order type or action, you will have to cancel the order and submit a new order.

In this example, a user initially decides to buy 100 shares and sets the limit price to \$85.25. Then, customer wants to modify the same order and change the limit price to \$86.25. Note that the first order is assigned an Order ID of 3:

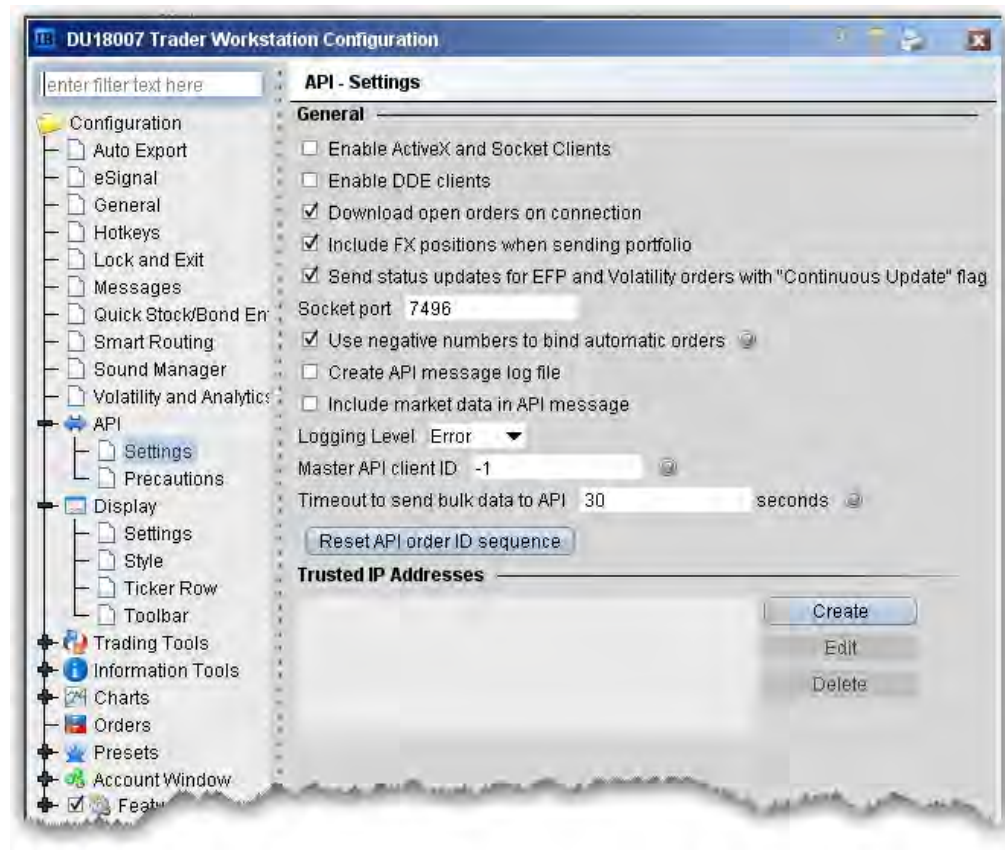
```
.placeOrder(3, IBM, BUY, $85.25, 100...)
```

You can now modify the limit price for this order by calling the same `.placeOrder` method and using the same Order ID of 3, with the limit price modified to \$86.25

```
.placeOrder(3, IBM, BUY, $86.25, 100...)
```

## Trader Workstation API Settings

In addition to configuring Trader Workstation (TWS) to communicate with the API, there are a number of other API-related settings in TWS that you can configure.



### To configure API settings in TWS

1. In TWS, select the **Edit** menu, then select *Global Configuration*.
2. Click *API* in the left pane, and select *Settings*.
3. Configure the API settings as required. These are described below.

**Note:** With the exception of DDE, the API application does not need to be running on the same computer on which the application is running.

### General

- **Enable Active X and Socket Clients** - Check to enable integration using ActiveX or socket clients including Java and C++.
- **Enable DDE clients** - Check to enable integration with TWS with TWS through DDE.
- **Download open orders on connection** - uncheck if you do not want to download all open orders when you connect to your API.

- **Include FX positions when sending portfolio** - If you have the Include FX Positions feature activated, all FX positions will be included when portfolio updates are sent to the API client. Uncheck this box if you don't want FX positions sent to the API client when the portfolio updates are sent.
- **Send status updates for EFP and Volatility orders with "Continuous Update" flag** - If you have Continuous Update activated for EFP or Volatility orders, all updates are sent to the API client by default. Uncheck if you don't want these updates sent from TWS to the API client.
- **Use negative numbers to bind automatic orders** - if checked, all orders that are automatically bound to an API client via the reqOpenOrders or reqAutoOpenOrders calls or via system-generated orders (i.e. volatility hedging orders) will be assigned negative API order IDs. Otherwise, these orders will be assigned incremental API order IDs. Volatility hedging orders will have the order ID "parent API order ID + 1" when possible.
- **Create API message log file** - check to create a message log file. Use the Logging Level selector to define the level of detail in the log.
- **Include market data in API message** - shows market data in the API log file.
- **Socket port** - Enter a socket port number which allows you to access multiple instances of TWS or IB Gateway running on a single host. By assigning a unique socket port number to each TWS or IB Gateway instance, a single ActiveX or socket API client will be able to access each of these instances. This does not apply to DDE clients.
- **Logging Level** - Set the level of log detail for the API text log. System gives the most general level of logging; Detail gives the most detailed level. Note that Detail uses more computing resources and may result in a decrease in performance.
- **Master API client ID** - The API client with the specified client ID will receive all orders, even those placed by other API clients. This differs from the Client ID of "0" which will receive all orders sent from the TWS GUI.
- **Timeout to send bulk data to API** - define the time in seconds that TWS will wait before disconnecting the API client if data cannot be sent quickly enough.

## Trusted IP Addresses

If you connect to the API through a trusted IP address, the connection is not questioned. Otherwise, you will get a verification message asking if you are sure you want to make the connection.

- Click **Create** to add a new trusted IP address to the list.
- Click **Edit** to modify the selected address.
- Click **Delete** to remove the selected address.

## Uninstalling and Re-installing the TWS API Software on Windows

If you encounter problems running the TWS API software on the Windows platform, you can uninstall and re-install the API software.

**Note:** This procedure is usually only necessary when troubleshooting the most extreme API problems.

### To uninstall and re-install the TWS API software on Windows

1. Open the Windows Control Panel, then open *Add or Remove Programs*.
2. Select *TWS Interoperability Components* from the list of installed programs, then click **Change/Remove**.
3. Select *Automatic*, then click **Next** to uninstall the TWS API software.
4. In the Windows Explorer, delete the file *TwsSocketClient.dll* from the *Windows\system32* folder.
5. Reboot your computer.
6. Re-install the TWS API software.

# DDE for Excel

This chapter describes the DDE for Excel API, including the following topics:

- [Getting Started with the DDE for Excel API](#)
- [Using the DDE for Excel Sample Spreadsheet](#)
- [DDE for Excel API Reference](#)

DDE is an acronym for Dynamic Data Exchange, a Microsoft-created communication method that allows multiple applications that are running simultaneously to exchange data and commands. We use this protocol to link Excel with your running version of TWS or the IB Gateway, allowing you to view real-time market data (including market depth) manage orders and monitor your executions and account information using an Excel spreadsheet.

The following figure shows the Tickers page in the Excel DDE API sample spreadsheet.

Symbol	Type	Expiry	Strike	Vol	Manager	Exchange	Primary Exchange	Currency	Contract	Bid	Bid Price	Ask Price	Ask Size
<b>Stocks</b>													
MSFT	STK					NYSE	NYSE	USD		96	26.62	26.68	280
YHOO	STK					NYSE	NYSE	USD		184	20.12	20.13	135
GE	STK					NYSE	NYSE	USD		197	20.17	20.18	16
GOOG	STK					NYSE	NYSE	USD		1	492.47	492.76	6
QQQQ	STK					NYSE	NYSE	USD		204	45.21	45.22	565
IBM	STK					NYSE	NYSE	USD		1	127.29	127.37	1
MOT	STK					NYSE	NYSE	USD		1	7.41	7.42	184
<b>Options</b>													
GOOG	OPT	200812	27.5	C	133	NYSE	NYSE	USD		0.3755722	0.390132	215.8	217.3
GOOG	OPT	200812	27.5	P	133	NYSE	NYSE	USD		0.4039953	0.374134	50	1.24
IBM	OPT	200810	135	P	133	NYSE	NYSE	USD		0.2755823	0.266139	181	10.3
IBM	OPT	200810	29	C	133	NYSE	NYSE	USD		0	447	98.9	98.2
MSFT	OPT	200810	29	C	133	NYSE	NYSE	USD		0.2533675	0.2891957	8339	6.65
MSFT	OPT	200810	29	P	133	NYSE	NYSE	USD		0.3339432	0.3539712	9447	0.13
<b>Index Futures</b>													
Z	FUT	200812				CBOT	CBOT	USD					
SPX	FUT	200812				CBOT	CBOT	USD					
SPX	FUT	200812				CBOT	CBOT	USD					
<b>Interest Rate Futures</b>													
ZB	FUT	200812				CBOT	CBOT	USD					
ZB	FUT	200812				CBOT	CBOT	USD					
<b>Energy Futures</b>													

## Getting Started with the DDE for Excel API

We have created a sample DDE-linked Excel spreadsheet, TwsDde.xls, that you can use with your TWS to create a custom Excel application. It's easy to get started with the DDE for Excel API:

- [Download the API components](#) and sample Excel spreadsheet.
- Ensure that the DDE clients are enabled, either in Trader Workstation or IB Gateway, as described here [here](#), or that the IB Gateway is running.
- [Open the spreadsheet](#) and start using the DDE for Excel API.

The sample spreadsheet currently comprises several pages complete with sample data and action buttons that make it easy for you to get market data, send orders and view your activity.

## Download the API Components and Spreadsheet

We recommend using the sample Excel spreadsheet that we provide as a starting point toward creating your own DDE for Excel API. Follow the steps below to download the sample spreadsheet.

### To install the sample DDE Spreadsheet

1. From the [IB homepage](#), select *API Solutions* from the **Software** menu.
2. Click the *IB API* icon, and on the API Software page, find the column appropriate to your operating system, click *Download latest version*.

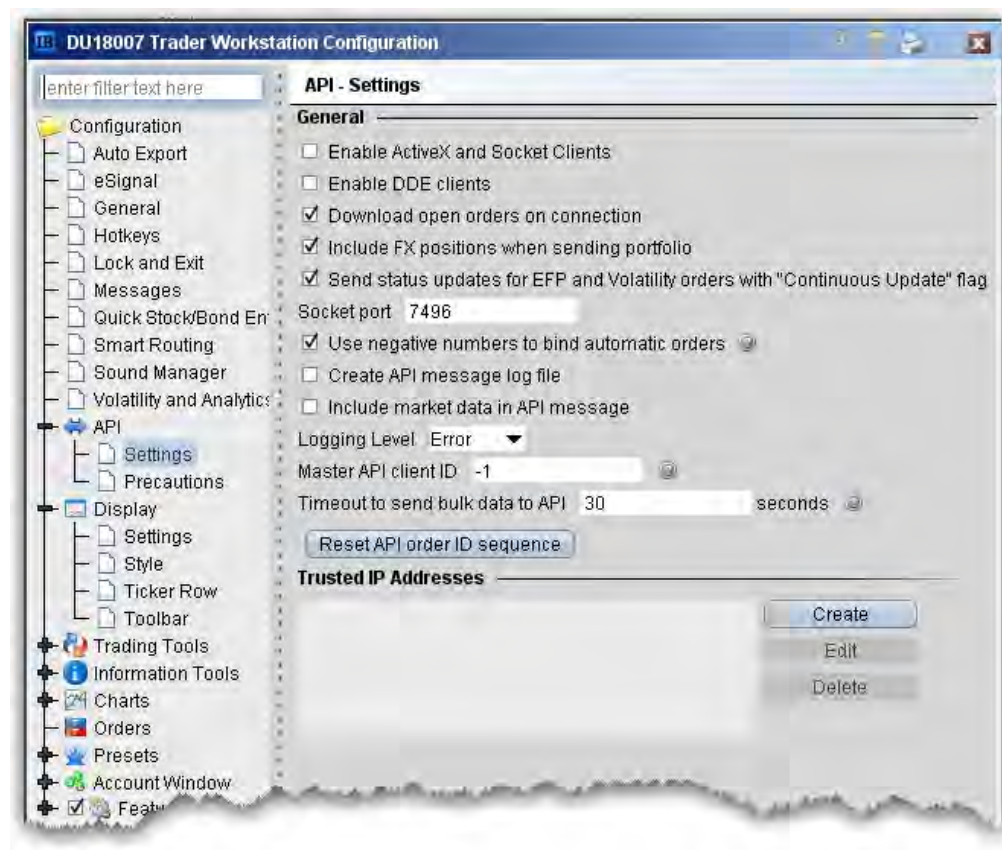
**Note:** Windows users can download the beta test version of the API by using the Windows Beta column, or revert to the previous production version by selecting *Downgrade to Previous Version*.

3. Save the installation program to your computer, and if desired, select a different directory. Click **Save**. Note that the API installation file is named for the API version; for example, *InstallAX\_960*.
4. Close any versions of TWS, the IB Gateway and Excel that you have running.
5. Locate the API installation program you just saved to your computer, then double-click the file to begin the API installation.
6. Follow the instructions in the installation wizard. By default, the sample DDE spreadsheet is saved to C:\Jts\Excel\TwsDde.xls.

**Note:** Before you can [use the spreadsheet](#), you must have TWS running and configured to support the DDE API. You can also run the sample against the [IB Gateway](#) but we recommend you start by running TWS.

## Configure Trader Workstation to Support API Components

You must have your system running to use any of the API components.



### To configure the application to support accessing its functionality via the API

1. On the **Edit** menu select *Global Configuration*.
2. Click *API* in the left pane, and select *Settings*.
3. On the right panel, check *Enable DDE clients* to enable integration with TWS with TWS through DDE. Download sample programs from the Software page on the IB website.
4. Set the rest of the API parameters as required. For details, see [Trader Workstation API Settings](#).

**Note:** Not more than one API application can simultaneously access a single instance. With the exception of DDE, the API application does not need to be running on the same computer on which the application is running.



## Open the Sample Spreadsheet

After you have downloaded the sample spreadsheet and configured the application to allow the DDE for Excel API to link to it, open the spreadsheet and save it as your personal file.

### To open the sample spreadsheet

1. Go to the API installation folder in which the Excel API sample spreadsheet was installed (typically C:\Jts\Excel\TwsDde.xls), and double-click **TwsDde.xls**.
2. In the macro warning message box, click **Enable Macros**. If you receive a message asking if you want to link to information in another worksheet, click **Yes**.

**Note:** To use the spreadsheet macros, your Excel macro security must be set to Medium or Low. If you cannot open the spreadsheet or if the macros don't work, you need to modify your macro security level.

In Microsoft Excel 2007, click the Microsoft Office Button, click **Excel Options**, and then click **Trust Center** in the Excel Options window. In the Trust Center, click *Macro Settings*, then change your settings as required.

In previous versions of Excel, select *Macro* from the **Tools** menu, and then select *Security*. Set security to Medium or Low.

3. In the *User Name* field in the *Which Trader Workstation?* area, type your account user name. Note that you must type your User Name on each page of the worksheet to properly connect.



We recommend using this spreadsheet as the starting point for your API application. This means that when new features are added, you will need to cut and paste your information from your Excel spreadsheet to the newly released sample spreadsheet, then save the application under a different filename.

## Using the DDE for Excel Sample Spreadsheet

The DDE for Excel API sample spreadsheet, TwsDde.xls, includes the following pages (tabs):

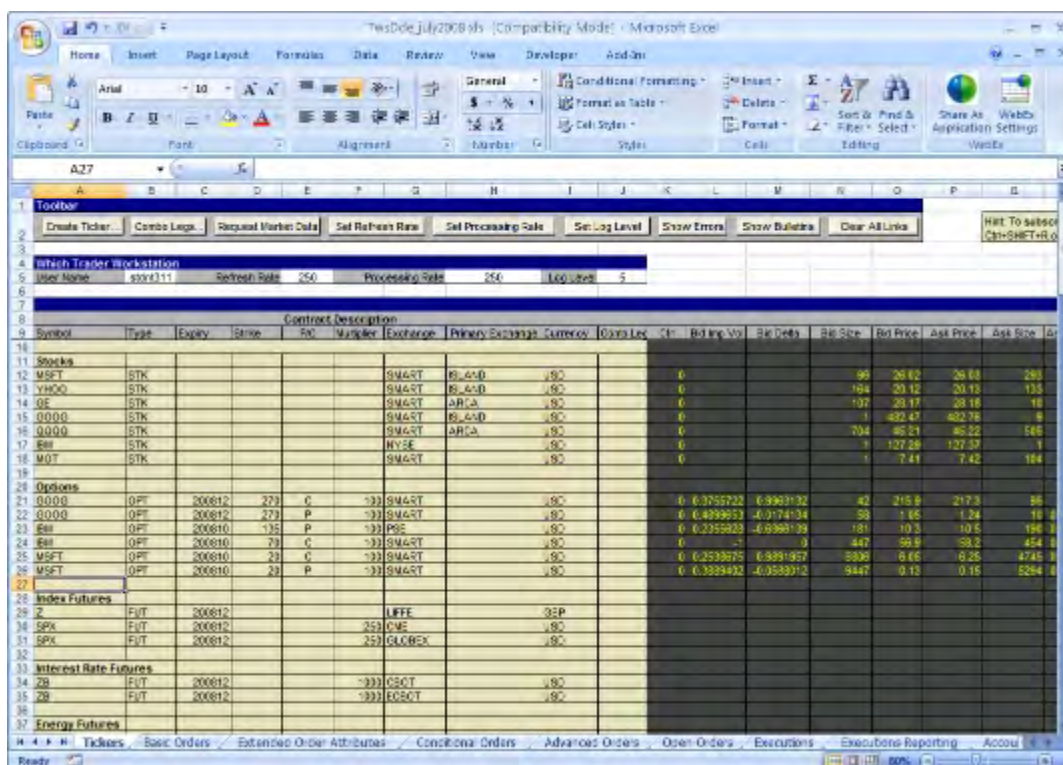
Page	Description
<a href="#"><u>Tickers</u></a>	Lets you set up your ticker lines and request market data. You can view market data for all asset types including EFPs and combination orders.
<a href="#"><u>Basic Orders</u></a>	Lets you send and modify orders, and set up combination orders and EFPs.
<a href="#"><u>Extended Order Attributes</u></a>	Used in conjunction with the Basic Orders, Advanced Orders, Conditional Orders and Advisors pages, this page lets you change the time in force, create Hidden or Iceberg orders and apply many other order attributes.
<a href="#"><u>Conditional Orders</u></a>	Lets you create an order whose submission is contingent on other conditions being met, for example an order based on a prior fill.
<a href="#"><u>Open Orders</u></a>	Shows you transmitted orders that are still working, including those that have been accepted by the IB system, and those that are working at an exchange.
<a href="#"><u>Advanced Orders</u></a>	Lets you send and modify advanced orders types that require the use of extended order attributes, such as Bracket, Scale and Trailing Stop Limit orders.
<a href="#"><u>Executions</u></a>	Lets you view all execution reports, and includes a filtering box so you can limit your results.
<a href="#"><u>Executions Reporting</u></a>	Linked to the Executions page, this page lets you run four different types of execution reports.
<a href="#"><u>Account</u></a>	Provides up to date account information.
<a href="#"><u>Portfolio</u></a>	Displays all your current positions.
<a href="#"><u>Historical Data</u></a>	Request historical data for an instrument based on data you enter in a query.
<a href="#"><u>Market Scanner</u></a>	Subscribe to TWS market scanners.
<a href="#"><u>Contract Details</u></a>	Lets you collect contract-specific information you will need for other actions, including the conid and supported order types for a contract
<a href="#"><u>Bond Contract Details</u></a>	Lets you collect bond contract-specific information you will need for other actions, including bond coupon and maturity date.
<a href="#"><u>Market Depth</u></a>	Lets you view market depth for selected quotes.
<a href="#"><u>Advisors</u></a>	Lets Financial Advisors send and modify FA orders.

**Note:** Two additional pages, Old Style Executions and Old Style Account-Portfolio, represent functionality that has been replaced by other pages in the spreadsheet (Executions, Account and Portfolio pages). While these older pages are still included in the TswDde.xls sample spreadsheet, they are no longer documented in this API Users' Guide and you should not use them.

## Tickers Page

Use the Tickers page to:

- Create market data (ticker) lines.
- Request market data.



## Using the Tickers Page

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To create a ticker using the Create Ticker button

1. Click the **Tickers** tab at the bottom of the spreadsheet.
2. Click the line number to the left of a blank row to select the row. You must have a blank row selected to create a ticker line.
3. Click the **Create Ticker** button on the toolbar and enter information in the Tickers box.
4. Click **OK**.

For stocks, you only need to specify the *Symbol*, *Type*, *Exchange* (usually SMART), and *Currency*.

The image shows a 'Ticker' dialog box with the following fields:

- Symbol: Text input field
- Type: Dropdown menu
- Expiry: Text input field
- Strike: Text input field
- Right: Dropdown menu
- Multiplier: Text input field
- Exchange: Text input field
- Primary Exchange: Text input field
- Currency: Text input field
- Buttons: OK and Cancel at the bottom

### To create a ticker on the spreadsheet

1. Select a blank cell in the *Symbol* column and enter a symbol.
2. Tab through the all contract description fields and enter data where necessary, for example if you are entering a stock ticker, you don't need values in the Expiry, Strike, P/C and Multiplier fields.

The *Exchange* field accepts the following values: SMART (for smart order routing), and any valid exchange acronym.

### To request market data for a ticker

1. Select the ticker row for which you want to request market data by clicking the row number.
2. Press **Ctrl+R**, or click **Request Market Data** on the toolbar.

To get market data for a group of tickers, select multiple ticker rows while holding down the **Shift** key, then click **Request Market Data** multiple times until all rows are showing data.

### To set the refresh rate

The refresh rate determines how often the DDE link to TWS is refreshed.

TWS market data updates every 300 milliseconds by default, so setting the refresh rate to 250 will get every tick to the spreadsheet.

### To set the processing rate

The server processing rate affects the speed at which the DDE handles requests between TWS and the spreadsheet.

The allowed range is 100 ms- 2000 ms, inclusive.

### To set the level of detail for logging of API client requests

1. In the *Log Level* field in the *Which Trader Workstation?* area, enter the desired log level value (1 =SYSTEM, 2=ERROR, 3=WARNING, 4=INFORMATION, 5=DETAIL).
2. Move your cursor out of the *Log Level* field, then click the **Set Log Level** button.

### To remove all DDE links to TWS

The **Clear All Links** button on the Tickers page lets you remove all DDE links from the TwsDde.xls spreadsheet to TWS that the Visual Basic for Applications (VBA) code provided with the spreadsheet could create. You typically use this button when you are preparing to save the spreadsheet.

Clicking this button cancels all market data, historical data, market scanner subscriptions, and other data requests. If you add your own links to existing or new pages, update the *clearAllLinks* macro to clear those links as well. Each page in the spreadsheet contains its own *clearLinks* macro; these are all called by the *clearAllLinks* macro.

**Note:** Clearing all links does NOT cancel orders.

## Tickers Page Toolbar Buttons

The toolbar on the Tickers page includes the buttons described below.

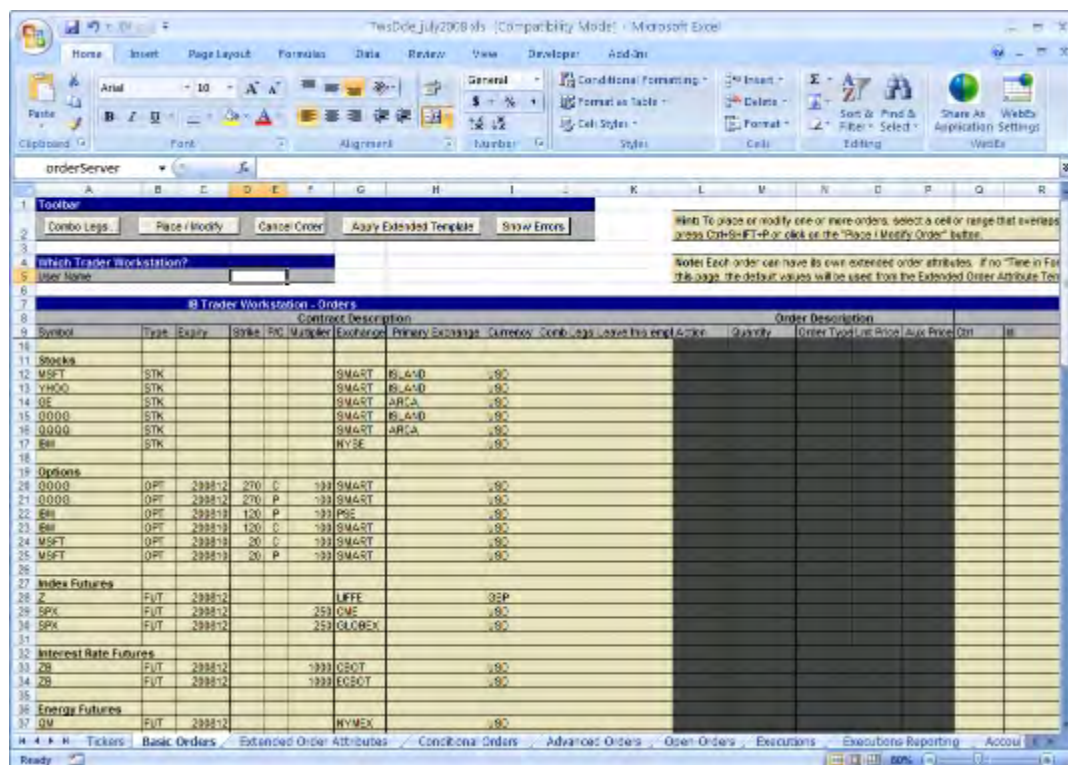
Button	Description
<b>Create Ticker</b>	Opens the Ticker box. Enter information to create a market data line.
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Request Market Data</b>	Select a line and click to get market data for the selected contract.
<b>Set Refresh Rate</b>	<p>The Refresh Rate value is in milliseconds, and determines how often the DDE link to TWS is refreshed. The default refresh rate is 1000 (updates every 1 second), and the allowed range is 100ms to 2000ms, inclusive.</p> <p>Note that the TWS market data updates every 300 milliseconds. This means the default "every 1 second" rate will only show 30% of the ticks. A Refresh Rate of 250 will get every tick to the spreadsheet.</p>
<b>Set Processing Rate</b>	Set the TWS/DDE server message processing rate (also in milliseconds) to affect the speed at which DDE will handle requests between the spreadsheet and TWS. The allowed range is 100ms to 2000ms, inclusive.

<b>Set Log Level</b>	<p>This specifies the level of log entry detail used when processing API requests. Valid values include:</p> <p>1 = SYSTEM</p> <p>2 = ERROR</p> <p>3 = WARNING</p> <p>4 = INFORMATION</p> <p>5 = DETAIL</p>
<b>Show Errors</b>	Jumps to the Error Code field and shows the most recent error code.
<b>Show Bulletins</b>	Opens the News Bulletins message. If you subscribe to bulletins, news will appear in the RED box in the upper right corner of the spreadsheet.
<b>Clear All Links</b>	Clears all DDE links to the TWS.

## Basic Orders Page

Use the Basic Orders page to:

- Create an order.
- Create a "basket" of orders.
- Modify and cancel orders.
- Create combination orders.



## Placing Orders

This topic describes how to place the following types of orders on the Orders page:

- Simple orders
- Basket orders
- Modified orders

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To place an order

1. Click the **Basic Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.  
You must define the *Action* (Buy, Sell or Short Sell), *Quantity*, *Order Type*, *Limit Price* (unless it's a market order) and if necessary, the *Aux. Price* for order types that require it.
4. If desired, apply extended order attributes by clicking the **Apply Extended Template** button on the toolbar. This applies all attributes you have defined on the [Extended Order Attributes](#) page.
5. Click the **Place/Modify Order** button in the *Toolbar* section of the page.

### To place a "basket" of orders

1. Click the **Basic Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using fields in the *Order Description* section.
4. Repeat Steps 1 and 2 for additional orders.
5. Select a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.
6. Click the **Place/Modify Order** button.

### To modify an order (or group of orders)

1. On the Basic Orders page, change any necessary parameters in an order or group of orders.
2. Select the order or a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.



- Click the **Place/Modify Order** button.

**Note:** You cannot modify orders in the DDE for Excel API that were submitted from TWS. This is because the DDE for Excel API uses its own created order ID to modify the orders, not the global customer order ID. All orders created in TWS have an internal order ID of 0. If you try to modify an order in the DDE for Excel API with id = "id0" you will get the error "duplicate order ID".

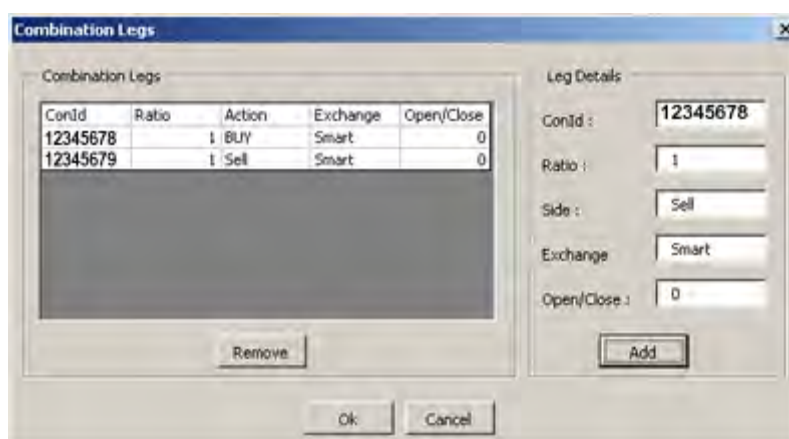
## Placing a Combination Order

A combination order is a special type of order that is constructed of many separate legs but executed as a single transaction.

For example, to buy a calendar spread, you would:

- Buy 1 OPT JUL03 17.5 CALL (100)
- Sell 1 OPT AUG03 17.5 CALL (100)

The following example walks you through the process of placing a hypothetical calendar spread order for XYZ on ISE.



## To create a calendar spread order

- Use the [Contract Details](#) page to get the contract id for both of the leg definitions.
  - The conid for XYZ option JUL08 17.5 CALL on ISE is "12345678".
  - The conid for XYZ option AUG08 17.5 CALL on ISE is "12345679".
- Click the **Basic Orders** tab to build the combo leg definitions. Click the **Combo Legs** button on the Basic Orders page toolbar and enter leg information. Your leg information is translated into the format:

[CMBLGS]\_[NumOfLegs]\_[Combo Leg Definitions]\_[CMBLGS]

where:

- [CMBLGS] is the delimiter used to identify the start and end of the leg definitions
- [NumOfLegs] is the number of leg definitions



- [Combo Leg Definitions] defines N leg definitions, and each leg definition consists of [conid]\_[ratio]\_[action]\_[exchange]\_[openClose], so the resulting combo substring looks as follows:

CMBLGS\_2\_17496957\_1\_BUY\_EMPTY\_0\_15910089\_1\_SELL\_EMPTY\_0\_CMBLGS

3. The combination leg definitions must occur before the extended order attributes. The full place order DDE request string will look like this:

```
=acctName|ord!id12345?place?BUY_1_XYZ_BAG_ISE_LMT_1_CMBLGS_2_12345678_1_BUY_EMPTY_0_12345679_1_SELL_EMPTY_0_CMBLGS_DAY_EMPTY_0_O_0_EMPTY_0_EMPTY_0_0_EMPTY_0_0
```

If the order legs do not constitute a valid combination, one of the following errors will be returned:

- 312 = The combo details are invalid.
- 313 = The combo details for '<leg number>' are invalid.
- 314 = Security type 'BAG' requires combo leg details.
- 315 = Stock combo legs are restricted to SMART exchange.

**Note:** 1. The exchange for the leg definition must match that of the combination order. The exception is for a STK leg definition, which must specify the SMART exchange.

2. The openClose leg definition value is always 'SAME' (i.e.0) for retail accounts. For institutional accounts, the value may be any of the following: (SAME, OPEN, CLOSE).

## Supported Order Types

The order types currently supported through the DDE for Excel API are:

- Limit (LMT)
- Market (MKT)
- Limit if Touched (LIT)
- Market if Touched (MIT)
- Market on Close (MOC)
- Limit on Close (LOC)
- Pegged to Market (PEGMKT)
- Relative (REL)
- Stop (STP)
- Stop Limit (STPLMT)
- Trailing Stop (TRAIL)
- Trailing Stop Limit (TRAILLIMIT)
- Volume-Weighted Average Price (VWAP)
- Volatility orders (VOL)

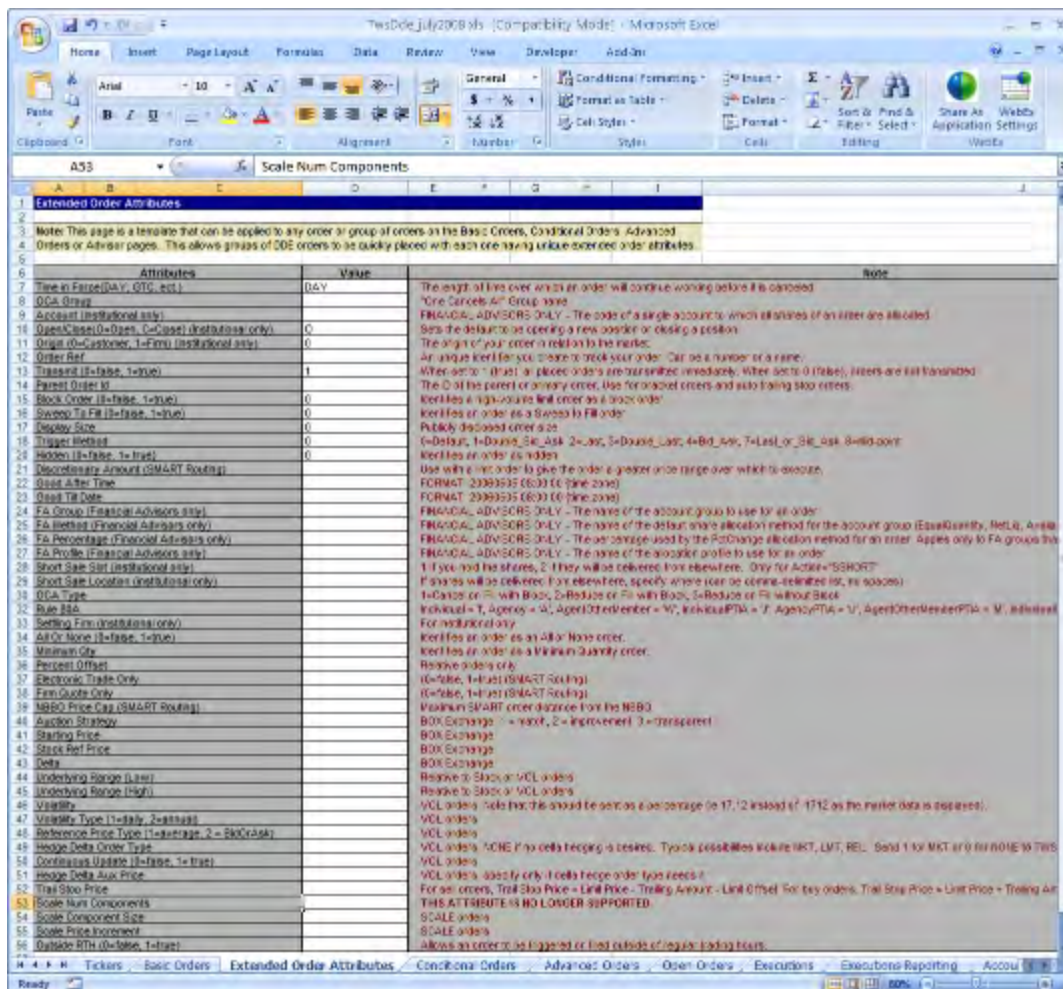
## Basic Orders Page Toolbar Buttons

The toolbar on the Basic Orders page includes the following buttons:

Button	Description
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Place/Modify Orders</b>	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
<b>Cancel Order</b>	This button cancels the order(s) you have highlighted.
<b>Apply Extended Template</b>	Applies the current values on the Extended Order Attributes page to the highlighted order row.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Extended Order Attributes Page

The Extended Order Attributes page includes all of the optional attributes you can use when you send an order, such as setting a display size to create an iceberg order, adding orders to an OCA group, and setting the transmit date for a Good After Time order. Once you define the attributes on this page, you can apply them to a single order or selected group of orders using the **Apply Extended Template** button, which occurs on both the Orders page and the Conditional Orders page. The attributes populate the extended order attributes fields that follow the *Order Status* fields to the far right of the page.



## Manually Program Extended Order Attributes

Observe the following guidelines when you manually assign an attribute:

- When appended to orderDescription, the number and order of attributes cannot be changed.
- For any attribute that is not defined, use the value 'EMPTY' or {}. Since a string length is limited to 255 characters, we recommend using the open/close curly braces {}.
- A place order message for a simple stock limit day order looks as follows, with the primary exchange "Supersoes" separating the extended attributes:

[illegible]

## Apply Extended Order Attributes to Individual Orders and Groups of Orders

Normally, values that you enter on the Extended Order Attributes page apply to all subsequent orders. However, you also can apply selected attributes to an individual order or a group of orders on the Orders page.

**Note:** You can also use this procedure to apply extended order attributes to orders on the Conditional Orders page.

### To apply extended order attributes to individual orders or a group of orders

1. Enter the value or values on the Extended Order Attributes page that you want to apply to an individual order or group of orders.
2. On the Orders page, select the order or group of orders.
3. Click the **Apply Extended Template** button.

The extended order attributes are applied to the order(s) and the values you entered on the Extended Order Attributes page are added to the corresponding fields in the *Extended Order Attributes* section of the Orders page.

When you place the order or group of orders, the extended order attribute values you entered are applied to the order.

For example, you might want to assign a unique Order Ref number to a group or basket of orders. To do this, you would enter the number for the Order Ref attribute on the Extended Order Attributes page, then select all the orders in the group on the Orders page and click Apply Extended Template.

4. Delete the value of the extended order attributes you used for the order from the Extended Order Attributes page. These values will still apply to all subsequent orders that you place from the DDE for Excel API spreadsheet unless you remove the value.

### Extended Order Attributes

The following table shows the available extended order attributes.

Attribute	Valid Values
timeInForce	DAY GTC OPG IOC GTD FOK DTC
ocaGroup	String that identifies an OCA (One Cancels All) group
account	String (for institutions)
open/close	O, C (for institutions)
origin	0, 1 (for institutions)
orderRef	String

transmit	<p>Specifies whether the order is transmitted immediately (set to 1) or not (set to 0).</p> <p>This parameter can be useful for example when working with basket orders. First, prepare a basket of orders (untransmitted), then when ready, set the value of the transmit parameter of each order to 1 to transmit the basket for execution.</p>
parentId	String (the order ID used for the parent order, use for bracket and auto trailing stop orders)
blockOrder	<p>0 (not a block order)</p> <p>1 (this is a block order)</p>
sweepToFill	<p>0 (not a sweep-to-fill order)</p> <p>1 (this is a sweep-to-fill order)</p>
displaySize	Publicly disclosed order size for iceberg orders. The value is a number that should be stored as a String.
triggerMethod	<p>Specifies how simulated Stop, Stop-Limit, and Trailing Stop orders are triggered:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 1 - use "double bid/ask" method, where stop orders are triggered based on two consecutive bid or ask prices.</li> <li>• 2 - "last" method, where stop orders are triggered based on the last price.</li> <li>• 3 - "double-last" method, where stop orders are triggered based on last two prices.</li> <li>• 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.</li> <li>• 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.</li> <li>• 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.</li> </ul> <p>For a complete description of Trigger Methods, see <a href="#">Modify the Trigger Method</a> in the Trader Workstation Users' Guide.</p>
hidden	<p>0</p> <p>1 (order not visible when viewing market depth)</p>

Discretionary Amount (SMART Routing)	Used in conjunction with a limit order to give the order a greater price range over which to execute.
Good After Time	Enter the date and time after which the order will become active. Use the format YYYYMMDD hh:mm:ss TMZ, where TMZ is optional three-letter time zone identifier. Allowed timezones are listed <a href="#">here</a> .
Good Till Date	The order continues working until the close of market on the date you enter. Use the format YYYYMMDD. To specify a time of day to close the order, enter the time using the format HH:MM:SS. Specify the time zone using a valid three-letter acronym.
FA Group	For Advisor accounts only. The name of the Financial Advisor group to which the trade will be allocated to. Use an empty String if not applicable.
FA Method	For Advisor accounts only. The share allocation method. <ul style="list-style-type: none"> <li>• EqualQuantity</li> <li>• NetLiq</li> <li>• AvailableEquity</li> <li>• PctChange</li> </ul>
FA Percentage	For Advisor accounts only. The share allocation percentage.
FA Profile	For Advisor accounts only. The name of the Share Allocation profile.
Short Sale Slot	For institutions only. Valid values are 1 (broker holds shares) and 2 (shares come from elsewhere).
Short Sale Location	Institutional accounts only.
OCA Type	1 = Cancel on Fill with Block 2 = Reduce on Fill with Block 3 = Reduce on Fill without Block

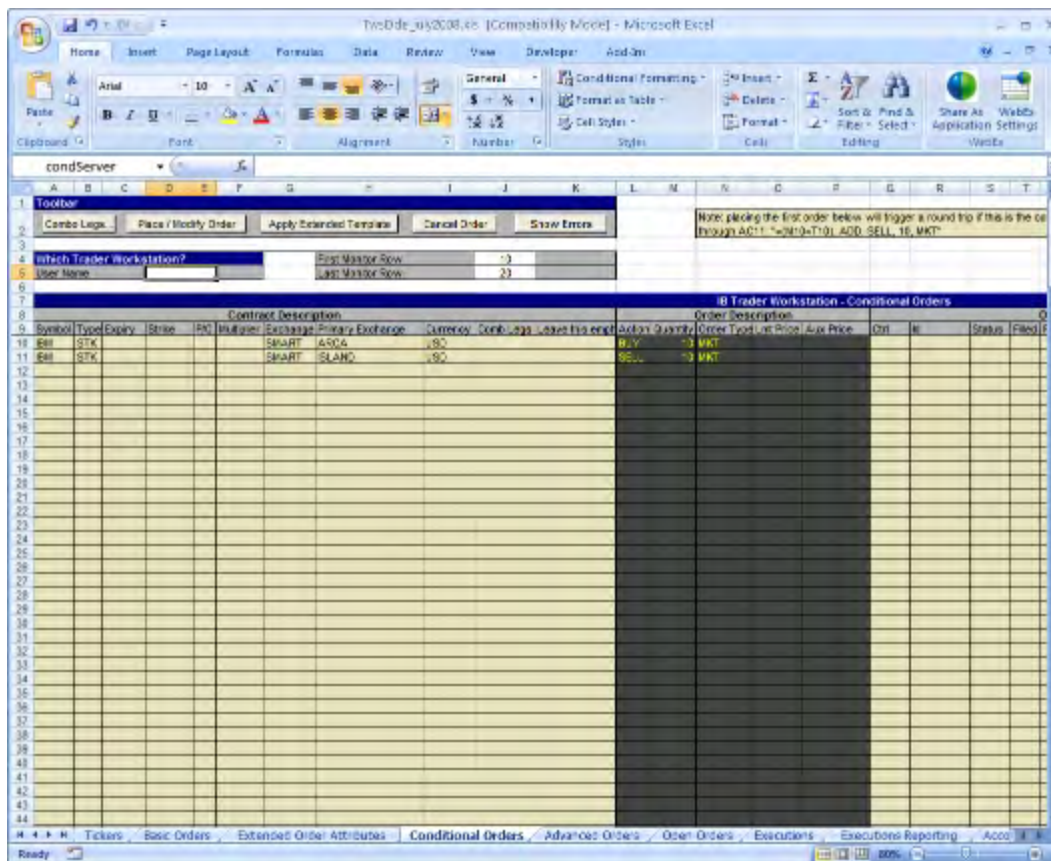
Rule 80A	<ul style="list-style-type: none"> <li>• Individual = 'I'</li> <li>• Agency = 'A',</li> <li>• AgentOtherMember = 'W'</li> <li>• IndividualPTIA = 'I'</li> <li>• AgencyPTIA = 'U'</li> <li>• AgentOtherMemberPTIA = 'M'</li> <li>• IndividualPT = 'K'</li> <li>• AgencyPT = 'Y'</li> <li>• AgentOtherMemberPT = 'N'</li> </ul>
Settling Firm	Institutions only
All or None	0 = false 1 = true
Minimum Qty	Identifies the order as a minimum quantity order.
Percent Offset	The percent offset for relative orders.
Electronic Trade Only	0 = false 1 = true
Firm Quote Only	0 = false 1 = true
NBBO Price Cap	Maximum SMART order distance from the NBBO.
Auction Strategy	match = 1 improvement = 2 transparent = 3 For BOX exchange only.
Starting Price	The starting price. For BOX orders only.
Stock Ref Price	Used for VOL orders to compute the limit price sent to an exchange (whether or not Continuous Update is used), and for price range monitoring. Also used for price improvement option orders.
Delta	The stock delta. For BOX orders only.
Underlying Range (Low)	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
Underlying Range (High)	The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.

Volatility	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.
Volatility Type	1 = daily 2 = annual
Reference Price Type	1 = average 2 = BidOrAsk
Hedge Delta Order Type	Prior to TWS Release 859, use "1" to send a market order, "0" for no order. After TWS 859, enter an accepted order type such as: MKT, LMT, REL or MTL.
Continuous Update	0 = false 1 = true
Hedge Delta Aux Price	Enter the Aux Price for Hedge Delta order types that require one.
Trail Stop Price	Used for Trailing Stop Limit orders only. This is the stop trigger price for TRAILLMT orders.
Scale Component Size	Used for Scale orders only, this value defines the order size of the each order component.
Scale Price Increment	Used for Scale orders only, this value is used to calculate the per-unit price of each component in the order. This cannot be a negative number.
Outside RTH	0 = false 1 = true

### Conditional Orders Page

Use the Conditional Orders page to create an order whose submission is contingent on other conditions being met, for example, an order based on a prior fill or a change in the bid or ask price. To see the Conditional Statement fields (in blue), use the scroll bar on the bottom of the page to scroll to the right.





## Setting Up Conditional Orders

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To set up a conditional order

1. On the **Conditional Orders** page, first create the order you want transmitted when a condition is met by defining the contract in the *Contract Description* fields, and then using the *Order Description* area to set up the order parameters.
2. In the blue *Condition Statements* area, use the *Statement* field to set the criteria which must be met to trigger the order. When the Statement = TRUE, your order will be submitted.

The sample spreadsheet includes a pair of orders, with the second orders transmission depending on the first order being completely filled. In this case, the Statement field trigger is that the value in cell T10 (the *Filled* field) must be equal to the value in M10 (the order *Quantity* field).

3. Type **ADD** in the *ADD/MOD* field because you are creating a one-time order.

4. Define the remaining order parameters just as you did in the *Order Description* area.

IB Trader Workstation - Conditional Orders												
Option				Order Description					Order			
Primary Exchange	Currency	Comb Legs	Leave this empty	Action	Quantity	Order Type	Lmt Price	Aux Price	Ctrl	Id	St	Filled
ARCA	USD			BUY	10 MKT				0	id	Fi	10
ISLAND	USD			SELL	10 MKT				0	id	Fi	10
	usd			buy	200 lmt		81.00					

5. Complete the necessary fields on the **Conditional Orders** page according to the syntax in the following table.

Field	Description
<i>Statement</i>	An Excel function which returns a true or false. When true, the order will be submitted; when false, nothing happens.
<i>ADD/MOD</i>	Use ADD for a one-time order. Use MOD to continue checking and modifying the order until it is completely filled. This is the field that activates a conditional order, and orders will be activated only with the "ADD" or "MOD" tags.
<i>Action</i>	BUY SELL
<i>Quantity</i>	Enter the quantity of the order.
<i>Order Type</i>	Refer to <a href="#">list of supported order types</a> .
<i>Lmt Price</i>	The limit price for Limit and Stop Limit order types.
<i>Aux. Price</i>	The stop-election price for Stop and Stop Limit order types, or the offset for relative orders.

All of the fields described above may be variables that depend on other cells, so any type of conditional order may be created.

## Conditional Order Examples

### If-Filled order

An if-filled order is an order that executes if a prior order executes. To create an if-filled order with the condition "If a Buy order fully executes, enter a sell limit order at a price of \$50.00":

Field	Value
<i>Statement</i>	Filled cell = 100
<i>ADD/MOD</i>	ADD
<i>Action</i>	SELL

<i>Quantity</i>	100
<i>Order Type</i>	LMT
<i>Lmt Price</i>	50
<i>Aux. Price</i>	empty

### Price-change order

A price-change order will be triggered if a specific bid or ask price is greater than, less than or equal to a specific price. To create a price change order with the condition "If the bid price drops below 81.20, submit a buy limit order for 200 shares with a limit price of \$81.10:

Field	Value
<i>Statement</i>	On the <b>Tickers</b> page, put your cursor in the bid price field you want to use, then copy the value that appears in the formula bar ("=" entry field) at the top of the spreadsheet. This value looks something like this: =username tik!id4?bid where "4" identifies the bid price for a specific contract. Paste this in the formula bar ("=" entry field) for the Statement, and add your qualifier, "=" ">" or "<" followed by the price. In this example, the formula would be: =username tik!id4?bid<81.20
<i>ADD/MOD</i>	ADD
<i>Action</i>	BUY
<i>Quantity</i>	200
<i>Order Type</i>	LMT
<i>Lmt Price</i>	81.10
<i>Aux. Price</i>	Not used in this example.

### To modify an order (or basket of orders)

- Select the order or a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.
- Click the **Place/Modify Order** button.
- Change any necessary parameters, then click the **Place/Modify Order** button.

## Conditional Orders Page Toolbar Buttons

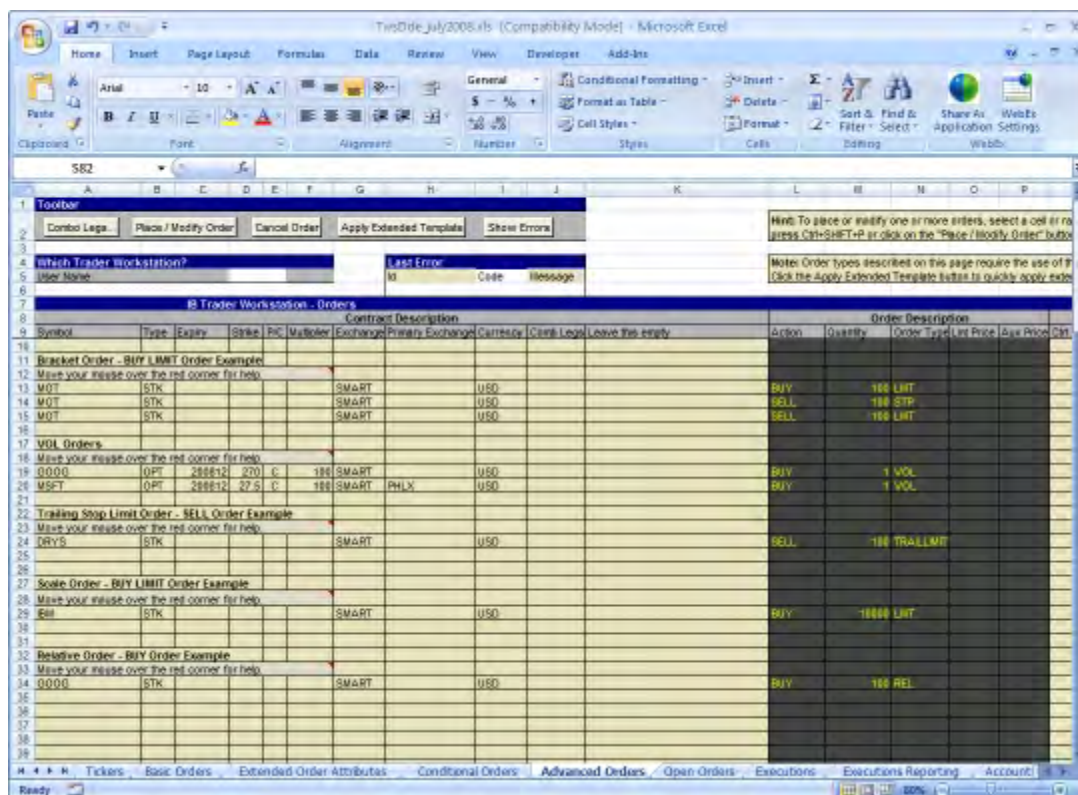
The toolbar on the Conditional Orders page includes the following buttons:

Button	Description
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Place/Modify Order</b>	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
<b>Apply Extended Template</b>	Applies all attributes on the Extended Order Attributes page to the selected order(s).
<b>Cancel Order</b>	This button cancels the order(s) you have highlighted.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

## Advanced Orders Page

Use the Advanced Orders page to create complex orders that require the use of extended order attributes, including:

- Bracket orders
- VOL orders
- Trailing Stop Limit Orders
- Scale Orders
- Relative Orders



For more information about using extended order attributes for individual orders or groups of orders, see [Apply Extended Order Attributes to Individual Orders and Groups of Orders](#)

## Placing a Bracket Order

Bracket orders in the DDE for Excel sample spreadsheet require the use of the extended order attributes *Transmit* and *Parent Order Id*. You must turn *Transmit* off until the order is completely set up, and you must identify the first order in the bracket as the Parent Order.

### To place a Buy-Limit bracket order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Enter the contract descriptions and order descriptions for all three orders on three contiguous rows:
  - The first order should be a BUY LMT order.
  - The second order should be a SELL STP order.
  - The third order should be a SELL LMT order.
2. Click the **Extended Order Attributes** tab. Change the value for *Transmit* to **0** (row 13 on the Extended Order Attributes page).

This ensures that your orders are not transmitted until you have completed the order setup.

3. Click the **Advanced Orders** tab, highlight the first order in the bracket order, then click the **Place/Modify Order** button.

The order is not executed, but the system generates an Order ID.

4. Copy the Order ID for the first order, omitting the “id” prefix, then click the **Extended Order Attributes** tab and paste the Order ID into the *Value* field for *Parent Order Id* (row 14). This value will be applied to all subsequent orders until you remove it from the Extended Order Attributes page.

The first order of the bracket order is now the primary order.

5. Click the **Advanced Orders** tab, highlight the second order, then click the **Place/Modify Order** button.

The order is not executed but is now associated with the primary order by means of the Parent Order Id extended order attribute.

6. Click the **Extended Order Attributes** tab and change the value for *Transmit* back to **1** (row 13).
7. Click the **Advanced Orders** tab, highlight the third order in the bracket order, then click the **Place/Modify Order** button. The entire bracket order is transmitted.
8. When you are done placing your bracket order, go to the **Extended Order Attributes** page and delete the *Parent Order Id* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Volatility Order

In the DDE for Excel sample spreadsheet, you place volatility (VOL) orders by entering values for the following extended order attributes:

- Volatility
- Volatility Type
- Reference Price Type
- Continuous Update
- Underlying Range (Low) - optional
- Underlying Range (High) - optional
- Hedge Delta Order Type - optional
- Hedge Delta Aux Price - optional

### To place a VOL order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.
  - Enter **VOL** in the *Order Type* field.
4. Click the **Extended Order Attributes** tab. Enter values in the *Value* field for the following extended order attributes:
  - *Volatility* - This value represents the volatility to use in calculating a limit price for the option. Enter this value as a percentage, not as the market data is displayed. For example, enter 17.12 instead of .1712.

- *Volatility Type* - Enter 1 for daily volatility or 2 for annual volatility.
  - *Reference Price Type* - This value is used to compute the limit price sent to an exchange and for stock range price monitoring. Enter 1 to use the average of the best bid and ask; or 2 to use NBB (bid) when buying a call or selling a put, or the NBO (ask) when selling a call or buying a put.
  - *Continuous Update* - Enter 1 to automatically update the option price as the underlying stock price (or futures price, for index options) moves. Enter 0 if you do not want to use this feature.
5. On the **Extended Order Attributes** page, enter values in the *Value* field for the following optional extended order attributes:
    - *Underlying Range (Low)* - Enter a low-end acceptable stock price relative to the selected option order. If the price of the underlying instrument falls below the lower stock range price, the option order will be canceled.
    - *Underlying Range (High)* - Enter a high-end acceptable stock price relative to the selected option order. If the price of the underlying instrument rises above the higher stock range price, the option order will be canceled.
    - *Hedge Delta Order Type* - Enter LMT, MKT or REL. Enter NONE if you do not want to use delta hedging.
    - *Hedge Delta Aux Price* - If you have entered LMT or REL as the Hedge Delta Order Type, enter the price as the value for this attribute.
  6. Click the **Advanced Orders** tab, then highlight the order row.
  7. Click the **Apply Extended Template** button. The values you entered for the extended order attributes are applied to the order row and displayed in the *Extended Order Attributes* section of the page.
  8. With the order row highlighted, click the **Place/Modify Order** button.
  9. When you are done placing VOL orders, go to the **Extended Order Attributes** page and delete the VOL order values you entered. If you do not, these values will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Trailing Stop Limit Order

In TWS, there are four values that make up a trailing stop limit order:

- trailing amount
- stop price
- limit price
- limit offset

In the DDE for Excel API spreadsheet, you enter the trailing amount, stop price and limit price. There is no field or extended order attribute for the limit offset value. You must include the limit offset in the stop price (the *Trail Stop Price* extended order attribute).

### To create a Trailing Stop Limit Order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.



- Enter **BUY** or **SELL** in the *Action* field.
  - Enter the limit price in the *Lmt Price* field.
  - Enter **TRAILLIMIT** in the *Order Type* field.
  - Enter the trailing amount in the *Aux Price* field.
4. Click the **Extended Order Attributes** tab. Specify the trailing stop price as an extended order attribute. Type this value in the Trail Stop Price *Value* field.  
The Trail Stop Price value must include the limit offset.
    - For a sell order:  
**Trail Stop Price = Limit Price - Trailing Amount - Limit Offset**
    - For a buy order:  
**Trail Stop Price = Limit Price + Trailing Amount + Limit Offset**
  5. On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The *Trail Stop Price* value is applied to the selected order and displayed in the *Trail Stop Price* field in the *Extended Order Attributes* section of the page.
  6. Click the **Place/Modify Order** button.
  7. When you are done placing your order, go to the **Extended Order Attributes** page and delete the *Trail Stop Price* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Scale Order

In the DDE for Excel sample spreadsheet, you place scale orders by entering values for the following extended order attributes:

### To place a scale order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields. The order type should be LMT or REL.
4. Click the **Extended Order Attributes** tab. Enter values in the *Value* field for the following extended order attributes:
  - *Scale Component Size* - Enter the size of the first, or initial, order component. For example, if you submit a 10,000-share order with a Scale Component Size value of 1000, the first component will be for 1000 shares.
  - *Scale Price Increment* - Enter the amount used to calculate the per-unit price of each component in the scale ladder. This cannot be a negative number.

**Note:** As of API Release 9.41, the *Scale Num Components* not supported.

3. On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The scale order values are applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
4. Click the **Place/Modify Order** button.



- When you are done placing your order, go to the **Extended Order Attributes** page and delete the scale order values you entered. If you do not, these values will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Relative Order

In the DDE for Excel sample spreadsheet, you place relative orders by entering a value for the *Percent Offset* extended order attribute.

### To place a relative order

- Click the **Advanced Orders** tab at the bottom of the spreadsheet.
- Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
- Select a contract and set up the order using the *Order Description* fields.
  - Enter **REL** as the order type.
  - Enter the price cap in the *Lmt Price* cell.
- Click the **Extended Order Attributes** tab. Enter a percentage in decimal form in the *Value* field for the *Percent Offset* extended order attribute.
- On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The percent offset value is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
- Click the **Place/Modify Order** button.
- When you are done placing your order, go to the **Extended Order Attributes** page and delete the *Percent Offset* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

## Advanced Orders Page Toolbar Buttons

The toolbar on the Advanced Orders page includes the following buttons:

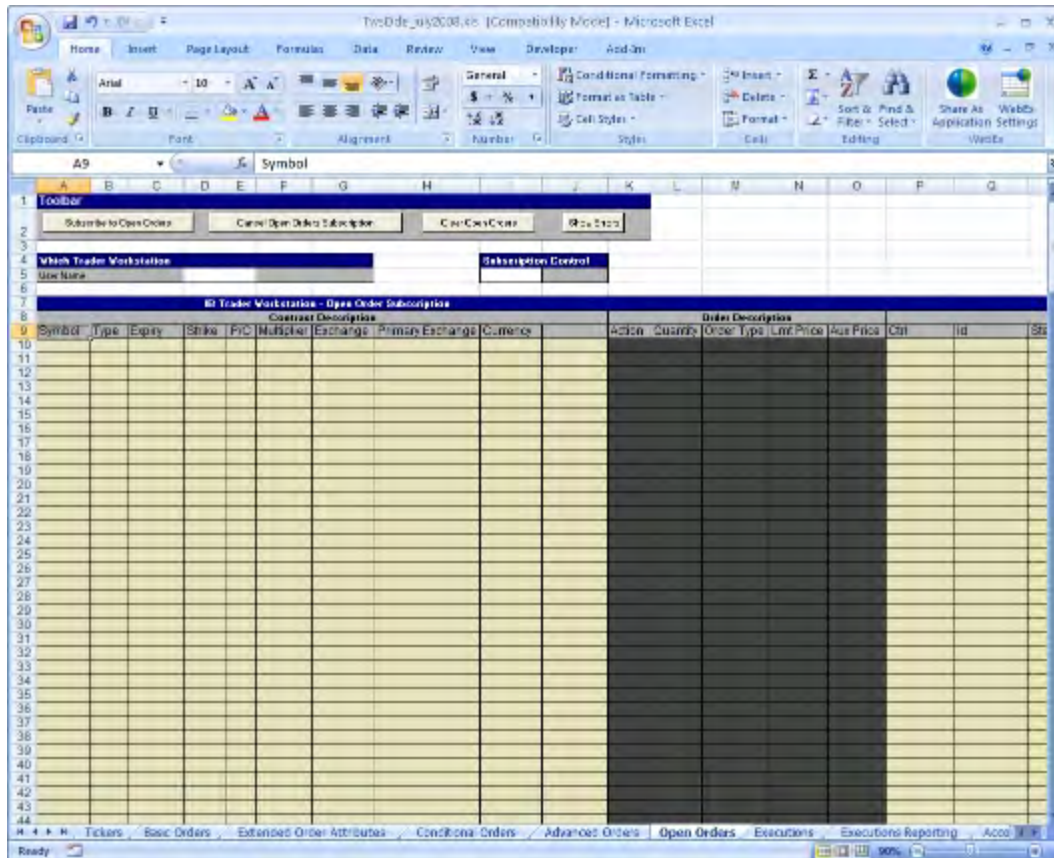
Button	Description
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Place/Modify Orders</b>	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
<b>Cancel Order</b>	This button cancels the order(s) you have highlighted.
<b>Apply Extended Template</b>	Applies the current values on the Extended Order Attributes page to the highlighted order row.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

## Open Orders Page

The Open Orders page shows you all transmitted orders, including those that have been accepted by the IB system, and those that are working at an exchange. Once you have subscribed, the page is updated each time you submit a new order,

either through the API or in TWS.

Once an order executes, it remains on the Open Orders page for 30 seconds, with the Status value changed to FILLED. Then the filled order is cleared and you can see it on the Executions page if you subscribed to real-time executions.



## Viewing Open Orders

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To view open orders:

1. Click the **Open Orders** tab at the bottom of the spreadsheet.
2. Click **Subscribe to Open Orders** on the toolbar.

All of your open orders are displayed on the page, including orders you enter in the Excel API spreadsheet and in TWS.

Orders that fill remain on the page for 30 seconds with a value of *Fill* in the *Status* field.

### To remove open orders

1. Click the **Cancel Open Orders Subscription** button on the toolbar.
2. Click the **Clear Open Orders** button.

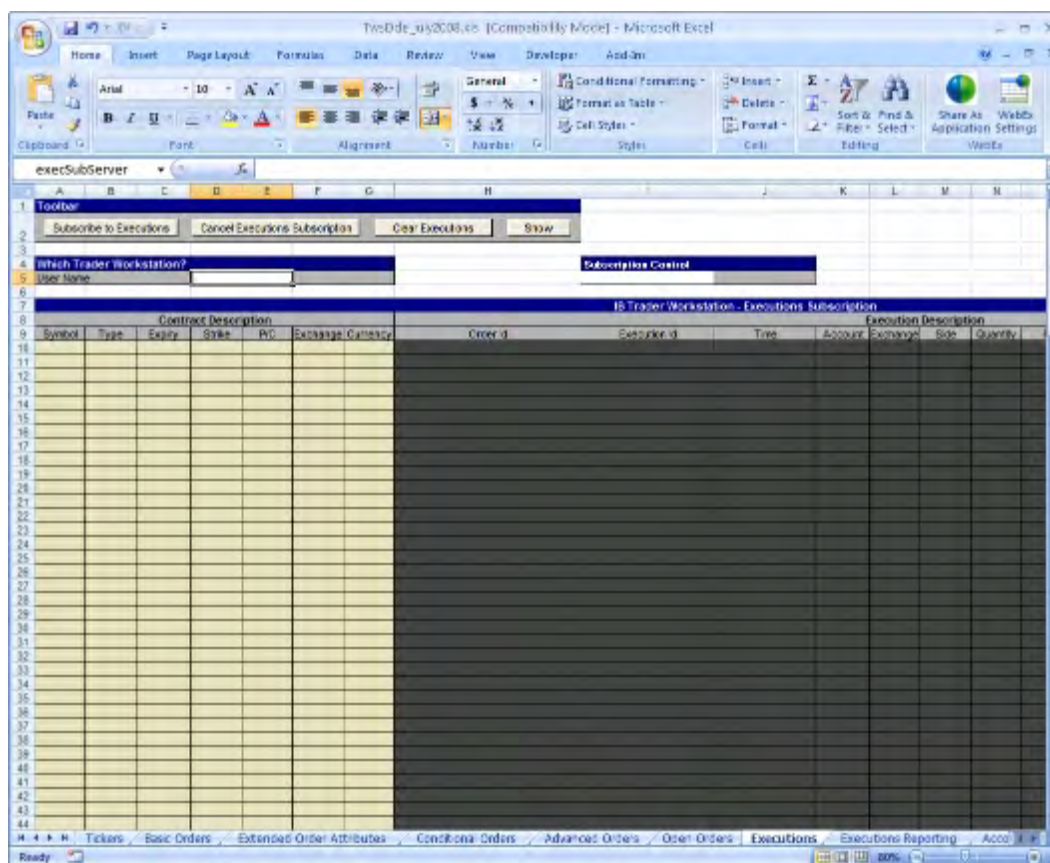
## Open Orders Tab Toolbar Buttons

The toolbar on the Open Orders page includes the following buttons:

Button	Description
<b>Subscribe to Open Orders</b>	Once you enter a valid user name, clicking this button queries TWS and returns all open orders. Once you subscribe to open orders, this page updates each time there is a new open order.
<b>Cancel Open Orders Subscription</b>	Cancels the open orders subscription. The page will no longer show your open orders.
<b>Clear Open Orders</b>	Removes all open orders from the page.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

## Executions Page

When you subscribe to executions, the Executions page displays information about all completed trades (also called “execution reports”).



## Viewing Executions

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To view executions

1. Click the Executions tab at the bottom of the spreadsheet.
2. Click the **Subscribe to Executions** button in the toolbar.

### To remove execution reports

1. Click the **Cancel Executions Subscription** button on the toolbar.
2. Click the **Clear Executions** button.

## Executions Page Toolbar Buttons

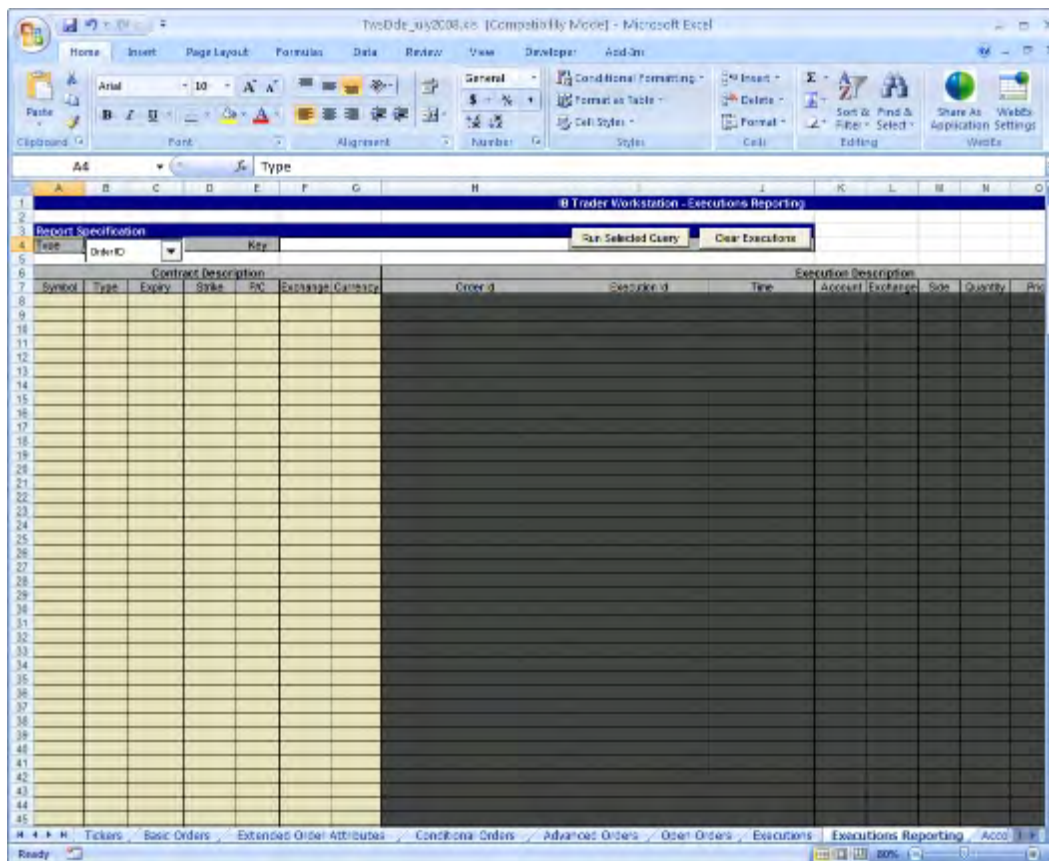
The toolbar on the Executions page includes the following buttons:

Button	Description
<b>Subscribe to Executions</b>	After you have entered a valid user name, this button queries TWS and returns information about all valid executions. After you subscribe to executions, this page updates each time an order executes.
<b>Cancel Executions Subscription</b>	Click to cancel the execution subscription.
<b>Clear Executions</b>	Removes all execution reports from the page.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

## Executions Reporting Page

Once you have subscribed to executions on the Executions page, you can use the Executions Reporting page to run reports based on an Order ID, Order Reference number, VOL order key, or strategy

From a programming point of view, the Executions Reporting page is a practical example of how you can extract array subscription data from the [named ranges](#) into which the data is put when it is received, and how such data can be used in your own custom DDE for Excel API applications.



## Running Execution Reports

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To run execution reports

1. On the Executions page, click the **Subscribe to Executions** button on the toolbar.
2. Click the **Executions Reporting** tab at the bottom of the worksheet.
3. In the *Type* field select from:
  - *Order ID* - finds all executions resulting from orders with a specified PermID.
  - *Order Ref* - finds all executions resulting from orders with a given order reference; for example executions from a specific basket order.
  - *VOL order* - finds all executions resulting from specific volatility order, including any hedge delta executions.
  - *Strategy* - in the *Key* field, enter a value to define the Type you selected. For example, if you selected *Order ID* as the type, enter a specific order ID in the *Key* field.

## Account Page

Use the Account page to:



- View account details including your current Equity with Loan Value and Available funds.
- View list of advisor-managed account codes.
- View your current portfolio.

[illegible]

## Using the Account Page

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To view account information

1. Click the **Account** tab at the bottom of the spreadsheet.
2. Click the **Subscribe to Account Updates** button on the toolbar.

## To remove account information

1. Click the **Cancel Account Subscription** button.
2. Click the **Clear Account Data** button.

### To request the list of Financial Advisor (FA) managed account codes

1. Click the **Account** tab at the bottom of the spreadsheet.
2. Click the **Request Managed Accounts** button.

### To request details of a Financial Advisor (FA) managed account

1. Click the **Account** tab at the bottom of the spreadsheet.
2. In the *Account Code* field in the *Which Trader Workstation?* area, type the account code for which you want details.
3. Click the **Request Managed Accounts** button.

### Account Page Toolbar Buttons

The toolbar on the Account page includes the following buttons.

Button	Description
<b>Subscribe to Account Updates</b>	Each click gives you data for a specific account value. All blank lines that precede the Account Portfolio section will hold data. Continue to click until all lines are populated.
<b>Cancel Account Subscription</b>	Click this button one time for each position you hold. When you get a line of "0's" you know you have downloaded all current positions. These values continue to update in real-time.
<b>Clear Account Data</b>	Clears all information from the page. You must first cancel your subscription before you can clear the data.
<b>Request Managed Accounts</b>	For advisor accounts, click this button one time for each account.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Account Page Values

The Account page displays the following values:

Field	Description	Notes
<i>Account Code</i>	The account number.	
<i>Account Ready</i>	For internal use only.	
<i>Account Type</i>	Identifies the IB account type.	
<i>Accrued Cash</i>	Reflects the current month's accrued debit and credit interest to date, updated daily.	At the beginning of each month, the past month's accrual is added to the cash balance and this field is zeroed out.

<i>Available Funds</i>	For securities: Equity with Loan Value - Initial margin For commodities: Net Liquidation Value - Initial margin	
<i>Buying Power</i>	Cash Account: (Minimum (Equity with Loan Value, Previous Day Equity with Loan Value)-Initial Margin) Standard Margin Account: Available Funds*4	
<i>Cash Balance</i>	For securities: Settled cash + sales at the time of trade For commodities: Settled cash + sales at the time of trade + futures PNL	
<i>Currency</i>	Shows the currency types that are listed in the Market Value area.	
<i>Cushion</i>	Shows your current margin cushion.	
<i>Day Trades Remaining</i>	Number of day trades left for pattern day trader period.	
<i>Day Trades Remaining T+1, T+2, T+3, T+4</i>	The number of day trades you have left for a 4-day pattern day-trader.	
<i>Equity With Loan Value</i>	For Securities: Cash Account: Settled Cash  Margin Account: Total cash value + stock value + bond value + (non-U.S. & Canada securities options value)  For Commodities: Cash Account: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures PNL)  Margin Account: Total cash value + commodities option value - futures maintenance margin requirement	
<i>Excess Liquidity</i>	Equity with Loan Value - Maintenance margin	
<i>Exchange Rate</i>	The exchange rate of the currency to your base currency.	



<i>Full Available Funds</i>	For securities: Equity with Loan Value - Initial margin  For commodities: Net Liquidation Value - Initial margin	
<i>Full Excess Liquidity</i>	Equity with Loan Value - Maintenance margin	
<i>Full Init Margin Req</i>	Overnight initial margin requirement in the base currency of the account.	
<i>Full Maint Margin Req</i>	Maintenance margin requirement as of next period's margin change in the base currency of the account.	
<i>Future Option Value</i>	Real-time mark-to-market value of futures options.	
<i>Futures PNL</i>	Real-time change in futures value since last settlement.	
<i>Gross Position Value</i>	Long Stock Value + Short Stock Value + Long Option Value + Short Option Value.	
<i>Init Margin Req</i>	Initial margin requirement in the base currency of the account.	
<i>Leverage</i>	For Securities: Gross Position value / Net Liquidation value  For Commodities: Net Liquidation value - Initial margin	
<i>Look Ahead Available Funds</i>	For Securities: Equity with loan value - look ahead initial margin.  For Commodities: Net Liquidation value - look ahead initial margin.	
<i>Look Ahead Excess Liquidity</i>	Equity with loan value - look ahead maintenance margin.	
<i>Look Ahead Init Margin Req</i>	Initial margin requirement as of next period's margin change in the base currency of the account.	
<i>Look Ahead Maint Margin Req</i>	Maintenance margin requirement as of next period's margin change in the base currency of the account.	
<i>Maint Margin Req</i>	Maintenance margin requirement in the base currency of the account.	

<i>Net Liquidation</i>	For Securities: Total cash value + stock value + securities options value + bond value  For Commodities: Total cash value + commodities options value	
<i>Net Liquidation by Currency</i>	Same as above for individual currencies.	
<i>Option Market Value</i>	Real-time mark-to-market value of securities options.	
<i>PNL</i>	The difference between the current market value of your open positions and the average cost, or Value - Average Cost.	
<i>Previous Day Equity with Loan Value</i>	Marginable Equity with Loan Value as of 16:00 ET the previous day, only applicable to securities.	
<i>Realized PnL</i>	Shows your profit on closed positions, which is the difference between your entry execution cost and exit execution cost, or (execution price + commissions to open the positions) - (execution price + commissions to close the position).	
<i>Reg T Equity</i>	Initial margin requirements calculated under US Regulation T rules.	
<i>Reg T Margin</i>	For Securities: Cash Account : Settled Cash Margin Account : Total cash value + stock value + bond value + (non-U.S. & Canada securities options value)  For Commodities: Cash Account : Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures PNL) Margin Account : Total cash value - futures maintenance margin requirement	
<i>SMA</i>	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	Only applicable for securities.

<i>Stock Market Value</i>	Real-time mark-to-market value of stock	
<i>Total Cash Balance</i>	Cash recognized at the time of trade + futures PNL	
<i>Total Cash Value</i>	Total cash value of stock, commodities and securities	

## Portfolio Page

The Portfolio page displays all of your current positions. This page communicates with TWS and updates the values every three minutes, which you can see in the *Last Update Time* field in the *Which Trader Workstation?* area of the page.

Symbol	Sec Type	Expiry	Strike	Right	Currency	Local Symbol	Position	Market Price	Market Value	Avg Cost	Unrealized P/L	Realized P/L
AA	STK				USD	AA	200	31.41555615	7116.2	36.24190475	-1222.6	
AB	STK				USD	AB	100	25.25555555	2525.6	32.6981818	-887.2	
AMZN	STK				USD	AMZN	100	72.8449555	7284.5	72.18	666.5	
AXP	STK				USD	AXP	100	39.86555555	3986.6	42.3918182	-252.5	
BAL	STK				USD	BAL	100	93.16555555	9316.6	73.3918182	-1132.1	
C	STK				USD	C	100	18.26555555	1826.6	26.2418182	-215.8	
DELL	STK				USD	DELL	100	24.65555555	2465.6	23.87	62	
DRYS	STK				USD	DRYS	100	73.87555555	7387.6	74.59	-62	
FB	FUT	20060919			USD	FB	100	103.25	10325.6	67577.4	-66673.5	
GE	STK				USD	GE	1000	1.55644444	1556.4	1.5917389	-35.1	
GO	STK				USD	GO	200	30.25	5950	32.70	-290	
GOOGL	OPF	20061219	270	CALL	USD	GOOGL	4	215.8444444	863.38	23.071	-6944	
GOOGL	STK				USD	GOOGL	50	482.6444444	24132.2	524.9633334	-6362.75	
IBM	OPF	20061017	135	PUT	USD	IBM	20	10.6666664	213.33	13.65	-13.65	
IBM	STK				USD	IBM	2500	27.75555555	69388.9	124.377777	-8258.98	
KEY	STK				USD	KEY	9990	11.51666664	114877.22	23.88911925	-121538.45	
MMM	STK				USD	MMM	200	71.58555555	14317.1	66.67	-2055	
WMT	STK				USD	WMT	400	7.46555555	2986.2	8.9475	-593	
MSFT	OPF	20061017	20	CALL	USD	MSFT	2	8.17555555	16.35	86.1999997	-80	
MSFT	OPF	20061017	20	PUT	USD	MSFT	15	0.14	2.1	8.53333325	-80	
MSFT	STK				USD	MSFT	400	26.6444444	10657.8	27.69333335	-634.39	
GOOGL	STK				USD	GOOGL	100	45.36666666	4536.7	48.00	-263.3	
SOL	STK				USD	SOL	1705	15.88555555	26944.94	17.69949715	-3434.01	
UNH	STK				USD	UNH	100	27.52444444	2752.5	34.42	-689.5	
USD	CASH				USD	USD	25000	1.02454444	25613.75	1.0055	478.25	
USD	CASH				USD	USD	25000	1.45225	36306.25	1.00955	881.88	
USD	CASH				USD	USD	25000	1.21125	30281.25	1.00883334	347750	
SOBE	STK				USD	SOBE	100	32.44444444	3244.5	33.99	-1154	
VHOO	STK				USD	VHOO	300	21.12444444	6337.3	23.6225	-1647.75	
Z	FUT	20061219			USD	Z	4	51.76755555	207.06	58.662.95	-17141.8	

## Viewing Your Portfolio

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To view your portfolio

1. Click the **Portfolio** tab at the bottom of the worksheet.
2. Click the **Subscribe to Portfolio Updates** button.

### To remove portfolio information

1. Click the **Cancel Portfolio Subscription** button.
2. Click the **Clear Portfolio Data** button.

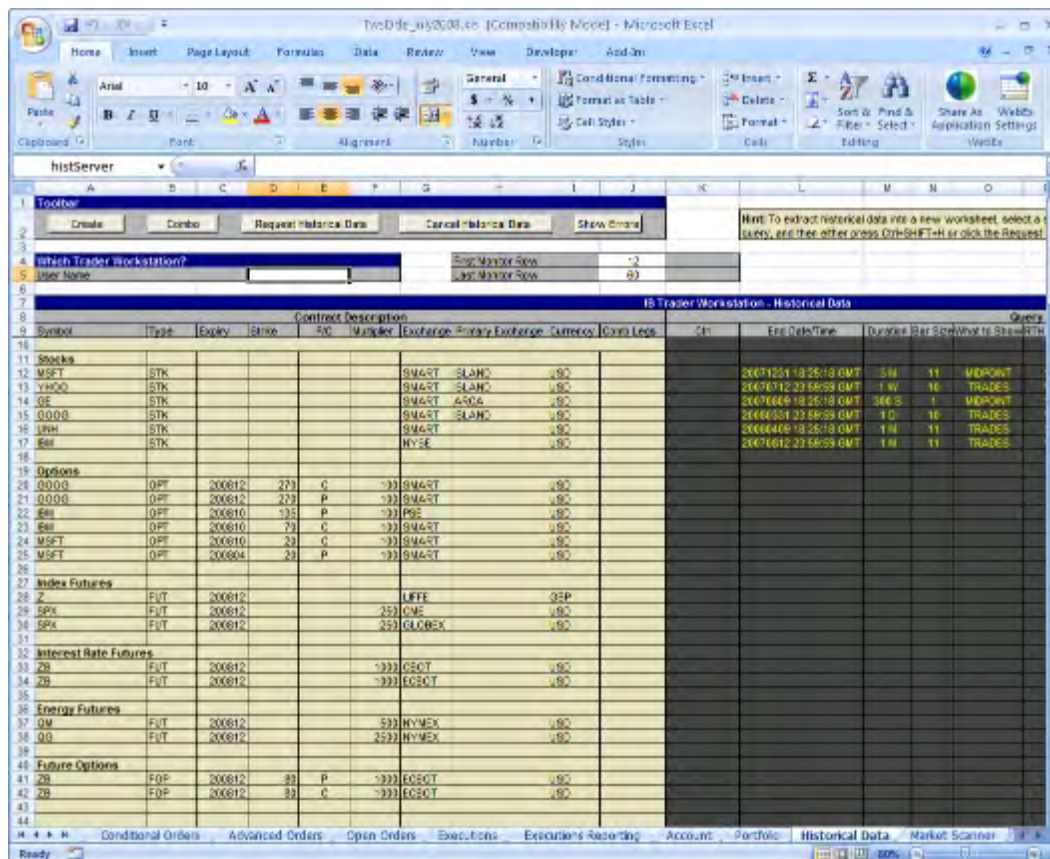
### Portfolio Page Toolbar Buttons

The toolbar on the Portfolio page includes the following buttons.

Button	Description
<b>Subscribe to Portfolio Updates</b>	Click to view all current portfolio data.
<b>Cancel Portfolio Subscription</b>	Cancels the connection to TWS that updates your portfolio information.
<b>Clear Portfolio Data</b>	Removes all data from the page. You must cancel your subscription before you can clear all data.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Historical Data Page

Use the Historical Data page to request historical data for an instrument based on data you enter in query fields. The query results display on a separate worksheet page and creates a new page for the results if the page doesn't currently exist. Note that since the query returns in a named range of cells, you can write VBA macros to perform computations on it, and you can chart and sort the data in Excel.



**Note:** For information about historical data request limitations, see [Historical Data Limitations](#).

## Viewing Historical Data

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To request historical data

1. Click the **Historical Data** tab at the bottom of the spreadsheet.
2. Create a ticker by filling in the fields in the *Contract Description* section of the page, or by clicking the **Create Ticker** button on the toolbar and entering the required information in the Ticker box.
3. Enter the parameters of your query in the *Query Specification* fields. For complete descriptions of the query fields, Historical Data Page Query Specification Fields.
4. Select the line, then click the **Request Historical Data** button. When the *Ctrl* field displays "Finished," the results are displayed on the specified page.

### To request historical data for expired contracts

1. On the Historical Data page, create a ticker by filling in the fields in the *Contract Description* section of the page, or by clicking the **Create Ticker** button on the toolbar and entering the required information in the Ticker box.
2. Enter the parameters of your query in the *Query Specification* fields.

3. In the *Expired* field in the Query Specification section, enter **TRUE**.
4. Select the line, then click the **Request Historical Data** button. When the *Ctrl* field displays "Finished," the results are displayed on the specified page.

**Note:** Historical data queries on expired contracts are limited to the last year of the life of the contract.

The following figure shows a typical historical data results page.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1													
2		DATE/TIME	20060609	OPEN	HIGH	LOW	CLOSE	VOLUME	WAP	HAS GAPS			
3		20060609 14:20:18		77.89	77.89	77.89	77.89	-1	-1	FALSE			
4		20060609 14:20:19		77.89	77.89	77.89	77.89	-1	-1	FALSE			
5		20060609 14:20:20		77.89	77.89	77.89	77.89	-1	-1	FALSE			
6		20060609 14:20:21		77.89	77.89	77.89	77.89	-1	-1	FALSE			
7		20060609 14:20:22		77.89	77.89	77.89	77.89	-1	-1	FALSE			
8		20060609 14:20:23		77.89	77.89	77.89	77.89	-1	-1	FALSE			
9		20060609 14:20:24		77.89	77.89	77.89	77.89	-1	-1	FALSE			
10		20060609 14:20:25		77.89	77.89	77.89	77.89	-1	-1	FALSE			
11		20060609 14:20:26		77.89	77.89	77.89	77.89	-1	-1	FALSE			
12		20060609 14:20:27		77.89	77.89	77.89	77.89	-1	-1	FALSE			
13		20060609 14:20:28		77.89	77.89	77.89	77.89	-1	-1	FALSE			
14		20060609 14:20:29		77.89	77.89	77.89	77.89	-1	-1	FALSE			
15		20060609 14:20:30		77.89	77.89	77.89	77.89	-1	-1	FALSE			
16		20060609 14:20:31		77.89	77.89	77.89	77.89	-1	-1	FALSE			
17		20060609 14:20:32		77.89	77.89	77.89	77.89	-1	-1	FALSE			
18		20060609 14:20:33		77.89	77.89	77.89	77.89	-1	-1	FALSE			
19		20060609 14:20:34		77.89	77.89	77.89	77.89	-1	-1	FALSE			
20		20060609 14:20:35		77.89	77.89	77.89	77.89	-1	-1	FALSE			
21		20060609 14:20:36		77.89	77.89	77.89	77.89	-1	-1	FALSE			
22		20060609 14:20:37		77.89	77.89	77.89	77.89	-1	-1	FALSE			
23		20060609 14:20:38		77.89	77.89	77.89	77.89	-1	-1	FALSE			
24		20060609 14:20:39		77.89	77.89	77.89	77.89	-1	-1	FALSE			
25		20060609 14:20:40		77.89	77.89	77.89	77.89	-1	-1	FALSE			
26		20060609 14:20:41		77.89	77.89	77.89	77.89	-1	-1	FALSE			
27		20060609 14:20:42		77.89	77.89	77.89	77.89	-1	-1	FALSE			
28		20060609 14:20:43		77.89	77.89	77.89	77.89	-1	-1	FALSE			
29		20060609 14:20:44		77.89	77.89	77.89	77.89	-1	-1	FALSE			
30		20060609 14:20:45		77.89	77.89	77.89	77.89	-1	-1	FALSE			
31		20060609 14:20:46		77.89	77.89	77.89	77.89	-1	-1	FALSE			
32		20060609 14:20:47		77.89	77.89	77.89	77.89	-1	-1	FALSE			
33		20060609 14:20:48		77.89	77.89	77.89	77.89	-1	-1	FALSE			

## Historical Data Page Toolbar Buttons

The toolbar on the Historical Data page includes the following buttons.

Button	Description
<b>Create Ticker</b>	Opens the Ticker box. Enter information to create a market data line.
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Request Historical Data</b>	Submits your historical data query to TWS and displays the results on a separate worksheet page.



<b>Cancel Historical Data</b>	Cancels the historical data request.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Historical Data Page Query Specification Fields

Historical Data queries include the following fields:

Parameter	Description
End Date/Time	Use the format <code>yyyymmdd {space}hh:mm:ss{space}tmz</code> where the time zone is allowed (optionally)after a space at the end.
Duration	<p>This is the time span the request will cover, and is specified using the format <i>integer {space} unit</i>, where valid units are:</p> <ul style="list-style-type: none"><li>• S (seconds)</li><li>• D (days)</li><li>• W (weeks)</li><li>• Y (years)</li></ul> <p>This unit is currently limited to one. If no unit is specified, seconds are used.</p>

Bar Size	<p>Specifies the size of the bars that will be returned. The following bar sizes may be used, and are specified using the parametric value:</p> <p><b>Bar Size String - Integer Value</b></p> <ul style="list-style-type: none"> <li>• 1 second - 1</li> <li>• 5 seconds - 2</li> <li>• 15 seconds - 3</li> <li>• 30 seconds - 4</li> <li>• 1 minutes - 5</li> <li>• 2 minutes - 6</li> <li>• 3 minutes - 16</li> <li>• 5 minutes - 7</li> <li>• 15 minutes - 8</li> <li>• 30 minutes - 9</li> <li>• 1 hour - 10</li> <li>• 1 day - 11</li> </ul> <p>On the query return page, each "bar" is represented by a line in the spreadsheet. If you specify a duration of 300 seconds, and a bar size of "1" (one second) your return will include 300 lines, and the value in each line is equal to one second, or is a one-second bar. Note that you can use either the Integer value of the Bar Size String or the Integer Value to define the bar sizes.</p>
What to Show	<p>Determines the nature of the data extracted. Valid values include:</p> <ul style="list-style-type: none"> <li>• TRADES</li> <li>• MIDPOINT</li> <li>• BID</li> <li>• ASK</li> <li>• BID_ASK</li> <li>• HISTORICAL_VOLATILITY</li> <li>• OPTION_IMPLIED_VOLATILITY</li> </ul> <p>All but the Bid/Ask data contain the <i>start time</i>, <i>open</i>, <i>high</i>, <i>low</i>, <i>close</i>, <i>volume</i> and <i>weighted average price</i> during the time slice queried.</p> <p>For the Bid/Ask query, the <i>open</i> and <i>close</i> values are the time-weighted average bid and the time-weighted average offer, respectively. These bars are identical to the TWS charts' candlestick bars.</p>



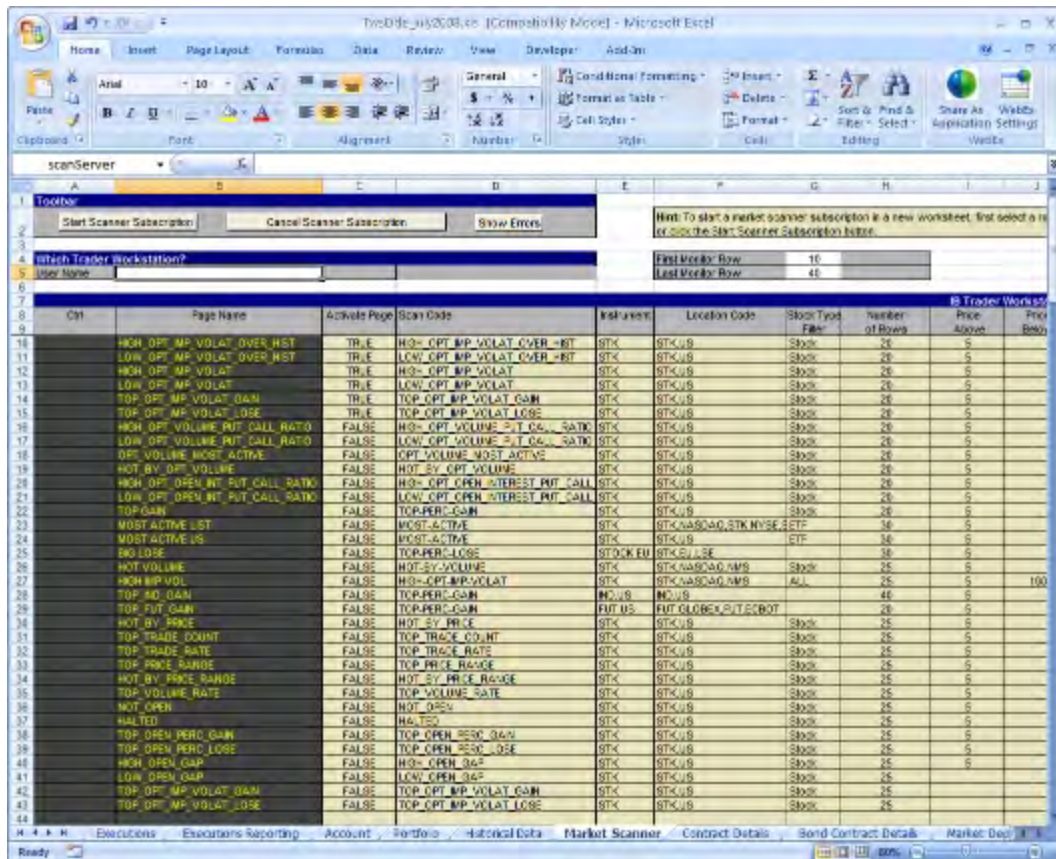
RTH Only	<p>Regular Trading Hours only. Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 - all data available during the time span requested is returned, including time intervals when the market in question was outside of regular trading hours.</li> <li>• 1 - only data within the regular trading hours for the product requested is returned, even if the time span falls partially or completely outside.</li> </ul>
Date Format Style	<p>Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 - dates that apply to bars are returned in the format <code>yyymmdd{space}{space}hh:mm:ss</code> (the same format used when reporting executions).</li> <li>• 2 - the dates are returned as an integer specifying the number of seconds since 1/1/1970 GMT.</li> </ul>
Page Name	The name of the results page. This appears in the tab for the results page at the bottom of the worksheet.
Expired	<p>Valid values: TRUE, FALSE</p> <p>If TRUE, the data query can be done on an expired futures contract, limited to the last year of a contract's life.</p>

For a request with a duration of 300 seconds and a bar of one second, the query return looks like this (the scroll bar on the right side of the page allows you to scroll down and see all 300 bars).

Note that the new page is added to the right of the existing tabs on the worksheet.

## Market Scanner Page

Use the Market Scanner page to subscribe to TWS market scanners. These scanners allow you to define criteria and set filters that return the top  $x$  number of underlyings which meet all scan criteria. The scan is continually updated in real time.



## Starting a Market Scanner Subscription

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To start a scanner subscription

1. Click the **Market Scanner** tab at the bottom of the spreadsheet.
2. Highlight an existing scanner row, or enter information for a different market scanner:
  - a. Type the name of the scan results page in the *Page Name* field.
  - b. Type **TRUE** or **FALSE** in the *Activate Page* field.

Setting these field to TRUE forces the scan results page to pop to the front of your application every time it updates. To stop this behavior, set the value of this field to FALSE.

- c. Type values for the rest of the scan parameters in the lightly shaded section of the page.
3. Click the **Start Scanner Subscription** button in the toolbar. A new page for the scanner is created and is displayed after the subscription is processed.

## Market Scanner Parameters

The following table describes the market scanner parameters that make up a scanner subscription.

Parameter	Description
<i>Page name</i>	The name that will be given to the new page that is created with the scanner data.
<i>Activate Page?</i>	If set to true, the new scanner page will display on top of the worksheet every time the scan results update. This could be as often as every minute.
<i>Scan Code</i>	The type of scan.
<i>Instrument</i>	The instrument type used in the scan.
<i>Location code</i>	The market used (i.e. US Stocks) for the scan.
<i>Stock type filter</i>	Allows you to specify just stocks, just ETFs, or both.
<i>Number of rows</i>	The number of rows of data to return in the results.
<i>Price Above</i>	Filters out returns with prices below the named price. Can be left empty.
<i>Price Below</i>	Filters out returns with prices above the named price. Can be left empty.
<i>Average volume above</i>	Filters out returns with an average volume below the named price. Can be left empty.
<i>Average Option Volume Above</i>	Filters out returns with an average option volume below the named price. Can be left empty.
<i>Market Cap Above</i>	Filters out returns with a market capitalization value below the named price. Can be left empty.
<i>Market Cap Below</i>	Filters out returns with a market capitalization value above the named price. Can be left empty.
<i>Moody Rating Above</i>	Filters out returns with a Moody rating below the named price. Can be left empty.
<i>Moody Rating Below</i>	Filters out returns with a Moody rating above the named price. Can be left empty.
<i>S &amp; P Rating Above</i>	Filters out returns with an S&P rating below the named price. Can be left empty.
<i>S &amp; P Rating Below</i>	Filters out returns with an S&P rating above the named price. Can be left empty.
<i>Maturity Date Above</i>	Filters out returns with a maturity date below the named price. Can be left empty.

<i>Maturity Date Below</i>	Filters out returns with a maturity date above the named price. Can be left empty.
<i>Coupon Rate Above</i>	Filters out returns with a coupon rate below the named price. Can be left empty.
<i>Coupon Rate Below</i>	Filters out returns with a coupon rate above the named price. Can be left empty.
<i>Exclude Convertible</i>	Filters out convertible bonds. Can be left empty.
<i>Scanner Settings Pairs</i>	For example "Annual/True" used on the Top Option Implied Vol% Gainers would instruct the scan to return annualized volatilities. Delimit setting pairs by slashes.

### Market Scanner Page Toolbar Buttons

The toolbar on the Market Scanner page includes the following buttons.

Button	Description
<b>Start Scanner Subscription</b>	Creates and displays a new page for results of the selected market scanner.
<b>Cancel Scanner Subscription</b>	Cancels the market scanner.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Available Market Scanners

The following table shows the available market scanners in the DDE for Excel API spreadsheet.

**Note:** To get more detailed market scan results than are available in the DDE for Excel API spreadsheet, run the Market Scanners in TWS.

Market Scanner (Scan Code)	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 Historical (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 current 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 <b>highest</b> 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 <b>lowest</b> 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)	<p>Contracts with the highest trading volume today, based on units used by TWS (lots for US stocks; contract for derivatives and non-US stocks).</p> <p>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.</p>

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: <ul style="list-style-type: none"> <li>• today's Volume/avgDailyVolume is highest.</li> <li>• avgDailyVolume is a 30-day exponential moving average of the contract's daily volume.</li> </ul>
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: <ul style="list-style-type: none"> <li>• (lastTradePrice-prevClose)/avgDailyChange is highest in absolute value (positive or negative).</li> <li>• The avgDailyChange is defined as an exponential moving average of the contract's (dailyClose-dailyOpen)</li> </ul>
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.

<b>EFP - High Synth Bid Rev Yield</b> (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.
<b>EFP - Low Synth Bid Rev Yield</b> (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.

\* 30-day (V30) Implied Volatilities:

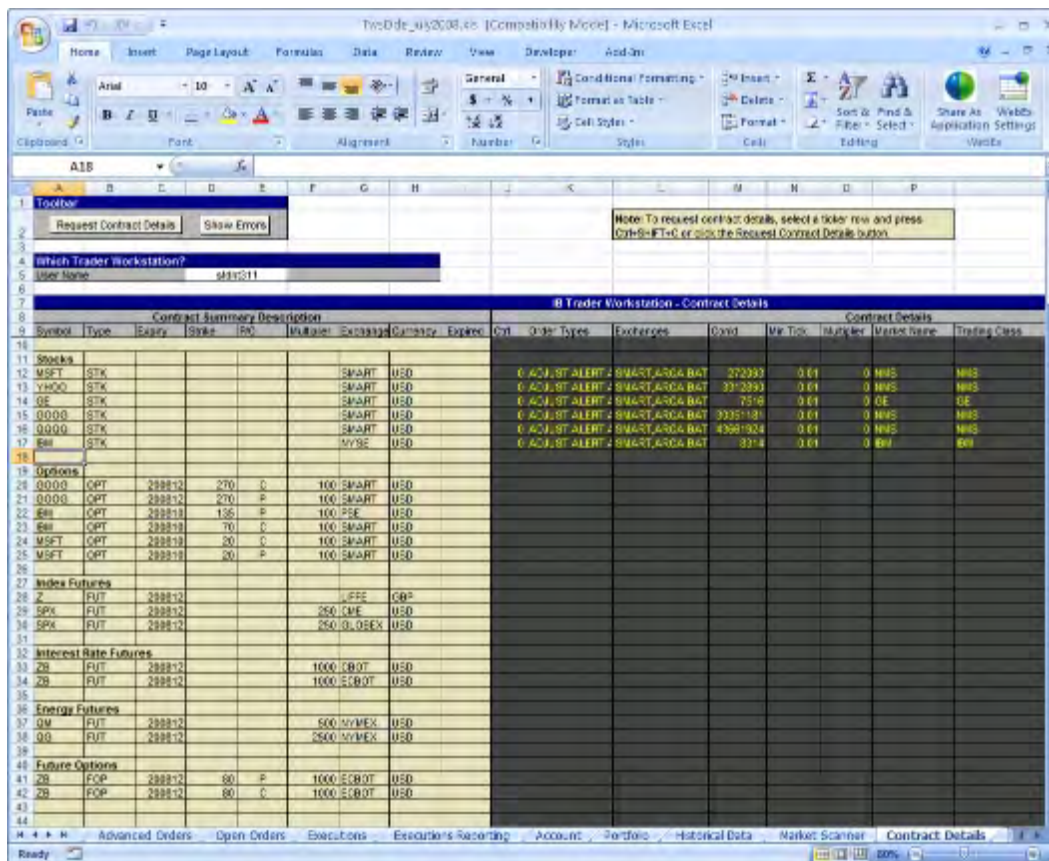
Implied volatility is calculated using a 100-step binary tree for American style options, and a Black-Scholes model for European style options. Interest rates are calculated using the settlement prices from the day's Eurodollar futures contracts, and dividends are based on historical payouts.

The IB 30-day volatility is the at-market volatility estimated for a maturity thirty calendar days forward of the current trading day. It is based on option prices from two consecutive expiration months. The first expiration month is that which has at least eight calendar days to run. The implied volatility is estimated for the eight options on the four closest to market strikes in each expiry. The implied volatilities are fit to a parabola as a function of the strike price for each expiry. The at-the-market implied volatility for an expiry is then taken to be the value of the fit parabola at the expected future price for the expiry. A linear interpolation (or extrapolation, as required) of the 30-day variance based on the squares of the at-market volatilities is performed. V30 is then the square root of the estimated variance. If there is no first expiration month with less than sixty calendar days to run, we do not calculate a V30.

## Contract Details Page

Use the Contract Details page to request contract-specific information such as supported order types, valid exchanges, the contract ID, and so on.





## Requesting Contract Details

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To request details for a contract

1. Click the **Contract Details** tab at the bottom of the spreadsheet to open the Contract Details page.
2. Select or enter the ticker symbol for which you want to request contract details.
3. To request contract details for an expired contract, type **TRUE** in the *Expired* field.
4. Click the **Request Contract Details** button on the toolbar.

## Contract Details Page Toolbar Buttons

The toolbar on the Contract Details page includes the following buttons:

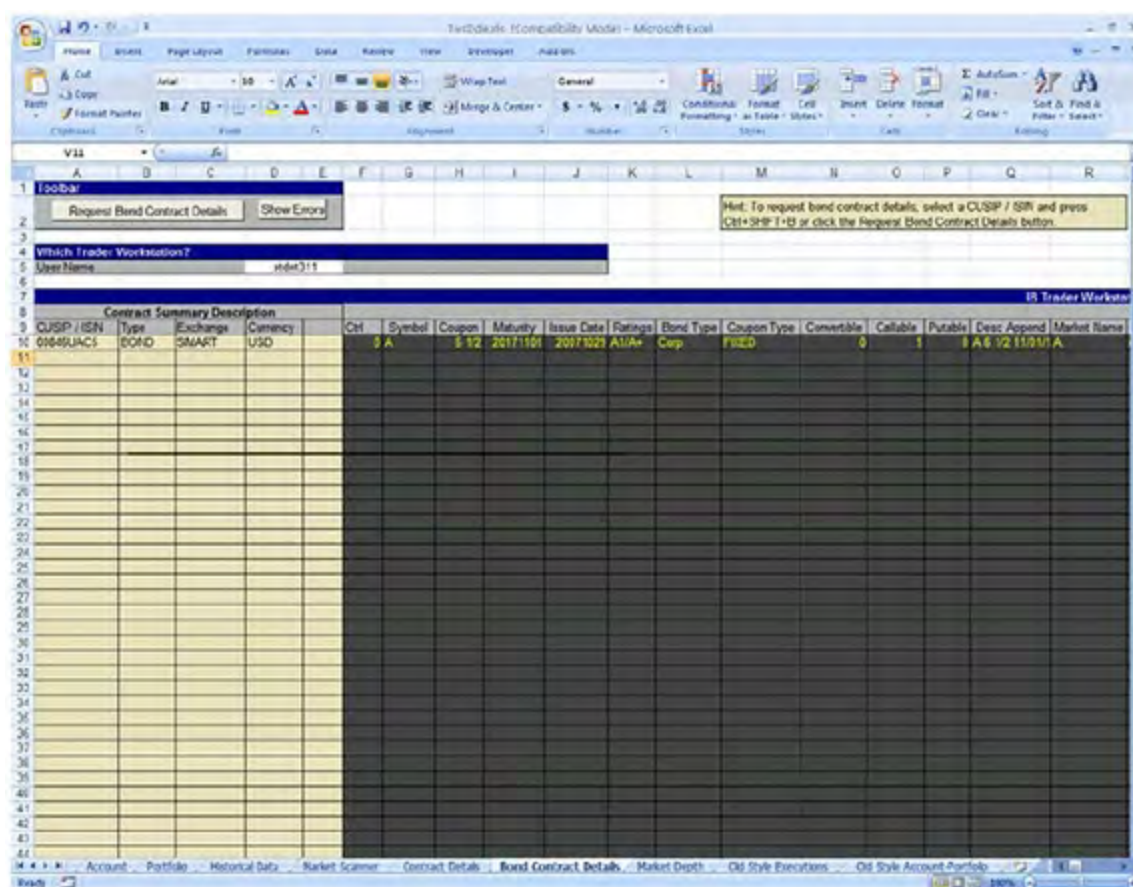
Button	Description
<b>Request Contract Details</b>	Returns information on the selected contract.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

## Bond Contract Details Page

Use the Bond Contract Details page to request contract-specific information for bonds, including the coupon, ratings, bond type, maturity date, and so on.

**Note:** Beginning with TWS Version 921, some bond contract data will be suppressed and will not be available from the API. All bond contract data will continue to be available from Trader Workstation, but only the following bond contract data will be available from the API:

- Contract ID
- Minimum Tick
- CUSIP (if you have subscribed to the CUSIP service)
- Rating (if you have subscribed to ratings)



## Requesting Bond Contract Details

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To request details for a bond contract

1. Click the **Bond Contract Details** tab at the bottom of the spreadsheet.
2. Enter the ticker symbol for which you want to request contract details.
3. Click the **Request Bond Contract Details** button on the toolbar.

**Note:** Beginning with TWS Version 921, some bond contract data will be suppressed and will not be available from the API. All bond contract data will continue to be available from Trader Workstation, but only the following bond contract data will be available from the API:

- Contract ID
- Minimum Tick
- CUSIP (if you have subscribed to the CUSIP service)
- Rating (if you have subscribed to ratings)

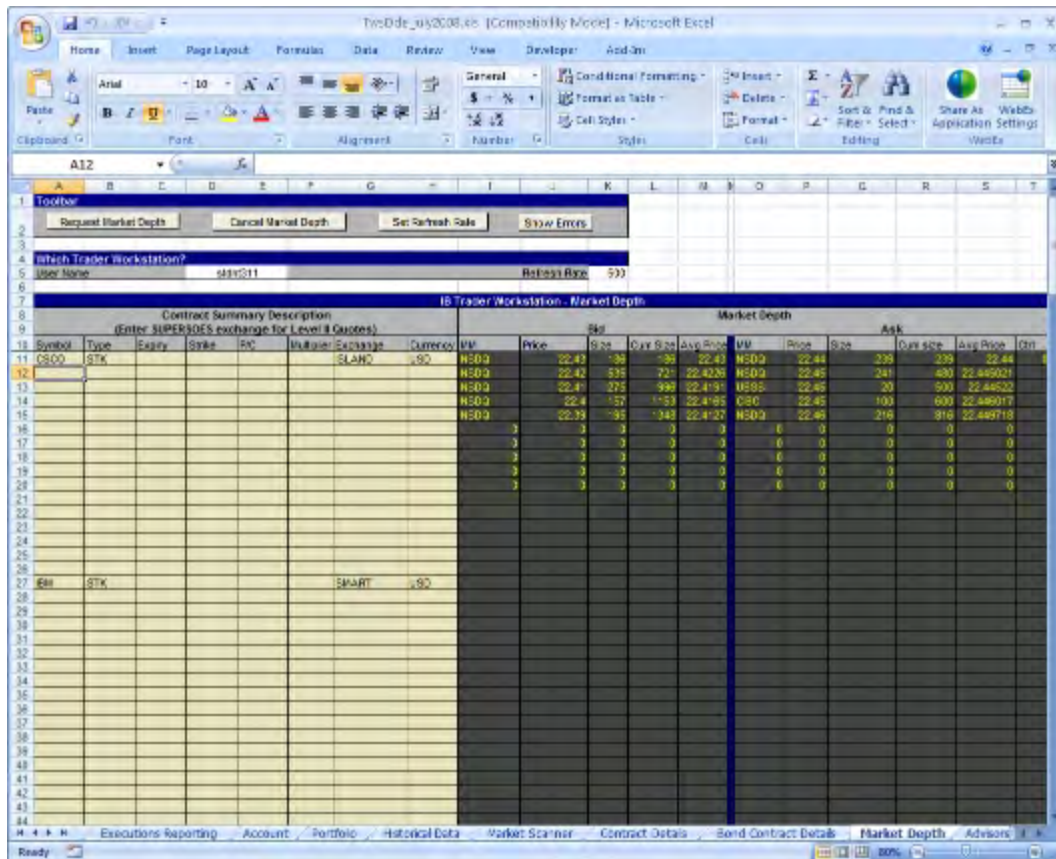
### Bond Contract Details Page Toolbar Buttons

The toolbar on the Bond Contract Details page includes the following buttons:

Button	Description
<b>Request Bond Contract Details</b>	Gets bond information data for the selected contract.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

### Market Depth Page

Use the Market Depth page to view market depth for selected contracts. You can also view market depth for NYSE-listed products through the Open Book Market Data Subscription, and NASDAQ-listed products through the TotalView Market Data Subscriptions, if you have signed up for those subscriptions.



## Using the Market Depth Page

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To request market depth for a contract

1. Click the **Market Depth** tab at the bottom of the spreadsheet to open the Market Depth page.
2. Select the ticker symbol for which you want to request the market depth, or enter a new ticker on a blank line.
3. Click the **Request Market Depth** button on the toolbar.

### To reset the market data refresh rate for tickers and market depth

1. Click the **Tickers** or **Market Depth** tab at the bottom of the spreadsheet.
2. Type the desired market data refresh rate in milliseconds in the *Refresh Rate* field in the *Which Trader Workstation?* area.
3. Move your cursor out of the *Refresh Rate* field.
4. Click the **Set Refresh Rate** button on the toolbar.

### To display more lines of market depth

You can edit the Request Market Depth macro to show more than the default 10 lines.

1. From the Market Depth page, press **Alt+F11**.

The Visual Basic Editor (VBE) opens and displays the code for the Market Depth page.

2. In the Declarations section at the top of the code window for the page, change the number value in **num-DisplayRows = 10** to a higher/lower value, then click the **Save** button on the VBE toolbar.
3. Close the Visual Basic Editor.

### Market Depth Page Toolbar Buttons

The toolbar on the Market Depth page includes the following buttons:

Button	Description
<b>Request Market Depth</b>	View bid/ask depth prices for the selected contract.
<b>Cancel Market Depth</b>	Cancel market depth for the selected contract.
<b>Set Refresh Rate</b>	Resets the refresh rate (in milliseconds) for market data.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.

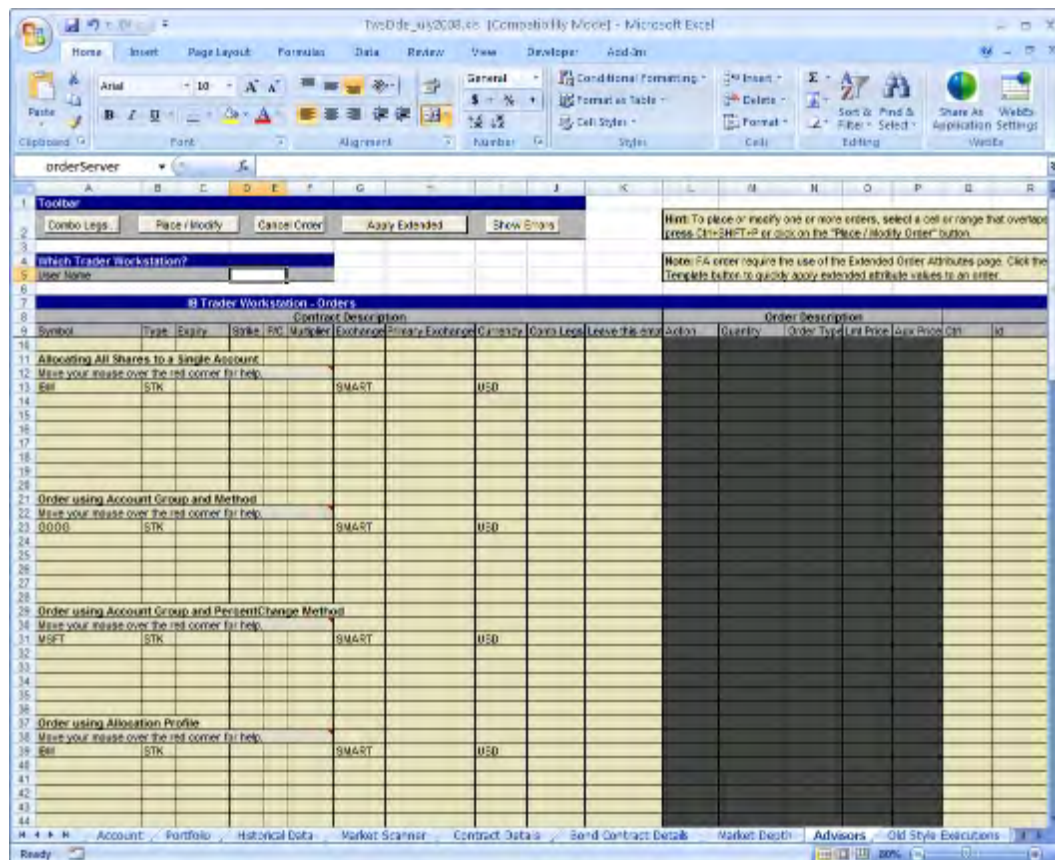
### Advisors Page

If you are a Financial Advisor and manage multiple accounts, use the Advisors page to create FA orders that:

- allocate shares to a single managed account
- use FA account groups and methods



- use allocation profiles



**Note:** You must set up your managed accounts, account groups, methods and allocation profiles in TWS before you can place FA orders in the DDE for Excel API sample spreadsheet.

### Allocating Shares to a Single Account

You can use the **Advisors** page to set up an order and allocate all shares in the order to a single account.

#### To allocate shares to a single account:

1. Create an account group in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.
4. Click the **Extended Order Attributes** tab. Enter the account code in the *Value* cell for the *Account (Institutional only)* extended order attribute.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the *Account* order attribute value to the order. The *Account* value is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
7. Click the **Place/Modify Order** button.

8. When you are done allocating shares to the account, delete the *Account* value from the **Extended Order Attributes** page. If you do not delete this value, it will be applied to all subsequent orders placed from the DDE for Excel API spreadsheet.

#### Placing an Order using an FA Account Group and Method

You can also use the **Advisors** page to set up an order using an FA account group and FA method.

#### To place an order using an FA account group and FA method:

1. Create the FA account group(s) and FA method(s) in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.
4. Click the **Extended Order Attributes** tab. Enter values for the following extended order attributes:
  - FA Group - Enter the name of the account group.
  - FA Method - Enter the name of the allocation method to use for this order.
  - FA Percentage - Enter the percentage used by the PctChange allocation method to use for this order. This attribute applies only to FA groups that use this method.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the extended order attribute values to the order. The values for *FA Group*, *FA Method* and *FA Percentage* are applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
7. Click the **Place/Modify Order** button.
8. When you are done allocating shares to the account, delete the values you entered on the **Extended Order Attributes** page. If you do not delete these values, they will be applied to all subsequent orders placed from the DDE for Excel API spreadsheet.

#### Placing an Order using an Allocation Profile

You can also use the **Advisors** page to set up an order using an FA allocation profile.

#### To place an order using an FA allocation profile:

1. Create the FA allocation profile in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.
4. Click the **Extended Order Attributes** tab. Enter the name of the allocation profile in the *Value* field for the *FA Profile* extended order attribute.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the extended order attribute value to the order. The value for *FA Profile* is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.

7. Click the **Place/Modify Order** button.
8. When you are done allocating shares to the account, delete the *FA Profile* value you entered on the **Extended Order Attributes** page. If you do not delete this value, it will be applied to all subsequent orders placed from the DDE for Excel API spreadsheet.

### Advisors Page Toolbar Buttons

The toolbar on the Basic Orders page includes the following buttons:

Button	Description
<b>Combo Legs</b>	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
<b>Place/Modify Orders</b>	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
<b>Cancel Order</b>	This button cancels the order(s) you have highlighted.
<b>Apply Extended</b>	Applies the current values on the Extended Order Attributes page to the highlighted order row.
<b>Show Errors</b>	Jumps to the Error Code field and shows the error code.



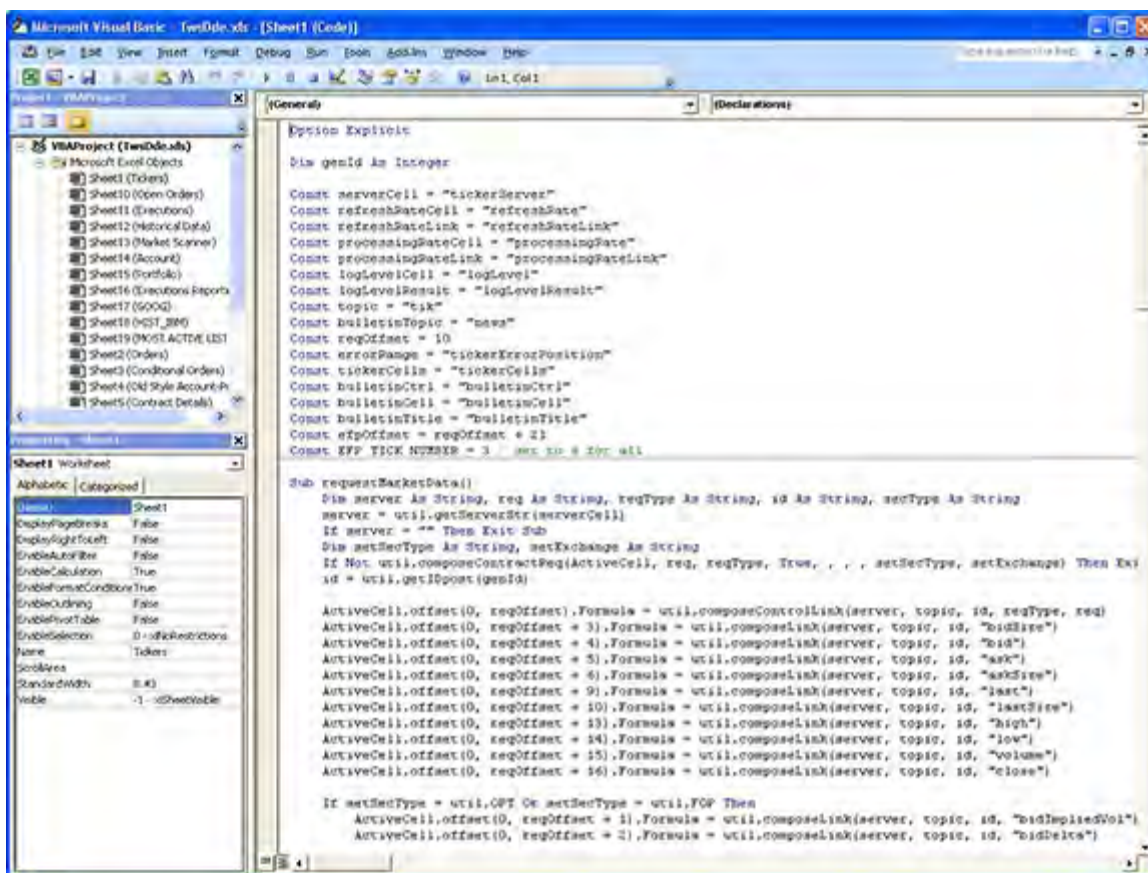
## DDE for Excel API Reference

This section provides a variety of reference information about the DDE for Excel API, including the following topics:

- [Viewing the Code](#)
- [Modules](#)
- [Named Ranges](#)
- [Macros](#)
- [DDE Syntax for Excel](#)

### Viewing the Code

To view the Visual Basic code behind the DDE for Excel API spreadsheet, press **Alt+F11** from any page. The Visual Basic Editor opens:



The Visual Basic Editor contains three main components:

- Project Explorer
- Properties Window
- Code Window

The Project Explorer contains a list of objects used in the spreadsheet. These objects correspond to the pages in the spreadsheet; to view the code for a particular page, double-click the page's corresponding object in the Project Explorer.

## Modules

The Visual Basic code includes the following modules (visible in the VBE Project Explorer):

- ArrayQueries
- ErrorDisplay
- Orders
- util

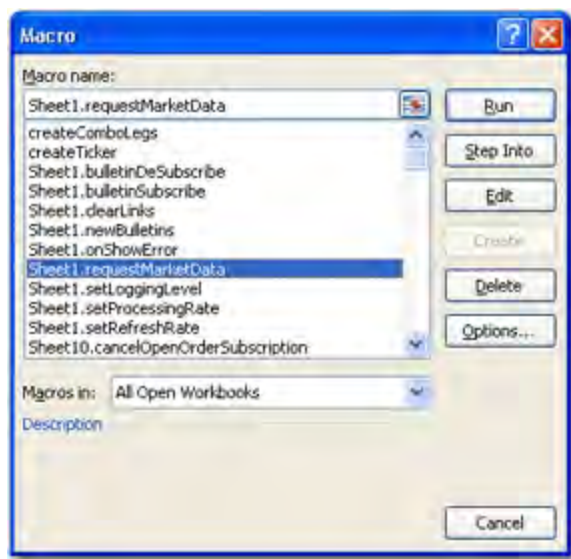
The util module contains many pre-defined constants that you can use when you program your own DDE API application. Using these constants instead of hard-coded values will make your application more robust and easier to maintain. Specifically, the following util functions are particularly useful in building new Visual Basic functionality:

- `composeLink` – put together a link that receives data, such as a market data bid size, option model volatility, or execution id.
- `composeControlLink` – put together a link that causes operations to occur, such as subscribing to market data, placing orders, or subscribing to the market scanner.
- `composeContractReq` – Read a contract description out of a page like Tickers or Orders, and build the DDE string representing it.

## Macros

The DDE API sample spreadsheet uses Microsoft Excel macros extensively. Each toolbar button on every page in the spreadsheet runs a macro when you click it. For example, when you click the Request Market Data button on the Tickers page, you are actually running a macro called `requestMarketData`.

You can see all the macros used in the sample spreadsheet by opening the Excel Macro dialog. From there you can choose to edit the macro, which opens macro code in the Visual Basic Editor.



**Note:** You must enable macros when you open Excel or none of the macros in the spreadsheet will function.

For information about recording, editing and viewing macros, refer to your Microsoft Excel documentation.

## Named Ranges

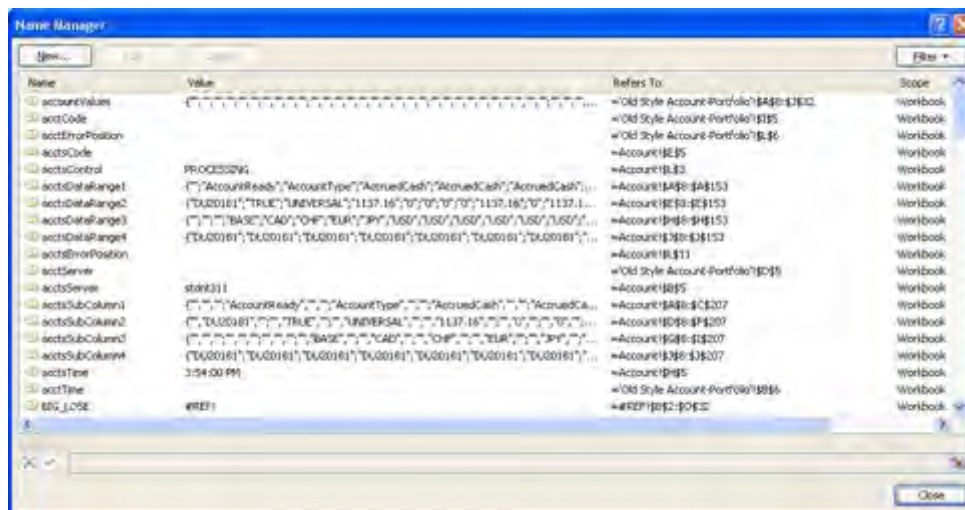
Named ranges are a Microsoft Excel feature that lets you assign meaningful names to a single cell or a range of cells in Microsoft Excel. The *TwsDde.xls* sample spreadsheet used named ranges extensively.

Named ranges help you to move data around on worksheets without breaking existing logic. It also makes important data available to your own custom macros and worksheets.

You can view all the named ranges used in the spreadsheet by doing the following, depending on which version of Excel you are using:

- In Excel 2007, you can see a complete list of all named ranges used in the spreadsheet by clicking **Formulas** then clicking **Name Manager**. The Name Manager displays every named range used in the spreadsheet, the value of the range, and the page and range of cells covered by the range.
- In earlier versions of Excel, you can view the named ranges by selecting **Name > Define** from the Tools menu. You can also download a free Name Manager from Microsoft that has additional functionality for these earlier versions of Excel.

The following screen shows the Name Manager for the DDE for Excel API sample spreadsheet:



## DDE Syntax for Excel

The table below defines possible cell values for DDE-supported functionality. The basic syntax, which appears in the Excel formula bar (the "=" enterable field at the top of the spreadsheet) when you put your cursor in a cell, is:

**=server|topic!id?reqType?field2**

or

**=server|error!error** (for an optional tag that will display errors)

where:

Description	Server	Topic	id	reqType	field2
Place an order	server	ord	idn	place	orderDescription
Modify an order	server	ord	idn	modify	orderModification
Cancel an order	server	ord	idn	cancel	
Check order status	server	ord	idn	status	
Request open orders	server	ord	idn	open	
Request executions	server	ord	idn	executed	
Check shares filled in order	server	ord	idn	sharesFilled	
Check shares remaining in order	server	ord	idn	sharesRemaining	
Execution (average) price	server	ord	idn	price	
Underlying	server	ord	idn	symbol	
Security type	server	ord	idn	secType	Refer to Note 6

Expiry	server	ord	idn	expiry	Refer to Note 7
Strike	server	ord	idn	strike	Refer to Note 8
Right	server	ord	idn	right	Refer to Note 8
Specify contract multiplier for options and futures	server	ord	idn	multiplier	Refer to Note 8
Order destination	server	ord	idn	exchange	
Currency	server	ord	idn	currency	
Order side	server	ord	idn	side	
Order quantity	server	ord	idn	size	
Order type	server	ord	idn	orderType	
Limit price	server	ord	idn	limitPrice	
Auxiliary price	server	ord	idn	auxPrice	
Local symbol	server	ord	idn	localSymbol	
Last fill price	server	ord	idn	lastFillPrice	
Create ticker	server	tik	idn	req	
Bid implied volatility	server	tik	idn	bidImpliedVol	
Bid delta	server	tik	idn	bidDelta	
Request bid size	server	tik	idn	bidSize	
Request bid price	server	tik	idn	bid	
Request ask price	server	tik	idn	ask	
Request ask size	server	tik	idn	askSize	
Ask implied volatility	server	tik	idn	askImpliedVol	
Ask delta	server	tik	idn	askDelta	
Request last price	server	tik	idn	last	
Request last size	server	tik	idn	lastSize	
Last implied volatility	server	tik	idn	lastImpliedVol	
Last delta	server	tik	idn	lastDelta	
Request today's high price	server	tik	idn	high	
Request today's low price	server	tik	idn	low	

Request today's volume size	server	tik	idn	volume	
Request last close price	server	tik	idn	close	
Request implied volatility calculated by the TWS option modeler	server	tik	idn	modelVolatility	
Request option delta calculated by the TWS option modeler	server	tik	idn	modelDelta	
Request the model price	server	tik	idn	modelPrice	
Request present value of dividends expected on the options underlier	server	tik	idn	pvDividend	
Request number of hold days until the expiry of the EFP	server	tik	idn	holdDays	
Request expiration date of the single stock future	server	tik	idn	futureExpiry	
Request dividends expected until the expiration of the single stock future	server	tik	idn	dividendsToExpiry	
Request annualized basis points	server	tik	idn	basisPoints	
Request annualized basis points in percentage form	server	tik	idn	formattedBasisPoints	
Request implied futures price	server	tik	idn	impliedFuture	
Request the dividend impact on the annualized basis points interest rate	server	tik	idn	dividendImpact	
Account statement control key	server	acct	idn	acctv	Account code (for Advisor-managed accounts only)
Request one account value string	server	acct	idn	key	
Account value	server	acct	idn	value	
Account currency	server	acct	idn	keyCurrency	

Account portfolio control key	server	acct	idn	acctp	Account code (for Advisor-managed accounts only)
Account portfolio underlying symbol	server	acct	idn	symbol	
Account portfolio security type	server	acct	idn	secType	
Account portfolio expiry	server	acct	idn	expiry	
Account portfolio strike price	server	acct	idn	strike	
Account portfolio right	server	acct	idn	right	
Account portfolio currency	server	acct	idn	currency	
Account portfolio local symbol	server	acct	idn	localSymbol	
Account portfolio market price	server	acct	idn	marketPrice	
Account portfolio market value	server	acct	idn	marketValue	
Account portfolio average cost	server	acct	idn	avgCost	
Account portfolio realized PNL	server	acct	idn	realizedPNL	
Account portfolio unrealized PNL	server	acct	idn	unrealizedPNL	
Request contract details	server	contract	idn	req	contractDescription
Valid order types	server	contract	idn	orderTypes	
Valid exchanges	server	contract	idn	validExchanges	
Contract identifier	server	contract	idn	conid	
Minimum tick	server	contract	idn	minTick	
Order multiplier	server	contract	idn	multiplier	
Market name	server	contract	idn	marketName	
Trading class	server	contract	idn	tradingClass	
Execution order id	server	exec	idn	orderId	
Underlying	server	exec	idn	symbol	
Security type	server	exec	idn	secType	

Expiry	server	exec	idn	expiry	
Strike	server	exec	idn	strike	
Right	server	exec	idn	right	
Order destination	server	exec	idn	exchange	
Currency	server	exec	idn	currency	
Local symbol	server	exec	idn	localSymbol	
Execution id	server	exec	idn	execId	
Execution time	server	exec	idn	time	
Account number	server	exec	idn	acctnNumber	
Exchange where executed	server	exec	idn	eExchange	
Side	server	exec	idn	side	
Number of shares filled in order	server	exec	idn	shares	
Execution (average) price	server	exec	idn	price	
Order ID	server	exec	idn	permId	
Identifies position as one to be liquidated last	server	exec	idn	liquidation	
Request execution details	server	exec	idn	Req	executionFilter
Request list of Advisor-managed accounts	server	FAaccts	idn	Req	
List of Advisor-managed accounts	server	FAaccts	idn	Value	
Request market depth	server	mktDepth	idn	req	contractDescripton? num_display_rows Refer to note (1) below.
Market maker	server	mktDepth	idn	mktMaker	rowId_side Refer to note (2) below.
Order price	server	mktDepth	idn	price	rowId_side Refer to note (2) below.



Order size	server	mktDepth	idn	size	rowId_side Refer to note (2) below.
Market data refresh rate	server	refreshRate	idn	millisec	Number of milliseconds
Subscribe to news bulletins	server	news	sub	0	Refer to note 3 below
News bulletin message ID	server	news	newsID		
News bulletin message type	server	news	newsType		Refer to note 4 below
News bulletin message text	server	news	msg		
Exchange from which news bulletins originated	server	news	exchange		
Set the server log level	server	logLevel	<log_level>		Refer to note 5 below.

Where:

orderDescription = side\_quantity\_symbol\_secType\_exp\_strike\_right\_exchange\_orderType\_lmtPrice\_auxPrice  
 {\_timeInForce\_ocaGroup\_account\_open/close\_origin\_orderRef\_transmit\_parentId\_blockOrder  
 \_sweepToFill\_displaySize\_triggerMethod\_ignoreRth\_hidden\_clientId\_accountCode\_goodAfterTime\_  
 goodTillDate\_faGroup\_faMethod\_faPercentage\_faProfile\_shortSaleSlot\_  
 PRIMARYEXCHANGE  
 \_shortSaleLocation\_ocaType\_rthOnly\_rule80A\_settlingFirm\_allOrNone\_minimumQty\_percentOffset\_  
 electronicTradeOnly\_firmQuoteOnly\_nbboPriceCap\_auctionStrategy\_startingPrice\_stockRefPrice\_  
 delta\_stockRangeLower\_stockRangeUpper\_volatility\_volatilityType\_referencePriceType\_hedgeDelta\_  
 continuousUpdate}

**Note:** Attributes in brackets are extended order attributes and are described in the topic [Extended Order Attributes](#).

- contractDescription - symbol\_secType\_exp\_strike\_right\_exchange
- tickerDescription = *symbol\_secType\_exp\_strike\_right\_exchange*
- ExecutionFilter = clientId\_accountCode\_date\_time\_symbol\_secType\_exchange\_side

Note 1: When requesting market depth you can specify the number of rows to display. If not supplied, the default number of rows is five (5) rows. This parameter can be used to optimize performance, as a low number of display rows requires less CPU overhead.

For example: =edemo|mktDepth!id0?req?MSFT\_STK\_SMART?10 will request 10 rows of market depth orders.

**Note 2:** Field 2 of the market depth 'price' and 'size' reqTypes contain additional information to specify the order row and side that the data applies to. Market depth orders are divided into two sides, BID and ASK, and order entries start

from the offset 0.

For example:

=demo|mktDepth!id0?price?0\_ASK will request the ASK price for the order entry 0.

=demo|mktDepth!id0?size?2\_BID will request the BID size for the order entry 2.

**Note 3:** When subscribing to news bulletins, the request should look like this:

=demo|news!sub?0

where the request type is a zero. To unsubscribe simply clear the subscription cell.

**Note 4:** The valid news bulletin types are:

- 1 = Regular news bulletin
- 2 = Exchange no longer available for trading
- 3 = Exchange available for trading

**Note 5:** The valid log levels are:

- 1 = SYSTEM (least detailed)
- 2 = ERROR (default, if no level is specified)
- 3 = WARNING
- 4 = INFORMATION
- 5 = DETAIL (most detailed)

**Note 6:** If the order is a combo, combo legs are inserted after the aux price. Here is a three-legged IBM combo description:

CMBLGS\_3\_8314\_100\_BUY\_SMART\_0\_36930759\_1\_BUY\_SMART\_0\_36930816\_1\_SELL\_SMART\_0\_CMBLGS.

Each leg contains : contract ID, number of contracts, side, exchange, and open/close (0 or 1). Retail can always specify 0, institutional need to specify a 1 if the order being executed would close a position.

**Note 7:** If the order is an option or a future, the expiry is inserted after the sec type.

**Note 8:** If the order is an option, the strike and right (put or call) are inserted after the expiry. If the multiplier is specified, it follows the strike and right.

## Active X

This chapter describes the ActiveX API, including the following topics:

- [Linking to the Application using ActiveX](#)
- [Registering Third-Party ActiveX Controls](#)
- [Running the ActiveX API on 64-bit Windows XP Systems](#)
- [Using the Visual Basic Sample Program](#)
- [ActiveX Methods](#)
- [ActiveX Events](#)
- [ActiveX COM Objects](#)
- [ActiveX Properties](#)
- [Placing a Combination Order](#)

The API software also includes an ActiveX for Excel sample spreadsheet, which duplicates most of the functionality of the DDE for Excel sample spreadsheet but is based on the ActiveX control, Tws.ocx. See [ActiveX for Excel](#) for details.

## Linking to the Application using ActiveX

Before you can use third-party ActiveX controls, you must [register them with Visual Basic](#).

### To link using the ActiveX control and VB, VBA or C++

1. Drop the application control onto a form or dialog box.
2. Call the following methods:
  - Call the connect() method to connect to the running application.
  - Call the methods you need to perform whatever operations you require, such as the reqMktData() method to request market data.
3. Call the placeOrder() method to place an order. Orders using extended attributes require that ActiveX properties representing them be set first.
4. Handle the following events:
  - Handle the nextValidId() event to receive the next available valid order ID. Increment the ID by one for successive orders.
  - Handle the tickPrice() and tickSize() events to receive the market data.
  - Handle the orderStatus() event to receive status information about orders.
  - Handle the error() event to receive error information.
  - Handle the connectionClosed() event to be notified in case the application stops communicating with the ActiveX control.

## Registering Third-Party ActiveX Controls

To use a third-party ActiveX control in Visual Basic it must be registered first.

**To register the ActiveX control with Visual Basic, follow these instructions:**

1. From the **Components** menu in your VB project, select the TWS ActiveX control (Tws.ocx).
2. Click **Apply**.
3. Verify that the TWS control appears in the toolbar with all standard controls.

## Running the ActiveX API on 64-bit Windows XP Systems

To run the ActiveX API on 64-bit Windows XP systems, do the following:

1. Install Microsoft Visual C++ 2005 SP1 Redistributable Package (x86).
2. Install Microsoft Visual J# 2.0 Redistributable Package.
3. Download and install the API software.

## Using the Visual Basic Sample Program

You can access the server through the ActiveX interface using the Visual Basic sample application. To run the sample you must:

- Install the API sample programs
- Configure the application to support the API components
- Have MS Visual Studio (Visual Basic 6.0 or higher) installed on your PC.

The Visual Basic program is a sample program that demonstrates use of the TWS ActiveX control to connect to the server from a Visual Basic application.

### To open the VB\_API\_sample program:

1. From the MS Visual Basic file menu, select **Open Project**.
2. Navigate to the \TestActiveXClient\_VB directory in your API installation folder, and select *VB\_API\_sample.vbproj*. X\_XX represents the API version number; for example, 9.60.

If you are using Visual Studio 2008, you will have to upgrade your project before it can be opened. Visual Basic presents an Upgrade Wizard to walk you through this simple process.

3. On the **Projects** menu, select *Components*.
4. In the Components dialog box, select the *TWS ActiveXControl* module and click **OK**.
5. Press **Ctrl+F5** to compile and run the project.

**Note:** If you have trouble getting the sample program to run, try re-registering the TWS ActiveX component, Tws.ocx.

## ActiveX Methods

ActiveX methods allow your application to call functions and request information from TWS.

<p><b>Connection and Server</b></p> <p><a href="#"><u>connect()</u></a>  <a href="#"><u>disconnect()</u></a>  <a href="#"><u>reqCurrentTime()</u></a>  <a href="#"><u>setServerLogLevel()</u></a></p> <p><b>Market Data</b></p> <p><a href="#"><u>reqMktDataEx()</u></a>  <a href="#"><u>cancelMktData()</u></a>  <a href="#"><u>calculateImpliedVolatility()</u></a>  <a href="#"><u>cancelCalculateImpliedVolatility()</u></a>  <a href="#"><u>calculateOptionPrice()</u></a>  <a href="#"><u>cancelCalculateOptionPrice()</u></a>  <a href="#"><u>reqMarketDataType()</u></a></p> <p><b>Orders</b></p> <p><a href="#"><u>placeOrderEx()</u></a>  <a href="#"><u>cancelOrder()</u></a>  <a href="#"><u>reqOpenOrders()</u></a>  <a href="#"><u>reqAllOpenOrders()</u></a>  <a href="#"><u>reqAutoOpenOrders()</u></a>  <a href="#"><u>reqIds()</u></a>  <a href="#"><u>exerciseOptionsEx()</u></a></p> <p><b>Executions</b></p> <p><a href="#"><u>reqExecutionsEx()</u></a></p> <p><b>Contract Details</b></p> <p><a href="#"><u>reqContractDetailsEx()</u></a></p> <p><b>Market Depth</b></p> <p><a href="#"><u>reqMktDepthEx()</u></a>  <a href="#"><u>cancelMktDepth()</u></a></p> <p><b>Account</b></p> <p><a href="#"><u>reqAccountUpdates()</u></a></p>	<p><b>News Bulletins</b></p> <p><a href="#"><u>reqNewsBulletins()</u></a>  <a href="#"><u>cancelNewsBulletins()</u></a></p> <p><b>Financial Advisors</b></p> <p><a href="#"><u>reqManagedAccts()</u></a>  <a href="#"><u>requestFA()</u></a>  <a href="#"><u>replaceFA()</u></a>  <a href="#"><u>reqAccountSummary()</u></a>  <a href="#"><u>cancelAccountSummary()</u></a>  <a href="#"><u>reqPositions()</u></a>  <a href="#"><u>cancelPositions()</u></a></p> <p><b>Historical Data</b></p> <p><a href="#"><u>reqHistoricalDataEx()</u></a>  <a href="#"><u>cancelHistoricalData()</u></a></p> <p><b>Market Scanners</b></p> <p><a href="#"><u>reqScannerParameters()</u></a>  <a href="#"><u>reqScannerSubscriptionEx()</u></a>  <a href="#"><u>cancelScannerSubscription()</u></a></p> <p><b>Real Time Bars</b></p> <p><a href="#"><u>reqRealTimeBarsEx()</u></a>  <a href="#"><u>cancelRealTimeBars()</u></a></p> <p><b>Factory Methods</b></p> <p><a href="#"><u>createComboLegList()</u></a>  <a href="#"><u>createContract()</u></a>  <a href="#"><u>createExecutionFilter()</u></a>  <a href="#"><u>createOrder()</u></a>  <a href="#"><u>createScannerSubscription()</u></a>  <a href="#"><u>createTagValueList()</u></a>  <a href="#"><u>createUnderComp()</u></a></p> <p><b>Fundamental Data</b></p> <p><a href="#"><u>reqFundamentalData()</u></a>  <a href="#"><u>cancelFundamentalData()</u></a></p>
---	--



**connect()**

Call this method to connect to the host application.

**Public Overridable Sub connect(ByVal host as String, ByVal port as Integer, ByVal clientID as Integer)**

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

**disconnect()**

Call this method to terminate the connections the host application. Calling this method does not cancel orders that have already been sent.

**Public Overridable Sub disconnect()**

**reqCurrentTime()**

Returns the current system time on the server side.

**Public Overridable Sub reqCurrentTime()**

**setServerLogLevel()**

**Public Overridable Sub setServerLogLevel(ByVal logLevel As Integer)**

Parameter	Description
logLevel	Specifies the level of log entry detail used by the server when processing API requests. Valid values include: <ul style="list-style-type: none"> <li>• 1 = SYSTEM</li> <li>• 2 = ERROR</li> <li>• 3 = WARNING</li> <li>• 4 = INFORMATION</li> <li>• 5 = DETAIL</li> </ul>

The default level is ERROR. See [API Logging](#) for more details.

**reqMktDataEx()**

Call this method to request market data. The market data will be returned by the [tickPrice\(\)](#), [tickSize\(\)](#), [tickOptionComputation\(\)](#), [tickGeneric\(\)](#), [tickString\(\)](#) and [tickEFP\(\)](#) events in `disinterface_DTwsEvents`.

**Public Overridable Sub reqMktDataEx(ByVal tickerId As Integer, ByVal contract As TWSLib.IContract, ByVal genericTicks As String, ByVal snapshot As Integer)**

Parameter	Description
tickerId	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	This object contains a description of the contract for which market data is being requested.
genericTicks	A comma delimited list of generic tick types. For more information about tick types, see <a href="#">Generic Tick Types</a> .
snapshot	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.

**cancelMktData()**

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

**Public Overridable Sub cancelMktData(ByVal id As Integer)**

Parameter	Description
id	The ID that was specified in the call to reqMktData().

**calculateImpliedVolatility()**

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

**Public Overridable Sub calculateImpliedVolatility(ByVal reqId As Integer, ByVal contract As TWSLib.IContract, ByVal optionPrice As Double, ByVal underPrice As Double)**

Parameter	Description
reqId	The ticker ID.
contract	Describes the contract.
optionPrice	The price of the option.
underPrice	Price of the underlying.

**cancelCalculateImpliedVolatility()**

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

**Public Overridable Sub calculateImpliedVolatility(ByVal reqId As Integer)**

Parameter	Description
reqId	The ticker id.

**calculateOptionPrice()**

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

**Public Overridable Sub calculateOptionPrice(ByVal reqId As Integer, ByVal contract As TWSLib.IContract, ByVal volatility As Double, ByVal underPrice As Double)**

Parameter	Description
reqId	The ticker ID.
contract	Describes the contract.
volatility	The volatility.
underPrice	Price of the underlying.

**cancelCalculateOptionPrice()**

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

**Public Overridable Sub calculateOptionPrice(ByVal reqId As Integer)**

Parameter	Description
reqId	The ticker id.

**reqMarketDataType()**

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.

**Public Overridable Sub reqMarketDataType(type As Integer)**

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

**placeOrderEx()**

Call this method to place an order. The order status will be returned by the [orderStatus\(\)](#) event in `dispinterface_DTwsEvents`.

**Public Overridable Sub placeOrderEx(ByVal orderId As Integer, ByVal contract As TWSLib.IContract, ByVal order As TWSLib.IOrder)**

Parameter	Description
orderId	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	This object contains attributes used to describe the contract.
order	This object contains the details of the order. Note: Each client MUST connect with a unique clientId.

### **cancelOrder()**

Call this method to cancel an order.

**Public Overridable Sub cancelOrder(ByVal id As Integer)**

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

### **reqOpenOrders()**

Call this method to request the open orders that were placed from this client. Each open order will be fed back through the [openOrderEx\(\)](#) events.

**Note:** The client with a clientId of 0 will also receive the application-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 application sessions.

**Public Overridable Sub reqOpenOrders()**

### **reqAllOpenOrders()**

Call this method to request the open orders that were placed from all clients and also from the application. Each open order will be fed back through the [orderStatus\(\)](#) event.

**Note:** No association is made between the returned orders and the requesting client

**Public Overridable Sub reqAllOpenOrders()**

### **reqAutoOpenOrders()**

Call this method to request that newly created application orders be implicitly associated with the client. When a new application order is created, the order will be associated with the client, and fed back through the [orderStatus\(\)](#) event.

**Note:** This request can only be made from a client with a clientId of 0.

**Public Overridable Sub reqAutoOpenOrders(ByVal bAutoBind As Integer)**

Parameter	Description
bAutoBind	If set to TRUE, newly created application orders will be implicitly associated with the client. If set to FALSE, no association will be made.

### reqIds()

Call this function to request the next valid ID that can be used when placing an order. After calling this method, the [next-ValidId\(\)](#) 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 Overridable Sub reqIds(ByVal numIds As Integer)

Parameter	Description
numIds	Set to 1.

### exerciseOptionsEx()

Call this method to exercise options.

**Note:** SMART is not an allowed exchange in [exerciseOptionsEx\(\)](#) calls, and that TWS does a request for the position in question whenever any API initiated exercise or lapse is attempted.

#### Public Overridable Sub exerciseOptionsEx(ByVal tickerId As Integer, ByVal contract As TWSLib.IContract, ByVal exerciseAction As Integer, ByVal exerciseQuantity As Integer, ByVal account As String, ByVal override As Integer)

Parameter	Description
tickerId	The Id for the exercise request
contract	This structure contains a description of the contract for which market data is being requested.
exerciseAction	This can have two values:  1 = exercise  2 = lapse
exerciseQuantity	The number of contracts to be exercised
account	For institutional orders. Specifies the IB account.
override	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

## reqExecutionsEx()

When this method is called, the execution reports that meet the filter criteria are downloaded to the client via the `execDetailsEx()` event in `dispinterface_DTwsEvents`. 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.

**Public Overridable Sub reqExecutionsEx(ByVal reqId As Integer, ByVal filter As TWSLib.IExecutionFilter)**

Parameter	Description
filter	The filter criteria used to determine which execution reports are returned.

## reqContractDetailsEx()

Call this method to download all details for a particular contract. The contract details will be received via the [contractDetailsEx\(\)](#) callback in `dispinterface_DTwsEvents`.

**Public Overridable Sub reqContractDetailsEx(ByVal reqId As Integer, ByVal contract As TWSLib.IContract)**

Parameter	Description
reqId	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contract	This object contains a description of the contract for which market data is being requested.

## reqMktDepthEx()

Call this method to request market depth for a specific contract. The market depth will be returned by the [updateMktDepth\(\)](#) and [updateMktDepthL2\(\)](#) events.

**Public Overridable Sub reqMktDepthEx(ByVal tickerId As Integer, ByVal contract As TWSLib.IContract, ByVal numRows As Integer)**

Parameter	Description
tickerId	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	This object contains a description of the contract for which market data is being requested.
numRows	Specifies the number of market depth rows to return.

## cancelMktDepth()

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

**Public Overridable Sub cancelMktDepth(ByVal id As Integer)**

Parameter	Description
id	The ID that was specified in the call to reqMktDepth() or reqMktDepth2().

### reqAccountUpdates()

Call this method to request account updates. The account data will be fed back through the [updateAccountTime\(\)](#), [updateAccountValue\(\)](#) and [updatePortfolioEx\(\)](#) events.

**Public Overridable Sub reqAccountUpdates(ByVal subscribe As Integer, ByVal acctCode As String)**

Parameter	Description
subscribe	If set to 1, the client will start receiving account and portfolio updates. If set to 0, the client will stop receiving this information.
acctCode	The account code for which to receive account and portfolio updates.

### reqNewsBulletins()

Call this method to start receiving news bulletins. Each bulletin will be returned by the [updateNewsBulletin\(\)](#) event.

**Public Overridable Sub reqNewsBulletins(ByVal allDaysMsgs As Integer)**

Parameter	Description
allDaysMsgs	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.

### cancelNewsBulletins()

Call this method to stop receiving news bulletins.

**Public Overridable Sub cancelNewsBulletins()**

### reqManagedAccts()

Call this method to request the list of managed accounts. The list will be returned by the [managedAccounts\(\)](#) event.

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

**Public Overridable Sub reqManagedAccts()**

### requestFA()

Call this method to request FA configuration information from the server. The data returns in an XML string via the [receiveFA\(\)](#) event.

**Public Overridable Sub requestFA(ByVal faDataType As Integer)**

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being requested. Valid values include: <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 =ACCOUNT ALIASES</li> </ul>

### replaceFA()

Call this method to modify FA configuration information from the API. Note that this can also be done manually.

**Public Overridable Sub replaceFA(ByVal faDataType As Integer, ByVal cxml As String)**

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being modified via the API. Valid values include:
cxml	The XML string containing the new FA configuration information.

### 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 an FA managed account.

**Public Overridable Sub reqAccountSummary(ByVal messageType As Integer, ByVal version As Integer, ByVal reqId As Integer, ByVal groupName As String, tags As String)**

Parameter	Type	Description
messageType	Integer	Set this to 62.
version	Integer	Set this to 1.
reqId	Integer	
groupName	String	Set to <i>All</i> 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.



Parameter	Type	Description
tags	String	<p>A comma-separated list of account tags. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>NetLiquidation</i></li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i></li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i></li> <li>• <i>RegTMargin</i></li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i></li> <li>• <i>MaintMarginReq</i></li> <li>• <i>AvailableFunds</i></li> <li>• <i>ExcessLiquidity</i></li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i></li> <li>• <i>FullMaintMarginReq</i></li> <li>• <i>FullAvailableFunds</i></li> <li>• <i>FullExcessLiquidity</i></li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i></li> <li>• <i>LookAheadMaintMarginReq</i></li> <li>• <i>LookAheadAvailableFunds</i></li> <li>• <i>LookAheadExcessLiquidity</i></li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> </ul>

Parameter	Type	Description
		<ul style="list-style-type: none"> <li><i>DayTradesRemaining</i> — 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.</li> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>

### cancelAccountSummary()

Cancels the request for Account Window Summary tab data.

**Note:** This request can only be made when connected to an FA managed account.

**Public Overridable Sub** cancelAccountSummary(ByVal messageId As Integer, ByVal version As Integer, ByVal 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.

### reqPositions()

Requests real-time position data for all accounts.

**Note:** This request can only be made when connected to an FA managed account.

**Public Overridable Sub** reqPositions(ByVal messageId As Integer, ByVal version As Integer)

Parameter	Type	Description
messageId	Integer	Set this to 62
version	Integer	Set this to 1.

### cancelPositions()

Cancels real-time position updates.

**Note:** This request can only be made when connected to an FA managed account.

**Public Overridable Sub** cancelPositions(ByVal messageId As Integer, ByVal version As Integer)

Parameter	Type	Description
messageId	Integer	Set this to 64.
version	Integer	Set this to 1.

### reqHistoricalDataEx()

Call this method to start receiving historical data results through the historicalData() event.

**Public Overridable Sub reqHistoricalDataEx(ByVal tickerId As Integer, ByVal contract As TWSLib.IContract, ByVal endDateTime As String, ByVal duration As String, ByVal barSize As String, ByVal whatToShow As String, ByVal useRTH As Integer, ByVal formatDate As Integer)**

Parameter	Description
tickerId	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	This structure contains a description of the contract for which market data is being requested.
endDateTime	Use the format yyyyymmdd hh:mm:ss tmz, where the time zone is allowed (optionally) after a space at the end.
durationStr	<p>This is the time span the request will cover, and is specified using the format: &lt;integer&gt; &lt;unit&gt;, i.e., 1 D, where valid units are:</p> <ul style="list-style-type: none"> <li>• S (seconds)</li> <li>• D (days)</li> <li>• W (weeks)</li> <li>• M (months)</li> <li>• Y (years)</li> </ul> <p><b>Note:</b> If no unit is specified, seconds are used. Also, note "years" is currently limited to one.</p>
barSize	<p>The size of the bars that will be returned (within IB/TWS limits). Valid values include:</p> <p><b>Bar Size</b></p> <ul style="list-style-type: none"> <li>• 1 sec</li> <li>• 5 secs</li> <li>• 15 secs</li> <li>• 30 secs</li> <li>• 1 min</li> <li>• 2 mins</li> <li>• 3 mins</li> <li>• 5 mins</li> <li>• 15 mins</li> <li>• 30 mins</li> <li>• 1 hour</li> <li>• 1 day</li> </ul>

whatToShow	<p>Determines the nature of data being extracted. Valid values include:</p> <ul style="list-style-type: none"> <li>• TRADES</li> <li>• MIDPOINT</li> <li>• BID</li> <li>• ASK</li> <li>• BID_ASK</li> <li>• HISTORICAL_VOLATILITY</li> <li>• OPTION_IMPLIED_VOLATILITY</li> </ul>
useRTH	<p>Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 - all data is returned even where the market in question was outside of its regular trading hours.</li> <li>• 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.</li> </ul>
formatDate	<p>Determines the date format applied to returned bars. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 - dates applying to bars returned in the format: yyymmdd {space} {space} hh:mm:dd</li> <li>• 2 - dates are returned as a long integer specifying the number of seconds since 1/1/1970 GMT.</li> </ul>

**Note:** For information about historical data request limitations, see [Historical Data Limitations](#).

### cancelHistoricalData()

Used if an internet disconnect has occurred or the results of a query are otherwise delayed and the application is no longer interested in receiving the data.

**Public Overridable Sub** cancelHistoricalData(ByVal tickerId As Integer)

Parameter	Description
tickerId	The ticker ID. Must be a unique value.

### reqScannerParameters()

Requests an XML string that describes all possible scanner queries.

**Public Overridable Sub** reqScannerParameters()

### reqScannerSubscriptionEx()

Call the reqScannerSubscriptionEX() method to start receiving market scanner results through the [scannerDataEx\(\)](#) event.

**Public Overridable Sub reqScannerSubscriptionEx(ByVal tickerId As Integer, ByVal subscription As TWSLib.I-ScannerSubscription)**

Parameter	Description
tickerId	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	Summary of the scanner subscription parameters including filters.

**cancelScannerSubscription()**

**Public Overridable Sub cancelScannerSubscription(ByVal tickerId As Integer)**

Parameter	Description
tickerId	The ticker ID. Must be a unique value.

**reqRealTimeBarsEx()**

Call the reqRealTimeBarsEx() method to start receiving real time bar results through the [realtimeBar\(\)](#) event.

**Public Overridable Sub reqRealTimeBarsEx(ByVal tickerId As Integer, ByVal contract As TWSLib.IContract, ByVal barSize As Integer, ByVal whatToShow As String, ByVal useRTH As Integer)**

Parameter	Description
tickerId	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	This structure contains a description of the contract for which market data is being requested.
barSize	Currently only 5 second bars are supported, if any other value is used, an exception will be thrown.
whatToShow	Determines the nature of the data extracted. Valid values include: <ul style="list-style-type: none"> <li>• TRADES</li> <li>• BID</li> <li>• ASK</li> <li>• MIDPOINT</li> </ul>

useRTH	<p>Regular Trading Hours only. Valid values include:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 1 = only data within the regular trading hours for the product requested is returned, even if the time span falls partially or completely outside.</li> </ul>
--------	--

### cancelRealTimeBars()

Used if an Internet disconnect has occurred or the results of a query are otherwise delayed and the application is no longer interested in receiving the data.

**Public Overridable Sub cancelRealTimeBars(ByVal tickerId As Integer)**

Parameter	Description
tickerId	The ticker ID. Must be a unique value.

### createComboLegList()

This factory method is used to create an [IComboLegList](#) COM object.

**Public Overridable Sub createComboLegList() As TWSLib.IComboLegList**

You must use the factory “create” methods to create the COM objects in this section. For example, the create-ComboLegList() method creates an IComboLeg object. The IComboLeg object contains the definition of the leg list.

### createContract()

This factory method is used to create an [IContract](#) COM object.

**Public Overridable Sub createContract() As TWSLib.IContract**

You must use the factory “create” methods to create the COM objects described in this chapter. The createContract() method creates an IContract object, which contains a description of the contract for which market data is being requested.

### createExecutionFilter()

This factory method is used to create an [IExecutionFilter](#) COM object.

**Public Overridable Sub createExecutionFilter() As TWSLib.IExecutionFilter**

You must use the factory “create” methods to create the COM objects described in this chapter. The createExecutionFilter() method creates an IExecutionFilter object, which contains the filter criteria used to determine which execution reports are returned.

### createOrder()

This factory method is used to create an [IOrder](#) COM object.

**Public Overridable Sub createOrder() As TWSLib.IOrder**

You must use the factory “create” methods to create the COM objects described in this chapter. The createOrder() method creates an IOrder object, which contains the details of an order.

### createScannerSubscription()

This factory method is used to create an [IScannerSubscription](#) COM object.

**Public Overridable Sub createScannerSubscription() As TWSLib.IScannerSubscription**

You must use the factory “create” methods to create the COM objects described in this chapter. The createScannerSubscription() method creates an IScannerSubscription object, which contains a summary of the scanner subscription parameters.

### createTagValueList

This factory method is used to create [ITagValueList](#) and [ITagValue](#) objects.

**Public Overridable Function createTagValueList() As TWSLib.ITagValueList**

You must use the factory “create” methods to create the COM objects described in this chapter.

### createUnderComp()

This factory method is used to create an [IUnderComp](#) COM object.

**Public Overridable Sub createUnderComp() As TWSLib.IScannerSubscription**

You must use the factory “create” methods to create the COM objects described in this chapter. The createUnderComp() method creates an IUnderComp object, which is used to define a Delta-Neutral Combo contract.

### reqFundamentalData()

Call this method to receive Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.

**Public Overridable Sub reqFundamentalData(ByVal reqId As Integer, ByVal contract As TWSLib.IContract, ByVal reportType As String)**

Parameter	Description
reqId	The ID of the data request.
contract	This structure contains a description of the contract for which Reuters Fundamental data is being requested.
reportType	Identifies the report type, which is one of the following: <ul style="list-style-type: none"> <li>• Estimates</li> <li>• Financial Statements</li> <li>• Summary</li> </ul>

**cancelFundamentalData()**

Call this method to stop receiving Reuters global fundamental data.

**Public Overridable Sub cancelFundamentalData(ByVal reqId As Integer)**

Parameter	Description
reqId	The ID of the data request.



## ActiveX Events

ActiveX events receive information from the system and make it available to an application. This section defines the ActiveX events you can receive via the DTwsEvents interface.

<b>Connection and Server</b> <a href="#"><u>connectionClosed()</u></a> <a href="#"><u>currentTime()</u></a> <a href="#"><u>errMsg()</u></a>  <b>Market Data</b> <a href="#"><u>tickPrice()</u></a> <a href="#"><u>tickSize()</u></a> <a href="#"><u>tickOptionComputation()</u></a> <a href="#"><u>tickGeneric()</u></a> <a href="#"><u>tickString()</u></a> <a href="#"><u>tickEFP()</u></a> <a href="#"><u>tickSnapshotEnd()</u></a> <a href="#"><u>marketDataType()</u></a>  <b>Orders</b> <a href="#"><u>orderStatus()</u></a> <a href="#"><u>openOrderEx()</u></a> <a href="#"><u>nextValidId()</u></a> <a href="#"><u>permId()</u></a>  <b>Account and Portfolio</b> <a href="#"><u>updateAccountValue()</u></a> <a href="#"><u>updatePortfolioEx()</u></a> <a href="#"><u>updateAccountTime()</u></a>  <b>News Bulletins</b> <a href="#"><u>updateNewsBulletin()</u></a>	<b>Contract Details</b> <a href="#"><u>contractDetailsEx()</u></a> <a href="#"><u>contractDetailsEnd()</u></a> <a href="#"><u>bondContractDetails()</u></a>  <b>Executions</b> <a href="#"><u>execDetailsEx()</u></a> <a href="#"><u>execDetailsEnd()</u></a> <a href="#"><u>commissionReport()</u></a>  <b>Market Depth</b> <a href="#"><u>updateMktDepth()</u></a> <a href="#"><u>updateMktDepthL2()</u></a>  <b>Financial Advisors</b> <a href="#"><u>managedAccounts()</u></a> <a href="#"><u>receiveFA()</u></a> <a href="#"><u>accountSummary()</u></a> <a href="#"><u>accountSummaryEnd()</u></a> <a href="#"><u>position()</u></a> <a href="#"><u>positionEnd()</u></a>  <b>Historical Data</b> <a href="#"><u>historicalData()</u></a>  <b>Market Scanners</b> <a href="#"><u>scannerParameters()</u></a> <a href="#"><u>scannerDataEx()</u></a> <a href="#"><u>scannerDataEnd()</u></a>  <b>Real Time Bars</b> <a href="#"><u>realtimebar()</u></a>  <b>Fundamental Data</b> <a href="#"><u>fundamentalData()</u></a>
--	---

**connectionClosed()**

This event is triggered when TWS closes the sockets connection with the ActiveX control, or when TWS is shut down.

**Sub connectionClosed()**

**currentTime()**

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

**Sub currentTime(ByVal time As Integer)**

Parameter	Description
time	The current system time on the server side.,

**errMsg()**

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

**Sub errMsg(ByVal id As Integer, ByVal errorCode As Integer, ByVal errorMsg As String)**

Parameter	Description
id	This is the orderId or tickerId of the request that generated the error
errorCode	Error codes are documented in the Error Codes topic.
errorString	This is the textual description of the error, also documented in the Error Codes topic.

**tickPrice()**

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

**Sub tickPrice(ByVal id As Integer, ByVal tickType As Integer, ByVal price As Double, ByVal canAutoExecute As Integer)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktData()

tickType	Specifies the type of price. Possible values are: <ul style="list-style-type: none"> <li>• 1 = bid</li> <li>• 2 = ask</li> <li>• 4 = last</li> <li>• 6 = high</li> <li>• 7 = low</li> <li>• 9 = close</li> </ul>
price	The bid, ask or last price, the daily high, daily low or last day close, depending on tickType value.
canAutoExecute	Specifies whether the price tick is available for automatic execution. Possible values are: <ul style="list-style-type: none"> <li>• 0 = not eligible for automatic execution</li> <li>• 1 = eligible for automatic execution</li> </ul>

### tickSize()

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

**Sub tickSize(ByVal id As Integer, ByVal tickType As Integer, ByVal size As Integer)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktData()
tickType	Specifies the type of price. Possible values are:
size	The bid size, ask size, last size or trading volume, depending on the tickType value.

### tickOptionComputation()

**Sub tickOptionComputation(ByVal id As Integer, ByVal tickType As Integer, ByVal impliedVol As Double, ByVal delta As Double, ByVal optPrice As Double, ByVal pvDividend As Double, ByVal gamma As Double, ByVal vega As Double, ByVal theta As Double, ByVal undPrice As Double)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktData()

tickType	Specifies the type of tick. Possible values are: <ul style="list-style-type: none"> <li>• 10 = Bid</li> <li>• 11 = Ask</li> <li>• 12 = Last</li> </ul>
ImpliedVol	The implied volatility calculated by the TWS option modeler, using the specified tick-type value.
delta	The option delta calculated by the TWS option modeler.
optPrice	The option price.
pvDividend	The present value of dividends expected on the options underlier.
gamma	The option gamma value.
vega	The option vega value.
theta	The option theta value.
undPrice	The price of the underlying.

### tickGeneric()

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

**Sub tickGeneric(ByVal id As Integer, ByVal tickType As Integer, ByVal value As Double)**

Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
tickType	Specifies the type of tick. 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	The value of the specified field.

### tickString()

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

**Sub tickString(ByVal id As Integer, ByVal tickType As Integer, ByVal value As String)**

Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()

tickType	Specifies the type of tick. Pass the field value into TickType.getField(int tickType) to retrieve the field description. For example, a field value of 45 will map to lastTimestamp, etc.
value	The value of the specified field.

**tickEFP()**

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

**Sub tickEFP(ByVal tickerId As Integer, ByVal field As Integer, ByVal basisPoints As Double, ByVal formattedBasisPoints As String, ByVal totalDividends As Double, ByVal holdDays As Integer, ByVal futureExpiry As String, ByVal dividendImpact As Double, ByVal dividendsToExpiry As Double)**

Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData().
field	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	Annualized basis points, which is representative of the financing rate that can be directly compared to broker rates.
formattedBasisPoints	Annualized basis points as a formatted string that depicts them in percentage form.
totalDividends	The total expected dividends.
holdDays	The number of hold days until the expiry of the EFP.
futureExpiry	The expiration date of the single stock future.
dividendImpact	The dividend impact upon the annualized basis points interest rate.
dividendsToExpiry	The dividends expected until the expiration of the single stock future.

**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.

**Sub tickSnapshotEnd(ByVal reqId As Integer)**

Parameter	Description
reqId	Id of the data request.

## 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.

**Sub marketDataType(ByVal reqId As Integer, type As Integer)**

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

## orderStatus()

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

**Sub orderStatus(ByVal id As Integer, ByVal status As String, ByVal filled As Integer, ByVal remaining As Integer, ByVal avgFillPrice As Double, ByVal permId As Integer, ByVal parentId As Integer, ByVal lastFillPrice As Double, ByVal clientId As Integer, ByVal whyHeld As String)**

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

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

status	<p>The order status. Possible values include:</p> <ul style="list-style-type: none"> <li>• PendingSubmit - indicates that you have transmitted the order, but have not yet received confirmation that it has been accepted by the order destination.</li> <li>• 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. PendingSubmit and PendingCancel order statuses are not sent by the system and should be explicitly set by the API developer when an order is canceled.</li> <li>• PreSubmitted - indicates that a simulated order type has been accepted by the system and that this order has yet to be elected. The order is held in the system until the election criteria are met. At that time the order is transmitted to the order destination as specified.</li> <li>• Submitted - indicates that your order has been accepted at the order destination and is working.</li> <li>• Cancelled - indicates that the balance of your order has been confirmed canceled by the system. This could occur unexpectedly when the destination has rejected your order.</li> <li>• Filled - indicates that the order has been completely filled.</li> <li>• 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.</li> </ul>
filled	<p>Specifies the number of shares that have been executed.</p> <p>For more information about partial fills, see <a href="#">Order Status for Partial Fills</a>.</p>
remaining	Specifies the number of shares still outstanding.
avgFillPrice	The average price of the shares that have been executed. This parameter is valid only if the <b>filled</b> parameter value is greater than zero. Otherwise, the price parameter will be zero.
permId	The id used to identify orders. Remains the same over sessions.
parentId	The order ID of the parent order, used for bracket and auto trailing stop orders.
lastFillPrice	The last price of the shares that have been executed. Valid only if the filled parameter value is greater than zero. Otherwise, the price parameter will be zero.
clientId	- The ID of the client who placed the order. Note that application orders have a fixed clientId and orderId of 0 that distinguishes them from API orders.

**openOrderEx()**

This method is called to feed in open orders.

**Sub openOrderEx(ByVal orderId As Integer, ByVal contract As TWSLib.IContract, ByVal order As TWSLib.IOrder, ByVal orderState As TWSLib.IOrderState)**

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

**nextValidId()**

This event is called after a successful connection to TWS.

**Sub nextValidId(ByVal id As Integer)**

Parameter	Description
id	The next available order ID received from TWS upon connection. Increment all successive orders by one based on this ID.

**permId()**

This event is always received after an order Status event. It gives the permId for the specified order id. The permId will remain the same from session to session.

**Sub permId(ByVal id As Integer, ByVal permId As Integer)**

Parameter	Description
id	The next available order ID received from TWS upon connection. Increment all successive orders by one based on this ID.
permId	This id will remain the same from session to session

**updateAccountValue()**

This event updates a single account value.

**Sub updateAccountValue(ByVal key As String, ByVal value As String, ByVal currency As String, ByVal accountName As String)**



Parameter	Description
-----------	-------------

key	<p>A string that indicates one type of account value. Below are some of the keys sent by TWS.</p> <ul style="list-style-type: none"> <li>• Account Type</li> <li>• Account Code</li> <li>• Available Funds</li> <li>• Buying Power</li> <li>• CashBalance - Account cash balance</li> <li>• Currency - Currency string</li> <li>• DayTradesRemaining - Number of day trades left</li> <li>• EquityWithLoanValue - Equity with Loan Value</li> <li>• Excess Liquidity</li> <li>• Full Available Funds</li> <li>• Full Excess Liquidity</li> <li>• Full Init Margin Req</li> <li>• Full Maint Margin Req</li> <li>• Future Option Value</li> <li>• Futures PNL</li> <li>• Gross Position Value</li> <li>• InitMarginReq - Current initial margin requirement</li> <li>• Leverage</li> <li>• Look Ahead Available Funds</li> <li>• Look Ahead Next Change</li> <li>• Look Ahead Excess Liquidity</li> <li>• Look Ahead Margin Req</li> <li>• Look Ahead Maint Margin Req</li> <li>• LongOptionValue - Long option value</li> <li>• MaintMarginReq - Current maintenance margin</li> <li>• NetLiquidation - Net liquidation value</li> <li>• OptionMarketValue - Option market value</li> <li>• Realized PNL</li> <li>• Settled Cash</li> <li>• ShortOptionValue - Short option value</li> <li>• StockMarketValue - Stock market value</li> </ul>
-----	---

	<ul style="list-style-type: none"> <li>• Total Cash Balance</li> <li>• Total Cash Value</li> <li>• UnalteredInitMarginReq - Overnight initial margin requirement</li> <li>• UnalteredMaintMarginReq - Overnight maintenance margin requirement</li> <li>• Unrealized PNL</li> </ul>
value	The value associated with the key.
curency	Defines the currency of the value, if the value is a monetary amount.
account	states the account the message applies to. Useful for Financial Advisor sub-account messages.

### **updatePortfolioEx()**

This callback is made in response to the [reqAccountUpdates\(\)](#) method.

**Sub updatePortfolioEx(ByVal contract As TWSLib.IContract, ByVal position As Integer, ByVal marketPrice As Double, ByVal marketValue As Double, ByVal averageCost As Double, ByVal unrealizedPNL As Double, ByVal realizedPNL As Double, ByVal accountName As String)**

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

### **updateAccountTime()**

This event sends the time at which the account values and portfolio market prices were calculated.

**Sub updateAccountTime(ByVal timeStamp As String)**

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

### **updateNewsBulletin()**

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

**Sub updateNewsBulletin(ByVal msgId As Short, ByVal msgType As Short, ByVal message As String, ByVal orig-Exchange As String)**

Parameter	Description
msgId	The bulletin ID, increments for each new bulletin.

msgType	Specifies the type of bulletin. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Regular IB news bulletin</li> <li>• 2 = Exchange no longer available for trading.</li> <li>• 3 = Exchange is available for trading.</li> </ul>
message	The bulletins message text.
origExchange	The exchange from which this message originated.

### contractDetailsEx()

This event is called only in response to the [reqContractDetailsEx\(\)](#) method having been called.

**Sub contractDetailsEx(ByVal reqId As Integer, ByVal contractDetails As TWSLib.IContractDetails**

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

### 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.

**Sub contractDetailsEnd(ByVal reqId As Integer)**

Parameter	Description
reqID	Id of the data request.

### bondContractDetails()

Beginning with TWS Version 921, some bond contract data is suppressed and is not be available from the API. All bond contract data continues to be available from Trader Workstation, but only the following bond contract data is available from the API:

- Contract ID
- Minimum Tick
- CUSIP (if you have subscribed to the CUSIP service)
- Rating (if you have subscribed to ratings)

**Sub bondContractDetails(ByVal symbol As String, ByVal secType As String, ByVal cusip As String, ByVal coupon As Double, ByVal maturity As String, ByVal issueDate As String, ByVal ratings As String, ByVal bondType As String, ByVal couponType As String, ByVal convertible As Integer, ByVal callable As Integer, ByVal putable As Integer, ByVal descAppend As String, ByVal exchange As String, ByVal curency As String, ByVal marketName As String, ByVal tradingClass As String, ByVal conId As Integer, ByVal minTick As Double, ByVal orderTypes As**

**String, ByVal validExchanges As String, ByVal nextOptionDate As String, ByVal nextOptionType As String, ByVal nextOptionPartial As Integer, ByVal notes As String)**

Parameter	Description
symbol	The bond symbol.
secType	BOND
cusip	The nine-character bond CUSIP, or 12 character SEDOL.
coupon	The interest rate used to calculate the amount you will receive in interest payments over the course of the year.
maturity	The date on which the issuer must repay the face value of the bond.
issueDate	he date on which the bond was issued.
ratings	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.
bondType	The type of the bond, such as "Corp" for corporate.
couponType	The type of the coupon, such as "FIXED."
convertible	Values are: True or False. If true, the bond can be converted to stock under certain conditions.
callable	Values are: True or False. If true, the bond can be called by the issuer under certain conditions.
putable	Values are: True or False. If true, the bond can be sold back to the issuer under certain conditions.
descAppend	Description string containing further descriptive information about the bond.
exchange	The exchange on which the BOND trades.
currency	The currency in which the bond trades.
marketName	The market name for this contract.
tradingClass	The trading class name for this contract.
conId	The IB contract ID of the bond.
minTick	The minimum price increment of the bond.
orderTypes	The order types that apply to this bond.
validExchanges	A comma-delimited string of exchanges on which this bond trades.
nextOptionDate	Next option date. Applies only to bonds with embedded options.

nextOptionType	Next option type. Applies only to bonds with embedded options.
nextOptionPartial	Next option partial. Applies only to bonds with embedded options (is the next option full or partial?).
notes	Bond notes, if populated for the bond in IB's database.

### execDetailsEx()

This event is called when the [reqExecutionsEx\(\)](#) method is invoked, or when an order is filled.

**Sub execDetailsEx(ByVal reqId As Integer, ByVal contract As TWSLib.IContract, ByVal execution As TWSLib.IExecution)**

Parameter	Description
orderId	The order Id that was specified previously in the call to <a href="#">placeOrderEx()</a> .
contract	This object contains a full description of the contract that was executed.
execution	This structure contains addition order execution details.

### execDetailsEnd()

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

**Sub execDetailsEnd(ByVal reqId As Integer)**

Parameter	Description
reqID	Id of the data request.

### commissionReport()

**Sub commissionReport(ByVal commissionReport As TWSLib.ICommissionReport)**

Parameter	Description
commissionReport	The structure that contains commission details.

### updateMktDepth()

This function is called when the market depth changes.

**Sub updateMktDepth(ByVal id As Integer, ByVal position As Integer, ByVal operation As Integer, ByVal side As Integer, ByVal price As Double, ByVal size As Integer)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktDepth()

position	Specifies the row ID of this market depth entry.
operation	Identifies how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>
side	The side of the book to which this order belongs. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>
price	The order price.
size	The order size.

### updateMktDepthL2()

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

**Sub updateMktDepthL2(ByVal id As Integer, ByVal position As Integer, ByVal marketMaker As String, ByVal operation As Integer, ByVal side As Integer, ByVal price As Double, ByVal size As Integer)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktDepth()
position	Specifies the row id of this market depth entry.
marketMaker	Specifies the exchange hosting this order.
operation	Identifies the how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>
side	Identifies the side of the book that this order belongs to. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>



price	The order price.
size	The order size.

### managedAccounts()

This event id is fired when a successful connection is made to a Financial Advisor account. It is also fired when the req-ManagedAccts() method is invoked.

#### Sub managedAccounts(ByVal accountsList As String)

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

### receiveFA()

This event receives previously requested FA configuration information from TWS.

#### Sub receiveFA(ByVal faDataType As Integer, ByVal cxml As String)

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being received from TWS. Valid values include: <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 =ACCOUNT ALIASES</li> </ul>
cxml	The XML string containing the previously requested FA configuration information.

### accountSummary()

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

#### Sub accountSummary(ByVal messageType As Integer, ByVal version As Integer, ByVal requestId As Integer, ByVal account As String, tag As String, value As String, currency As String)

Parameter	Type	Description
messageType	Integer	Set to 62.
version	Integer	Set to 1.
requestId	Integer	The ID of the data request.
account	String	The account ID.

Parameter	Type	Description
tag	String	<p>The tag from the data request. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i></li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i></li> <li>• <i>RegTMargin</i></li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i></li> <li>• <i>MaintMarginReq</i></li> <li>• <i>AvailableFunds</i></li> <li>• <i>ExcessLiquidity</i></li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i></li> <li>• <i>FullMaintMarginReq</i></li> <li>• <i>FullAvailableFunds</i></li> <li>• <i>FullExcessLiquidity</i></li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i></li> <li>• <i>LookAheadMaintMarginReq</i></li> <li>• <i>LookAheadAvailableFunds</i></li> <li>• <i>LookAheadExcessLiquidity</i></li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> <li>• <i>DayTradesRemaining</i> — The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1"</li> </ul>

Parameter	Type	Description
		means that the user can put on unlimited day trades. <ul style="list-style-type: none"> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>
value	String	The value of the tag.
currency	String	The currency of the tag.

### accountSummaryEnd

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

#### Sub accountSummaryEnd(ByVal reqId As Integer)

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

### position()

This event returns real-time positions for all accounts in response to the [reqPositions\(\)](#) method.

Sub position(ByVal messageId As Integer, ByVal version As Integer, ByVal account As String, ByVal conid As Integer, ByVal underlying As String, ByVal securityType As String, ByVal expiry As String, ByVal strike As String, ByVal right As String, ByVal multiplier As String, ByVal exchange As String, ByVal currency As String, By Val ibLocalSymbol As String, ByVal position As double)

Parameter	Type	Description
messageId	Integer	Set to 62.
version	Integer	Set to 1.
account	String	The account.
conid	Integer	Unique contract identifier.
underlying	String	The symbol of the underlying asset.
securityType	String	The security type.
expiry	String	The expiration date.
strike	String	The strike price.
right	String	Put or call.
multiplier	String	The multiplier.
exchange	String	The exchange.
currency	String	The currency.
ibLocalSymbol	String	The local symbol.
position	double	The position.

**positionEnd()**

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

**Sub positionEnd(ByVal reqId As Integer)**

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

**historicalData()**

This event receives requested historical data from TWS.

**Sub historicalData(ByVal reqId As Integer, ByVal date As String, ByVal open As Double, ByVal high As Double, ByVal low As Double, ByVal close As Double, ByVal volume As Integer, ByVal barCount As Integer, ByVal WAP As Double, ByVal hasGaps As Integer)**

Parameter	Description
reqId	The ticker ID of the request to which this bar is responding.
date	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	The bar opening price.
high	The high price during the time covered by the bar.
low	The low price during the time covered by the bar.
close	The bar closing price.
volume	The volume during the time covered by the bar.
WAP	The weighted average price during the time covered by the bar.
hasGaps	Identifies whether or not there are gaps in the data.

**scannerParameters()****Sub scannerParameters(ByVal xml As String)**

Parameter	Description
xml	An XML document that describes the valid parameters that a scanner parameter can have.

**scannerDataEx()**

This event receives the requested market scanner data results.

**Sub scannerDataEx(ByVal reqId As Integer, ByVal rank As Integer, ByVal contractDetails As TWSLib.IContractDetails, ByVal distance As String, ByVal benchmark As String, ByVal projection As String, ByVal legsStr As String)**

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

### **scannerDataEnd()**

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

**Sub scannerDataEnd(ByVal reqId As Integer)**

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

### **realtimeBar()**

This method receives the real-time bars data results.

**Sub realtimeBar(ByVal tickerId As Integer, ByVal time As Integer, ByVal open As Double, ByVal high As Double, ByVal low As Double, ByVal close As Double, ByVal volume As Integer, ByVal WAP As Double, ByVal Count As Integer)**

Parameter	Description
reqId	The ticker Id of the request to which this bar is responding.
time	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	The bar opening price.
high	The high price during the time covered by the bar.
low	The low price during the time covered by the bar.
close	The bar closing price.
volume	The volume during the time covered by the bar.

wap	The weighted average price during the time covered by the bar.
count	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers.

### **fundamentalData()**

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.

#### **Sub fundamentalData(ByVal reqId As Integer, ByVal data As String)**

Parameter	Description
reqId	The ID of the data request.
data	One of three XML reports: <ul style="list-style-type: none"><li>• Estimates (estimates)</li><li>• Financial statements (finstat)</li><li>• Summary (snapshot)</li></ul>

## ActiveX COM Objects

The tables below define properties for the following objects:

- [IExecution](#)
- [IExecutionFilter](#)
- [ICommissionReport](#)
- [IContract](#)
- [IContractDetails](#)
- [IComboLeg](#)
- [IComboLegList](#)
- [IOrder](#)
- [IOrderState](#)
- [IScannerSubscription](#)
- [ITagValueList](#)
- [ITagValue](#)
- [IUnderComp](#)

You must use the factory “create” methods to create the COM objects in this section. Once a COM object has been created by a factory method, the COM object is tied to a corresponding TWS COM object (an instance of the COM object). Do not try to pass a COM object to another instance of a TWS COM object.

### IExecution

Attribute	Description
acctNumber() As String	The customer account number.
avgPrice() As Double	Average price. Used in regular trades, combo trades and legs of the combo. Includes commissions.
clientId() As Integer	The id of the client that placed the order.  <b>Note:</b> TWS orders have a fixed client id of "0."
cumQty() As Integer	Cumulative quantity. Used in regular trades, combo trades and legs of the combo.
exchange() As String	Exchange that executed the order.
execId() As String	Unique order execution id.

liquidation() As Integer	Identifies the position as one to be liquidated last should the need arise.
orderId() As Integer	The order id. <b>Note:</b> TWS orders have a fixed order id of "0."
permId() As Integer	The TWS id used to identify orders, remains the same over TWS sessions.
price() As Double	The order execution price, not including commissions.
shares() As Integer	The number of shares filled.
side() As String	Specifies if the transaction was a sale or a purchase. Valid values are: <ul style="list-style-type: none"> <li>• BOT</li> <li>• SLD</li> </ul>
time() As String	The order execution time.
evRule() As 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.
evMultiplier As 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.

## IExecutionFilter

Attribute	Description
acctCode() As String	Filter the results of the reqExecutionsEx() method based on an account code. Note: this is only relevant for Financial Advisor (FA) accounts.
clientId() As Integer	Filter the results of the reqExecutionsEx() method based on the clientId.
exchange() As String	Filter the results of the reqExecutionsEx() method based on the order exchange.
secType() String	Filter the results of the reqExecutionsEx() method based on the order security type. <b>Note:</b> Refer to the Contract object for the list of valid security types.



side() As String	Filter the results of the reqExecutionsEx() method based on the order action.  <b>Note:</b> Refer to the Order object for the list of valid order actions.
symbol() As String	Filter the results of the reqExecutionsEx() method based on the order symbol.
time() As String	Filter the results of the reqExecutionsEx() method based on execution reports received after the specified time. The format for timeFilter is "yyyymmdd-hh:mm:ss"

### ICommissionReport

Attribute	Description
commission() As Double	The commission amount.
currency() As String	The currency.
execId() As String	Unique order execution id.
realizedPNL() As Double	The amount of realized Profit and Loss.
yield() As Double	The yield.
yieldRedemptionDate() As Double	Takes the YYYYMMDD format.

### IContract

Attribute	Description
comboLegs() As Object	Dynamic memory structure used to store the leg definitions for this contract.
comboLegsDescrip() As String	Description for combo legs
conId() As Integer	The unique contract identifier.
currency() As 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.
exchange() As String	The order destination, such as Smart.
expiry() As String	The expiration date. Use the format YYYYMM.

includeExpired() As Integer	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,
localSymbol() As String	This is the local exchange symbol of the underlying asset.
multiplier() As String	Allows you to specify a future or option contract multiplier. This is only necessary when multiple possibilities exist.
primaryExch() As String	Identifies the listing exchange for the contract (do not list SMART).
right() As String	Specifies a Put or Call. Valid values are: P, PUT, C, CALL.
secId as String	Unique identifier for the secIdType.
secIdType As String	Security identifier, when querying contract details or when placing orders. Supported identifiers are: <ul style="list-style-type: none"> <li>• ISIN (Example: Apple: US0378331005)</li> <li>• CUSIP (Example: Apple: 037833100)</li> <li>• SEDOL (Consists of 6-AN + check digit. Example: BAE: 0263494)</li> <li>• RIC (Consists of exchange-independent RIC Root and a suffix identifying the exchange. Exa</li> </ul>
secType() As String	This is the security type. Valid values are: <ul style="list-style-type: none"> <li>• STK</li> <li>• OPT</li> <li>• FUT</li> <li>• IND</li> <li>• FOP</li> <li>• CASH</li> <li>• BAG</li> </ul>
strike() As Double	The strike price.
symbol() As String	This is the symbol of the underlying asset.
tradingClass() As String	The trading class name for this contract.

**ContractDetails**

Attribute	Description
category() As String	The industry category of the underlying. For example, InvestmentSvc.
contractMonth() As String	The contract month. Typically the contract month of the underlying for a futures contract.
industry() As String	The industry classification of the underlying/product. For example, Financial.
liquidHours() As String	The liquid trading hours of the product. For example, 20090507:0930-1600;20090508:CLOSED.
longName() As String	The descriptive name of the asset.
marketName() String	The market name for this contract.
minTick() As Double	The minimum price tick.
orderTypes() As String	The list of valid order types for this contract.
priceMagnifier() Integer	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.
ratings() As String	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.
secIdList() As Object	A list of contract identifiers that the customer is allowed to view (CUSIP, ISIN, etc.)
subcategory() As String	The industry subcategory of the underlying. For example, Brokerage.
summary() As Object	A contract summary.
timeZoneId() As String	The ID of the time zone for the trading hours of the product. For example, EST.
tradingHours() As String	The trading hours of the product. For example, 20090507:0700-1830,1830-2330;20090508:CLOSED.
underConId() As String	The underlying contract ID.
validExchanges() As String	The list of exchanges this contract is traded on.

evRule() As 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.
evMultiplier As 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.
<b>Bond Values</b>	
bondType() As String	The type of bond, such as "CORP."
callable() As Integer	Values are True or False. If true, the bond can be called by the issuer under certain conditions.
convertible() As Integer	Values are True or False. If true, the bond can be converted to stock under certain conditions.
coupon() As Double	The interest rate used to calculate the amount you will receive in interest payments over the course of the year.
couponType() As String	The type of bond coupon, such as "FIXED."
cusip() As String	The nine-character bond CUSIP or the 12-character SEDOL.
descAppend() As String	A description string containing further descriptive information about the bond.
issueDate() As String	The date the bond was issued.
maturity() As String	The date on which the issuer must repay the face value of the bond.
nextOptionDate)_ As String	Applies to bonds with embedded options.
nextOptionPartial() As Integer	Applies to bonds with embedded options.
nextOptionType() As String	Applies to bonds with embedded options.
notes() As String	If populated for the bond in IB's database
puttable() As Integer	Values are True or False. If true, the bond can be sold back to the issuer under certain conditions.

## IComboLeg

Attribute	Description
action() As String	The side (buy or sell) for the leg you are constructing.

conId() As Integer	The unique contract identifier specifying the security.
exchange() As String	The exchange to which the complete combination order will be routed.
openClose() As Integer	Specifies whether the order is an open or close order. Valid values are: <ul style="list-style-type: none"> <li>• 0 - Same as the parent security. This is the only option for retail customers.</li> <li>• 1 - Open. This value is only valid for institutional customers.</li> <li>• 2 - Close. This value is only valid for institutional customers.</li> <li>• Unknown - (3)</li> </ul>
ratio() As Integer	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.
<b>For Short Sale Stock Legs</b>	
designatedLocation() As String	If shortSaleSlot == 2, the designatedLocation must be specified. Otherwise leave blank or orders will be rejected.
shortSaleSlot() Integer	For institutional customers only. <ul style="list-style-type: none"> <li>• 0 - inapplicable (i.e. retail customer or not short leg)</li> <li>• 1 - clearing broker</li> <li>• 2 - third party. If this value is used, you must enter a designated location.</li> </ul>

**IComboLegList**

Attribute	Description
Add() As Object	Adds combo legs to a combo leg list.
Count() As Integer	Leg count.
Item(Integer) As Object	Get leg by index.

**IOrder**

Attribute	Description
<b>Order Identifiers</b>	
clientId() As Integer	The id of the client that placed this order.

orderId() As Integer	The id for this order.
permId() As Integer	The TWS id used to identify orders, remains the same over TWS sessions.
<b>Main Order Fields</b>	
action() As String	Identifies the side. Valid values are: BUY, SELL, SSHORT
auxPrice() As Double	This is the STOP price for stop-limit orders, and the offset amount for relative orders. In all other cases, specify zero.
lmtPrice() As 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.
orderType() As String	Identifies the order type.  For more information about supported order types, see <a href="#">Supported Order Types</a> .
totalQuantity() As Integer	The order quantity.
<b>Extended Order Fields</b>	
allOrNone() As Integer	0 = no, 1 = yes
blockOrder() As Integer	Specifies that the order is an ISE Block order.
displaySize() As Integer	The publicly disclosed order size, used when placing Iceberg orders.
goodAfterTime() As String	The trade's "Good After Time," format "YYYYMMDD hh:mm:ss (optional time zone)" Use an empty String if not applicable.
goodTillDate() As 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() As Integer	Specifies that the order will not be visible when viewing the market depth. This option only applies to orders routed to the ISLAND exchange.
minQty() As Integer	Identifies a minimum quantity order type.
ocaGroup() As String	Identifies an OCA (one cancels all) group.

ocaType() As Integer	<p>Tells how to handle remaining orders in an OCA group when one order or part of an order executes. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = Cancel all remaining orders with block</li> <li>• 2 = Remaining orders are proportionately reduced in size with block</li> <li>• 3 = Remaining orders are proportionately reduced in size with no block</li> </ul> <p>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.</p>
orderRef() As 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.
outsideRth() As Integer	Specifies whether orders can trigger or fill outside of regular trading hours or not.
overridePercentageConstraints() As Integer	<p>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 parameter's value to True.</p> <p>Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 = False</li> <li>• 1 = True</li> </ul>
parentId() As Integer	The order ID of the parent order, used for bracket and auto trailing stop orders.
percentOffset() As Double	The percent offset amount for relative orders.

rule80A() As String	<p>Values include:</p> <ul style="list-style-type: none"> <li>• Individual = 'I'</li> <li>• Agency = 'A',</li> <li>• AgentOtherMember = 'W'</li> <li>• IndividualPTIA = 'J'</li> <li>• AgencyPTIA = 'U'</li> <li>• AgentOtherMemberPTIA = 'M'</li> <li>• IndividualPT = 'K'</li> <li>• AgencyPT = 'Y'</li> <li>• AgentOtherMemberPT = 'N'</li> </ul>
sweepToFill() As Integer	Specifies if the order is a Sweep-to-Fill order.
tif() As String	The time in force. Valid values are: DAY, GTC, IOC, GTD.
transmit() As Integer	Specifies whether the order will be transmitted by TWS.
triggerMethod() As Integer	<p>Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 1 - use "double bid/ask" function, where stop orders are triggered based on two consecutive bid or ask prices.</li> <li>• 2 - "last" function, where stop orders are triggered based on the last price.</li> <li>• 3 double last function.</li> <li>• 4 bid/ask function.</li> <li>• 7 last or bid/ask function.</li> <li>• 8 mid-point function.</li> </ul>
trailStopPrice() As Double	For TRAILLIMIT orders only



trailingPercent() As Double	<p>Specify the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field:</p> <ul style="list-style-type: none"> <li>• This field is mutually exclusive with the existing trailing amount. That is, the API client can send one or the other but not both.</li> <li>• This field is read AFTER the stop price (barrier price) as follows: deltaNeutralAuxPrice stopPrice trailingPercent scale order attributes</li> <li>• The field will also be sent to the API in the openOrder message if the API client version is <math>\geq</math> 56. It is sent after the stopPrice field as follows: stopPrice trailingPct basisPoint</li> </ul>
<b>Financial Advisor Fields</b>	
faGroup() As String	The Financial Advisor group the trade will be allocated to -- use an empty String if not applicable.
faMethod() As String	The Financial Advisor allocation function the trade will be allocated with -- use an empty String if not applicable.
faPercentage() As String	The Financial Advisor percentage concerning the trade's allocation -- use an empty String if not applicable.
faProfile() As String	The Financial Advisor allocation profile the trade will be allocated to -- use an empty String if not applicable.
<b>Institutional (non-cleared) Only</b>	
designatedLocation() As String	Used only when shortSaleSlot = 2.
openClose() As String	For institutional customers only. Valid values are O, C.
origin() As Integer	The order origin. For institutional customers only. Valid values are 0 = customer, 1 = firm
shortSaleSlot() As Integer	Valid values are 1 or 2.
<b>SMART Routing Only</b>	
discretionaryAmt() As Double	The amount off the limit price allowed for discretionary orders.
eTradeOnly() As Integer	Trade with electronic quotes. 0 = no, 1 = yes

firmQuoteOnly() As Integer	Trade with firm quotes. 0 = no, 1 = yes
nbboPriceCap() As Double	Maximum smart order distance from the NBBO.
optOutSmartRouting() As Integer	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.
notHeld() As Integer	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.
<b>BOX or VOL Orders Only</b>	
auctionStrategy() As Integer	Values include: <ul style="list-style-type: none"> <li>• match = 1</li> <li>• improvement = 2</li> <li>• transparent = 3</li> </ul> For orders on BOX only.
<b>BOX Exchange Orders Only</b>	
delta() As Double	The stock delta. For orders on BOX only.
startingPrice() As Double	The auction starting price. For orders on BOX only.
stockRefPrice() As 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.
<b>Pegged-to-Stock and VOL Orders Only</b>	
stockRangeLower() As Double	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
stockRangeUpper() As Double	The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
<b>Volatility Orders Only</b>	
continuousUpdate() As Integer	VOL orders only. Specifies whether TWS will automatically update the limit price of the order as the underlying price moves.

deltaNeutralOrderType() As 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.
deltaNeutralAuxPrice() As Integer	VOL orders only. Use this field to enter a value if the value in the <i>deltaNeutralOrderType</i> field is an order type that requires an Aux price, such as a REL order.
referencePriceType() As Integer	VOL orders only. Specifies how you want TWS to calculate the limit price for options, and for stock range price monitoring. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Average of NBBO</li> <li>• 2 = NBB or the NBO depending on the action and right.</li> </ul>
volatility() As 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.
volatilityType() As Integer	Values include: <ul style="list-style-type: none"> <li>• 1 = Daily volatility</li> <li>• 2 = Annual volatility</li> </ul>
deltaNeutralOpenClose() As 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.
deltaNeutralShortSale () As Integer	Used when the hedge involves a stock and indicates whether or not it is sold short.
deltaNeutralShortSaleSlot() As Integer	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.
deltaNeutralDesignatedLocation() As String	Used only when deltaNeutralShortSaleSlot = 2.
<b>Combo Orders Only</b>	
basisPoints() As Double	For EFP orders only
basisPointsType() As Integer	For EFP orders only

<b>Scale Orders Only</b>	
scaleAutoReset() As Integer	For extended Scale orders.
scaleInitFillQty() As Integer	For extended Scale orders.
scaleInitLevelSize() As Integer	For Scale orders: Defines the size of the first, or initial, order component.
scaleInitPosition() As Integer	For extended Scale orders.
scalePriceIncrement() As Double	For Scale orders: Defines the price increment between scale components. This field is required.
scalePriceAdjustInterval() As Integer	For extended Scale orders.
scalePriceAdjustValue() As Double	For extended Scale orders.
scaleProfitOffset() As Double	For extended Scale orders.
scaleRandomPercent() As Integer	For extended Scale orders.
scaleSubsLevelSize() As Integer	For Scale orders: Defines the order size of the subsequent scale order components. Used in conjunction with scaleInitLevelSize().
<b>Hedge Orders Only</b>	
hedgeParam() As String	Beta = x for Beta hedge orders, ratio = y for Pair hedge order
hedgeType() As String	For hedge orders. Possible values are: <ul style="list-style-type: none"> <li>• D = Delta</li> <li>• B = Beta</li> <li>• F = FX</li> <li>• P = Pair</li> </ul>
<b>Clearing Information</b>	
account() As String	The account. For institutional customers only.
clearingAccount() As String	For IBExecution customers: Specifies the true beneficiary of the order. This value is required for FUT/FOP orders for reporting to the exchange.
clearingIntent() As String	For IBExecution customers: Valid values are: IB, Away, and PTA (post trade allocation).

settlingFirm() As String	Institutional only.
<b>Algo Orders Only</b>	
algoStrategy() As String	For information about API Algo orders, see <a href="#">IBAlgo Parameters</a> .
algoParams() As Object	Support for IBAlgo parameters.
<b>What If</b>	
whatIf() As Integer	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.
<b>Smart Combo Routing</b>	
smartComboRoutingParams() As Object	Support for <a href="#">Smart combo routing</a> .
<b>Order Combo Legs</b>	
OrderComboLegs() As Object	Holds attributes for all legs in a combo order.

### OrderComboLeg

Attribute	Description
double price	Order-specific leg price.

### IOrderState

Attribute	Description
commission() As Double	Shows the commission amount on the order.
commissionCurrency() As String	Shows the currency of the commission value.
equityWithLoan() As String	Shows the impact the order would have on your equity with loan value.
initMargin() As String	Shows the impact the order would have on your initial margin.

maintMargin() As String	Shows the impact the order would have on your maintenance margin.
maxCommission() As 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.
minCommission() As 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.
status() As String	Displays the order status.
warningText() As String	Displays a warning message if warranted.

### IScannerSubscription

Attribute	Description
averageOptionVolumeAbove () As Integer	Can leave empty.
couponRateAbove() As String	Filter out contracts with a coupon rate lower than this value. Can be left blank.
couponRateBelow() As String	Filter out contracts with a coupon rate higher than this value. Can be left blank.
excludeConvertible() As Integer	Filter out convertible bonds. Can be left blank.
instrument() As String	Defines the instrument type for the scan.
locations() As String	The location.
marketCapAbove() As Double	Filter out contracts with a market cap lower than this value. Can be left blank.
marketCapBelow() As Double	Filter out contracts with a market cap above this value. Can be left blank.
maturityDateAbove() As String	Filter out contracts with a maturity date earlier than this value. Can be left blank.
maturityDateBelow() As String	Filter out contracts with a maturity date later than this value. Can be left blank.
moodyRatingAbove() As String	Filter out contracts with a Moody rating below this value. Can be left blank.
moodyRatingBelow() As String	Filter out contracts with a Moody rating above this value. Can be left blank.
numberOfRows() As Integer	Defines the number of rows of data to return for a query.

priceAbove() As Double	Filter out contracts with a price lower than this value. Can be left blank.
priceBelow() As Double	Filter out contracts with a price higher than this value. Can be left blank.
scanCode() As String	Can be left blank.
scannerSettingPairs() As String	Can leave empty. For example, a pairing "Annual, true" used on the "top Option Implied Vol % Gainers" scan would return annualized volatilities.
spRatingAbove() As String	Filter out contracts with an S&P rating below this value. Can be left blank.
spRatingBelow() As String	Filter out contracts with an S&P rating above this value. Can be left blank.
stockTypeFilter() As String	Valid values are: <ul style="list-style-type: none"> <li>• CORP = Corporation</li> <li>• ADR = American Depositary Receipt</li> <li>• ETF = Exchange Traded Fund</li> <li>• REIT = Real Estate Investment Trust</li> <li>• CEF = Closed End Fund</li> </ul>
volumeAbove() As Integer	Filter out contracts with a volume lower than this value. Can be left blank.

### ITagValueList

Attribute	Description
Count() As Integer	The number of tag-value pairs (IBAlgo parameters).
Item(Integer) As Object	A tag-value pair (IBAlgo parameter). For more information, see <a href="#">IBAlgo Parameters</a> .

### ITagValue

Attribute	Description
tag() As String	An IBAlgo order parameter. For more information, see <a href="#">IBAlgo Parameters</a> .
value() As String	The value of the IBAlgo parameter.

### IUnderComp

Attribute	Description
-----------	-------------

conId() As Integer	The unique contract identifier specifying the security. Used for Delta-Neutral Combo contracts.
delta() As Double	The underlying stock or future delta. Used for Delta-Neutral Combo contracts.
price() As Double	The price of the underlying. Used for Delta-Neutral Combo contracts.



## ActiveX Properties

The table below defines properties you can use when connecting to a server using ActiveX.

Property	Description
String TwsConnectionTime	Connection time.
long serverVersion	Server Version.

## 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 reqContractDetailsEx() method.
- Include each leg on the IComboLeg COM object by populating the related fields.
- Implement the placeOrderEx() method with the IContract and IOrder COM objects.

### To place this combo order

1. Get the Contract IDs for both leg definitions:

```
'First Leg
Dim con1 As TWSLib.IContract
con1 = Tws1.createContract

con1.symbol = "GOOG"
con1.secType = "OPT"
con1.expiry = "200909"
con1.strike = 150.0
con1.right = "C"
con1.multiplier = "100"
con1.exchange = "SMART"
con1.currency = "USD"

Tws1.reqContractDetailsEx(1, con1)

'Second Leg
Dim con2 As TWSLib.IContract
con2 = Tws1.createContract

con2.symbol = "GOOG"
con2.secType = "OPT"
con2.expiry = "201101"
con2.strike = 150.0
con2.right = "C"
con2.multiplier = "100"
con2.exchange = "SMART"
```

```

con2.currency = "USD"

Tws1.reqContractDetailsEx(2, con2)

'All conId numbers are delivered by the ContractDetail()

Private Sub Tws1_contractDetailsEx(ByVal sender As Object, ByVal e As AxTW-
SLib._DTwsEvents_contractDetailsExEvent) Handles Tws1.contractDetailsEx

Dim contractDetails As TWSLib.IContractDetails
contractDetails = e.contractDetails

Dim contract As TWSLib.IContract
contract = contractDetails.summary

'reqId = 1 is corresponding to the first request or first leg
'reqId = 2 is corresponding to the second request or second leg

If e.reqId = 1 Then
leg1 = contract.conId 'to obtain conId for the first leg
End If

If e.reqId = 2 Then
leg2 = contract.conId 'to obtain conId for the second leg
End If

End Sub

```

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

```

TWSLib.IComboLegList
addAllLegs = Tws1.createComboLegList

'First Combo leg
Dim Leg1 As TWSLib.IComboLeg
Leg1 = addAllLegs.Add()

Leg1.conId = leg1_conId
Leg1.ratio = 1
Leg1.action = "SELL"
Leg1.exchange = "SMART"
Leg1.openClose = 0
Leg1.shortSalesSlot = 0
Leg1.designatedLocation = ""

' Second Combo leg
Dim Leg2 As TWSLib.IComboLeg
Leg2 = addAllLegs.Add()

Leg1.conId = leg2_conId
Leg1.ratio = 1
Leg1.action = "BUY"
Leg1.exchange = "SMART"
Leg1.openClose = 0

```

```
Leg1.shortSaleSlot = 0  
Leg1.designatedLocation = ""
```

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

```
Dim contract As TWSLib.IContract  
contract = Tws1.createContract  
  
contract.symbol = "USD"  
contract.secType = "BAG"  
contract.exchange = "SMART"  
contract.currency = "USD"  
contract.comboLegs = addAllLegs  
  
Dim order As TWSLib.IOrder  
order = Tws1.createOrder  
  
order.action = "BUY"  
order.totalQuantity = 1  
order.orderType = "MKT"  
  
Tws1.placeOrderEx(OrderId, contract, order)
```

## C++

This chapter describes the C++ API, including the following topics:

- [Linking to TWS using the TwSocketClient.dll](#)
- [Using the C++ TestSocketClient Sample Program](#)
- [Class EClientSocket Functions](#)
- [Class EWrapper Functions](#)
- [SocketClient Properties](#)
- [Placing a Combination Order](#)

## Linking to TWS using the TwSocketClient.dll

### To link to TWS using the TwSocketClient.dll

1. Create a Windows application using MS Visual Studio (version 5.0 or higher).
2. Add the full path to the `\SocketClient\include` directory in your API installation folder to your project's include path. This should be done for any individual project that accesses the TwSocketClient library's header files.
3. Add the full path to the `\SocketClient\lib\TwSocketclient.lib` file in your API installation directory to your project's libraries path. This should be done for any individual project that accesses the TwSocketClient library.
4. Include `EWWrapper.h` and `EClientSocket.h` in any Visual C++ source code that accesses their functionality and data structures.
5. Subclass the `EWWrapper` class.
6. Override the following functions:

Ewrapper Function	Description
tickPrice()	Handles market data.
tickSize()	
tickOptionComputation()	
tickGeneric()	
tickString()	
tickEFP()	
orderStatus()	Receives order status.
openOrder()	Receives open orders.
error()	Receives error information.
connectionClosed()	Notifies when TWS terminates the connection.
updateAccountValue()	Receives current account values.
updateAccountTime()	Receives the last time account information was updated.
updatePortfolio()	Receives current portfolio information.

nextValidId()	Receives the next valid order ID upon connection.
contractDetails()	Receives contract information.
contractDetailsEnd()	Identifies the end of a given contract details request.
bondContractDetails()	Receives bond contract information.
execDetails()	Receives execution report information.
updateMktDepth()	Receives market depth information.
updateMktDepthL2()	Receives Level II market depth information.
updateNewsBulletin()	Receives IB news bulletins.
managedAccounts()	Receives a list of Financial Advisor (FA) managed accounts.
receiveFA()	Receives FA configuration information.
historicalData()	Receives historical data results.
scannerParameters()	Receives an XML document that describes the valid parameters of a scanner subscription.
scannerData()	Receives market scanner results.
realTimeBar()	Receives real-time bars.
currentTime()	Receives the current system time on the server.
fundamentalData()	Receives Reuters global fundamental market data.

7. Instantiate the EClientSocket class.
8. Call the following functions:
  - Import `com.ib.client.*` into your source code file.
  - Implement the EWrapper interface. This class will receive messages from the socket.
  - Override the following functions:

Wrapper Functions	Description
-------------------	-------------

tickPrice()	Handles market data.
tickSize()	
tickOptionComputation()	
tickGeneric()	
tickString()	
tickEFP()	
orderStatus()	Receives order status.
openOrder()	Receives open orders.
error()	Receives error information.
connectionClosed()	Notifies when TWS terminates the connection.
updateAccountValue()	Receives current account values.
updateAccountTime()	Receives the last time account information was updated.
updatePortfolio()	Receives current portfolio information.
nextValidId()	Receives the next valid order ID upon connection.
contractDetails()	Receives contract information.
contractDetailsEnd()	Identifies the end of a given contract details request.
bondContractDetails()	Receives bond contract information.
execDetails()	Receives execution report information.
updateMktDepth()	Receives market depth information.
updateMktDepthL2()	Receives Level II market depth information.
updateNewsBulletin()	Receives IB news bulletins.
managedAccounts()	Receives a list of Financial Advisor (FA) managed accounts.



receiveFA()	Receives FA configuration information.
historicalData()	Receives historical data results.
scannerParameters()	Receives an XML document that describes the valid parameters of a scanner subscription.
scannerData()	Receives market scanner results.
realTimeBar()	Receives real-time bars.
currentTime()	Receives the current system time on the server.
fundamentalData()	Receives Reuters global fundamental market data.

- Instantiate the EClientSocket class. This object will be used to send messages to TWS.
- Call the following functions:

EClientSocket Functions	Description
eConnect()	Connects to TWS.
eDisconnect()	Disconnects from TWS.
reqMktData()	Requests market data.
cancelMktData()	Cancels market data.
reqMktDepth()	Requests market depth.
cancelMktDepth()	Cancels market depth.
reqContractDetails()	Requests contract details.
placeOrder()	Places an order.
cancelOrder()	Cancels an order.
reqAccountUpdates()	Requests account values, portfolio, and account update time information.
reqExecutions()	Requests a list of the day's execution reports.
reqOpenOrders()	Requests a list of current open orders for the requesting client and associates TWS open orders with the client. The association only occurs if the requesting client has a Client ID of 0.
reqAllOpenOrders()	Requests a list of all open orders.

reqAutoOpenOrders()	Automatically associates a new TWS with the client. The association only occurs if the requesting client has a Client ID of 0.
reqNewsBulletin()	Requests IB news bulletins.
cancelNewsBulletins()	Cancels IB news bulletins.
setServerLogLevel()	Sets the level of API request and processing logging.
reqManagedAccts()	Requests a list of Financial Advisor (FA) managed account codes.
requestFA()	Requests FA configuration information from TWS.
replaceFA()	Modifies FA configuration information from the API.
reqScannerParameters()	Requests an XML document that describes the valid parameters of a scanner subscription.
reqScannerSubscription()	Requests market scanner results.
cancelScannerSubscription()	Cancels a scanner subscription.
reqHistoricalData()	Requests historical data.
cancelHistoricalData()	Cancels historical data.
reqRealTimeBars()	Requests real-time bars.
cancelRealTimeBars()	Cancels real-time bars.
exerciseOptions()	Exercises options.
reqCurrentTime()	Requests the current server time.
serverVersion()	Returns the version of the TWS instance to which the API application is connected.
TwsConnectionTime()	Returns the time the API application made a connection to TWS.
reqFundamentalData()	Requests Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.
cancelFundamentalData()	Cancels Reuters global fundamental data.

To run the program, ensure that the `TwsocketClient.dll` is in the same directory as your executable, or in your path. By default, the `TwsocketClient.dll` file installs into your `C:\Windows\system` or `C:\WINNT\system32` directory.

## Using the C++ TestSocketClient Sample Program

You can access the IB trading system via TWS or the IB Gateway through a Microsoft Windows-based C++ application using the TWSSocketClient.dll component. Before you can connect to TWS using the SocketClient component, you must:

- Install the socket client component and sample programs
- If you are using TWS, configure it to support the API components
- Have Microsoft Visual Studio (Visual C++ 5.0 or higher) installed on your PC.

The TestSocketClient program is a sample program that shows you how to use sockets to connect to TWS from a Microsoft Windows-based C++ application.

### To run the pre-built sample application

We've included a complete C++ client with our API software. To run this sample application, go to your TWS API installation folder, then open the \TestSocketClient\Release folder. Run the file named client2.exe.

### To run the TestSocketClient program from Microsoft Visual Studio 2008

#### To create the project file and compile the TestSocketClient application (client2.exe):

1. Download and install the latest version of the API software.
2. Create the folder where all project related files will be located. For example, *C:\SampleSocketClient*.
3. Copy the folders *TestSocketClient*, *Shared* and *SocketClient* into the folder you created in Step 2. (For example, *C:\SampleSocketClient*.)
4. In Visual Studio, select **New > Project From Existing Code** from the **File** menu.
5. Select **Visual C++** as the project type.
6. In the Create New Project from Existing Code Files dialog:
  - a. For the Project File location, select the folder you created in Step 2. (For example, *C:\SampleSocketClient*.)
  - b. For the Project Name, type **SampleSocketClient**.
  - c. Click **Finish**.
7. Right-click the project in the Solution Explorer and select **Properties**. In the Property Pages dialog:
  - a. On the left side of the dialog, select **Configuration Parameters > C/C++ > General**. In the Additional Include Directories field on the right side of the dialog, type:  
  
**./Shared;./SocketClient/src**
  - b. On left side of the dialog, select **Configuration Parameters > C/C++ > Code Generation**. In the Runtime Library field on the right side of the dialog, select **Multi-threaded (/MT)**.
8. Click **OK** to save your changes to the project properties.
9. Build the project.

## Class EClientSocket Functions

The list below define the class EClientSocket functions you can use when connecting to TWS. The list of functions includes:

<b>Connection and Server</b> <a href="#"><u>EClientSocket()</u></a> <a href="#"><u>eConnect()</u></a> <a href="#"><u>eDisconnect()</u></a> <a href="#"><u>isConnected()</u></a> <a href="#"><u>reqCurrentTime()</u></a> <a href="#"><u>serverVersion()</u></a> <a href="#"><u>TwsConnectionTime()</u></a> <a href="#"><u>setLogLevel()</u></a> <a href="#"><u>checkMessages()</u></a>  <b>Market Data</b> <a href="#"><u>reqMktData()</u></a> <a href="#"><u>cancelMktData()</u></a> <a href="#"><u>calculateImpliedVolatility()</u></a> <a href="#"><u>cancelCalculateImpliedVolatility()</u></a> <a href="#"><u>calculateOptionPrice()</u></a> <a href="#"><u>cancelCalculateOptionPrice()</u></a> <a href="#"><u>reqMarketDataType()</u></a>  <b>Orders</b> <a href="#"><u>placeOrder()</u></a> <a href="#"><u>cancelOrder()</u></a> <a href="#"><u>reqOpenOrders()</u></a> <a href="#"><u>reqAllOpenOrders()</u></a> <a href="#"><u>reqAutoOpenOrders()</u></a> <a href="#"><u>reqIDs()</u></a> <a href="#"><u>exerciseOptions()</u></a>  <b>Account</b> <a href="#"><u>reqAccountUpdates()</u></a>  <b>Executions</b> <a href="#"><u>reqExecutions()</u></a>	<b>Contract Details</b> <a href="#"><u>reqContractDetails()</u></a>  <b>Market Depth</b> <a href="#"><u>reqMktDepth()</u></a> <a href="#"><u>cancelMktDepth()</u></a>  <b>News Bulletins</b> <a href="#"><u>reqNewsBulletins()</u></a> <a href="#"><u>cancelNewsBulletins()</u></a>  <b>Financial Advisors</b> <a href="#"><u>reqManagedAccts()</u></a> <a href="#"><u>requestFA()</u></a> <a href="#"><u>replaceFA()</u></a> <a href="#"><u>reqAccountSummary()</u></a> <a href="#"><u>cancelAccountSummary()</u></a> <a href="#"><u>reqPositions()</u></a> <a href="#"><u>cancelPositions()</u></a>  <b>Historical Data</b> <a href="#"><u>reqHistoricalData()</u></a> <a href="#"><u>cancelHistoricalData()</u></a>  <b>Market Scanners</b> <a href="#"><u>reqScannerParameters()</u></a> <a href="#"><u>reqScannerSubscription()</u></a> <a href="#"><u>cancelScannerSubscription()</u></a>  <b>Real Time Bars</b> <a href="#"><u>reqRealTimeBars()</u></a> <a href="#"><u>cancelRealTimeBars()</u></a>  <b>Fundamental Data</b> <a href="#"><u>reqFundamentalData()</u></a> <a href="#"><u>cancelFundamentalData()</u></a>
--	--

**EClientSocket()**

This is the constructor.

**EClientSocket( EWrapper \*ptr)**

Parameter	Description
ptr	The pointer to an object that was derived from the EWrapper base class.

**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."

**bool eConnect( const char \*host, UINT port, int clientId=0)**

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

**eDisconnect()**

Call this function to terminate the connections with TWS. Calling this function does not cancel orders that have already been sent.

**void eDisconnect()**

Parameter	Description
ptr	The pointer to an object that was derived from the EWrapper base class.

**isConnected()**

Call this function to check if there is a connection with TWS

**void isConnected()**

**reqCurrentTime()**

Returns the current system time on the server side.

**void reqCurrentTime()**

**serverVersion()**

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

**serverVersion()**

**setLogLevel()**

The default detail level is ERROR. For more details, see [API Logging](#).

**void setLogLevel(int logLevel)**

Parameter	Description
logLevel	<p>Specifies the level of log entry detail used by the server (TWS) when processing API requests. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = SYSTEM</li> <li>• 2 = ERROR</li> <li>• 3 = WARNING</li> <li>• 4 = INFORMATION</li> <li>• 5 = DETAIL</li> </ul>

**TwConnectionTime()**

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

**TwConnectionTime()**

**checkMessages()**

This function should be called frequently (every 1 second) to check for messages received from TWS.

**void checkMessages()**

**reqMktData()**

Call this function to request market data. The market data will be returned by the tickPrice and tickSize events.

**void reqMktData(TickerID id, const Contract &contract, CString genericTicklist, bool snapshot)**

Parameter	Description
id	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	This structure contains a description of the contract for which market data is being requested.

genericTicklist	A comma delimited list of generic tick types. Tick types can be found in the <a href="#">Generic Tick Types</a> page.
snapshot	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.

### **cancelMktData()**

After calling this function, market data for the specified id will stop flowing.

**void cancelMktData(TickerID id)**

Parameter	Description
id	The ID that was specified in the call to reqMktData().

### **calculateImpliedVolatility()**

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

**void calculateImpliedVolatility(TickerID reqId, Contract &contract, double optionPrice, double underPrice)**

Parameter	Description
reqId	The ticker id.
contract	Describes the contract.
optionPrice	The price of the option.
underPrice	Price of the underlying.

### **cancelCalculateImpliedVolatility()**

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

**calculateImpliedVolatility(TickerId reqId)**

Parameter	Description
reqId	The ticker id.

### **calculateOptionPrice()**

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

**void calculateOptionPrice(TickerId reqId, const Contract &contract, double volatility, double underPrice)**

Parameter	Description
reqId	The ticker ID.



contract	Describes the contract.
volatility	The volatility.
underPrice	Price of the underlying.

### **cancelCalculateOptionPrice()**

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

#### **cancelCalculateOptionPrice(TickerId reqId)**

Parameter	Description
reqId	The ticker id.

### **reqMarketDataType()**

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 marketDataType)**

Parameter	Description
marketDataType	1 for real-time streaming market data or 2 for frozen market data.

### **placeOrder()**

Call this function to place an order. The order status will be returned by the orderStatus event.

#### **void placeOrder( OrderID id, const Contract &contract, const Order &order)**

Parameter	Description
id	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	This structure contains a description of the contract which is being traded.
order	This structure contains the details of the order. Note: Each client MUST connect with a unique clientId.

### **cancelOrder()**

Call this function to cancel an order.

#### **void cancelOrder(OrderID id)**

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

### reqOpenOrders()

Call this function to request the open orders that were placed from this client. Each open order will be fed back through the openOrder() and orderStatus() functions on the EWrapper.

#### void reqOpenOrders()

**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.

### reqAllOpenOrders()

Call this function to request the open orders placed from all clients and also from TWS. Each open order will be fed back through the openOrder() and orderStatus() functions on the EWrapper.

#### void reqAllOpenOrders()

**Note:** No association is made between the returned orders and the requesting client.

### reqAutoOpenOrders()

Call this function 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 fed back through the openOrder() and orderStatus() functions on the EWrapper.

#### reqAutoOpenOrders (bool bAutoBind)

**Note:** This request can only be made from a client with clientId of 0.

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

### reqIDs()

Call this function to request from TWS the next valid ID that can be used when placing an order. After calling this function, 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).

#### void reqIds(int numIds)

Parameter	Description
numIds	The number of ids you want to reserve.

**exerciseOptions()**

**void exerciseOptions(long id, const Contract &contract, int exerciseAction, int exerciseQuantity, const CString &account, int override)**

Parameter	Description
id	The ticker id. Must be a unique value.
contract	This structure contains a description of the contract for which market depth data is being requested.
exerciseAction	Specifies whether you want the option to lapse or be exercised. Values are 1 = exercise, 2 = lapse.
exerciseQuantity	The quantity you want to exercise.
override	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 = no, 1 = yes.

**reqAccountUpdates()**

Call this function to start getting account values, portfolio, and last update time information.

**void reqAccountUpdates(bool subscribe, const CString& acctCode)**

Parameter	Description
subscribe	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	The account code for which to receive account and portfolio updates.

**reqExecutions()**

When this function is called, the execution reports that meet the filter criteria are downloaded to the client via the execDetails() function. 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(int reqID, const ExecutionFilter& filter)**

Parameter	Description
reqId	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
filter	This object contains attributes that describe the filter criteria used to determine which execution reports are returned.

**reqContractDetails()**

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

**void reqContractDetails (const Contract &contract)**

Parameter	Description
reqId	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
Contract	The summary description of the contract being looked up.

**reqMktDepth()**

Call this function to request market depth for a specific contract. The market depth will be returned by the updateMktDepth() and updateMktDepthL2() events.

**void reqMktDepth (TickerID id, const Contract &contract, long numRows)**

Parameter	Description
id	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	This structure contains a description of the contract for which market depth data is being requested.
numRows	- specified the number of market depth rows to display.

**cancelMktDepth()**

After calling this function, market depth data for the specified id will stop flowing.

**void cancelMktDepth (TickerID id)**

Parameter	Description
id	The ID that was specified in the call to reqMktDepth().

**reqNewsBulletins()**

Call this function to start receiving news bulletins. Each bulletin will be returned by the updatedNewsBulletin() event.

**void reqNewsBulletins(bool allMsgs)**

Parameter	Description
allMsgs	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.

**cancelNewsBulletins()**

Call this function to stop receiving news bulletins.

**void cancelNewsBulletins()**

**reqManagedAccts()**

Call this function to request the list of managed accounts. The list will be returned by the managedAccounts() function on the EWrapper.

**Note:** This request can only be made when connected to a FA managed account.

**void reqManagedAccts()**

**requestFA()**

Call this function to request FA configuration information from TWS. The data returns in an XML string via a "receiveFA" ActiveX event.

**requestFA(long faDataType)**

Parameter	Description
faDataType	<p>Specifies the type of Financial Advisor configuration data being requested. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 = ACCOUNT ALIASES</li> </ul>

**replaceFA()**

Call this function to modify FA configuration information from the API. Note that this can also be done manually in TWS itself.

**replaceFA(long faDataType, string XML)**

Parameter	Description
faDataType	<p>Specifies the type of Financial Advisor configuration data being modified via the API. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 = ACCOUNT ALIASES</li> </ul>
XML	The XML string containing the new FA configuration information.

## 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 an FA managed account.

**void reqAccountSummary(int messageType, int version, int reqId, String groupName, String tags)**

Parameter	Type	Description
messageType	Integer	Set this to 62.
version	Integer	Set this to 1.
reqId	Integer	
groupName	String	Set to <i>All</i> 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.

Parameter	Type	Description
tags	String	<p>A comma-separated list of account tags. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>NetLiquidation</i>,</li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i>,</li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i>,</li> <li>• <i>RegTMargin</i>,</li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i>,</li> <li>• <i>MaintMarginReq</i>,</li> <li>• <i>AvailableFunds</i>,</li> <li>• <i>ExcessLiquidity</i>,</li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i>,</li> <li>• <i>FullMaintMarginReq</i>,</li> <li>• <i>FullAvailableFunds</i>,</li> <li>• <i>FullExcessLiquidity</i>,</li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i>,</li> <li>• <i>LookAheadMaintMarginReq</i>,</li> <li>• <i>LookAheadAvailableFunds</i>,</li> <li>• <i>LookAheadExcessLiquidity</i>,</li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> </ul>

Parameter	Type	Description
		<ul style="list-style-type: none"> <li><i>DayTradesRemaining</i> — 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.</li> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>

### cancelAccountSummary()

Cancels the request for Account Window Summary tab data.

**void cancelAccountSummary(int messageId, int version, int reqId)**

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.

### reqPositions()

Requests real-time position data for all accounts.

**Note:** This request can only be made when connected to an FA managed account.

**void reqPositions(int messageId, int version)**

Parameter	Type	Description
messageId	Integer	Set this to 62
version	Integer	Set this to 1.

### cancelPositions()

Cancels real-time position updates.

**void cancelPositions(int messageId, int version)**

Parameter	Type	Description
messageId	Integer	Set this to 64.
version	Integer	Set this to 1.

### reqHistoricalData()

**void reqHistoricalData (TickerID id, const Contract &contract, String endDateTime, String durationStr, long barSizeSetting String whatToShow, int useRTH, int formatDate)**

Parameter	Description
-----------	-------------



id	The id of the request. 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	This object contains a description of the contract for which market data is being requested.
endDateTime	Defines a query end date and time at any point during the past 6 mos. Valid values include any date/time within the past six months in the format: yyyyymmdd HH:mm:ss ttt where "ttt" is the optional time zone.
durationStr	Set the query duration up to one week, using a time unit of seconds, days or weeks. Valid values include any integer followed by a space and then S (seconds), D (days) or W (week). If no unit is specified, seconds is used.
barSizeSetting	Specifies the size of the bars that will be returned (within IB/TWS limits). Valid values include: <ul style="list-style-type: none"> <li>• 1 sec</li> <li>• 5 secs</li> <li>• 15 secs</li> <li>• 30 secs</li> <li>• 1 min</li> <li>• 2 mins</li> <li>• 3 mins</li> <li>• 5 mins</li> <li>• 15 mins</li> <li>• 30 mins</li> <li>• 1 hour</li> <li>• 1 day</li> </ul>
whatToShow	Determines the nature of data being extracted. Valid values include: <ul style="list-style-type: none"> <li>• TRADES</li> <li>• MIDPOINT</li> <li>• BID</li> <li>• ASK</li> <li>• BID_ASK</li> <li>• HISTORICAL_VOLATILITY</li> <li>• OPTION IMPLIED_VOLATILITY</li> </ul>

useRTH	<p>Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 - all data is returned even where the market in question was outside of its regular trading hours.</li> <li>• 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.</li> </ul>
formatDate	<p>Determines the date format applied to returned bars. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 - dates applying to bars returned in the format: <code>yyyymmdd {space} {space} hh:mm:dd</code></li> <li>• 2 - dates are returned as a long integer specifying the number of seconds since 1/1/1970 GMT.</li> </ul>

For a information about historical data request limitations, see [Historical Data Limitations](#).

### **cancelHistoricalData()**

Used if an internet disconnect has occurred or the results of a query are otherwise delayed and the application is no longer interested in receiving the data.

**void cancelHistoricalData (int tickerId)**

Parameter	Description
tickerId	The ticker ID. Must be a unique value.

### **reqScannerParameters()**

Requests an XML string that describes all possible scanner queries.

**void reqScannerParameters()**

### **reqScannerSubscription()**

**void reqScannerSubscription(int tickerId, const ScannerSubscription&subscription)**

Parameter	Description
tickerId	The ticker ID. Must be a unique value.
ScannerSubscription	This structure contains possible parameters used to filter results.

### **cancelScannerSubscription()**

**void cancelScannerSubscription(int tickerId)**

Parameter	Description
tickerId	The ticker ID. Must be a unique value.

### **reqRealTimeBars()**

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

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

Parameter	Description
tickerId	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 request.
contract	This object contains a description of the contract for which real time bars are being requested
barSize	Currently only 5 second bars are supported, if any other value is used, an exception will be thrown.
whatToShow	Determines the nature of the data extracted. Valid values include: <ul style="list-style-type: none"> <li>• TRADES</li> <li>• BID</li> <li>• ASK</li> <li>• MIDPOINT</li> </ul>
useRTH	Regular Trading Hours only. Valid values include: <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>

### **cancelRealTimeBars()**

Call the cancelRealTimeBars() function to stop receiving real time bar results.

**void cancelRealTimeBars (int tickerId)**

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

### **reqFundamentalData()**

Call this function to receive Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.

**void reqFundamentalData(int reqId, const Contract &contract, String reportType)**

Parameter	Description
reqId	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contract	This structure contains a description of the contract for which Reuters Fundamental data is being requested.
reportType	Identifies the report type, which is one of the following: <ul style="list-style-type: none"><li>• Estimates</li><li>• Financial Statements</li><li>• Summary</li></ul>

### **cancelFundamentalData()**

Call this function to stop receiving Reuters global fundamental data.

**void cancelFundamentalData(int reqId)**

Parameter	Description
reqId	The ID of the data request.

## Class EWrapper Functions

The tables below define the class EWrapper functions you can use when connecting to TWS. These functions receive events from TWS. The list of functions includes:

<p><b>Connection and Server</b></p> <p><a href="#"><u>winError()</u></a>  <a href="#"><u>error()</u></a>  <a href="#"><u>connectionClosed()</u></a>  <a href="#"><u>currentTime()</u></a></p> <p><b>Market Data</b></p> <p><a href="#"><u>tickPrice()</u></a>  <a href="#"><u>tickSize()</u></a>  <a href="#"><u>tickOptionComputation()</u></a>  <a href="#"><u>tickGeneric()</u></a>  <a href="#"><u>tickString()</u></a>  <a href="#"><u>tickEFP()</u></a>  <a href="#"><u>tickSnapshotEnd()</u></a>  <a href="#"><u>marketDataType()</u></a></p> <p><b>Orders</b></p> <p><a href="#"><u>orderStatus()</u></a>  <a href="#"><u>openOrder()</u></a>  <a href="#"><u>nextValidId()</u></a></p> <p><b>Account and Portfolio</b></p> <p><a href="#"><u>updateAccountValue()</u></a>  <a href="#"><u>updatePortfolio()</u></a>  <a href="#"><u>updateAccountTime()</u></a></p> <p><b>News Bulletins</b></p> <p><a href="#"><u>updateNewsBulletin()</u></a></p>	<p><b>Contract Details</b></p> <p><a href="#"><u>contractDetails()</u></a>  <a href="#"><u>contractDetailsEnd()</u></a>  <a href="#"><u>bondContractDetails()</u></a></p> <p><b>Executions</b></p> <p><a href="#"><u>execDetails()</u></a>  <a href="#"><u>execDetailsEnd()</u></a>  <a href="#"><u>commissionReport()</u></a></p> <p><b>Market Depth</b></p> <p><a href="#"><u>updateMktDepth()</u></a>  <a href="#"><u>updateMktDepthL2()</u></a></p> <p><b>Financial Advisors</b></p> <p><a href="#"><u>managedAccounts()</u></a>  <a href="#"><u>receiveFA()</u></a>  <a href="#"><u>accountSummary()</u></a>  <a href="#"><u>accountSummaryEnd()</u></a>  <a href="#"><u>position()</u></a>  <a href="#"><u>positionEnd()</u></a></p> <p><b>Historical Data</b></p> <p><a href="#"><u>historicalData()</u></a></p> <p><b>Market Scanners</b></p> <p><a href="#"><u>scannerParameters()</u></a>  <a href="#"><u>scannerData()</u></a>  <a href="#"><u>scannerDataEnd()</u></a></p> <p><b>Real Time Bars</b></p> <p><a href="#"><u>realtimeBar()</u></a></p> <p><b>Fundamental Data</b></p> <p><a href="#"><u>fundamentalData()</u></a></p>
--	--

**winError()**

This event is called when there is an error on the client side.

**virtual void winError(const CString &str, int lastError)**

Parameter	Description
str	This is the error message text.
lastError	The error code returned by GetLastError().

**error()**

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

**virtual void error(const int id, const int errorCode, const CString errorString)**

Parameter	Description
id	This is the orderId or tickerId of the request that generated the error.
errorCode	Error codes are documented in the <a href="#">API Message Codes</a> topic.
errorString	This is the textual description of the error, also documented in the Error Codes topic.

**connectionClosed()**

This function is called when TWS closes the sockets connection with the ActiveX control, or when TWS is shut down.

**virtual void connectionClosed()**

**currentTime()**

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

**virtual void currentTime(long time)**

Parameter	Description
time	The current system time on the server side.

**tickPrice()**

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

**virtual void tickPrice(TickerID id, TickType field, double price, int canAutoExecute)**

Parameter	Description
-----------	-------------

id	The ticker ID that was specified previously in the call to reqMktData()
tickType	Specifies the type of price. Possible values are: <ul style="list-style-type: none"> <li>• 1 = bid</li> <li>• 2 = ask</li> <li>• 4 = last</li> <li>• 6 = high</li> <li>• 7 = low</li> <li>• 9 = close</li> </ul>
price	- could be the bid, ask, last price, daily high, daily low or last day close, depending on tickType value.
canAutoExecute	Specifies whether the price tick is available for automatic execution. Possible values are: <ul style="list-style-type: none"> <li>• 0 = not eligible for automatic execution</li> <li>• 1 = eligible for automatic execution</li> </ul>

### tickSize()

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

**virtual void tickSize(TickerID id, TickType field, int size)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktData()
tickType	Specifies the type of size. Possible values are: <ul style="list-style-type: none"> <li>• 0 = bid size</li> <li>• 3 = ask size</li> <li>• 5 = last size</li> <li>• 8 = volume</li> </ul>
size	Could be the bid size, ask size, last size or trading volume, depending on the tick-Type value.

### tickOptionComputation()

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

**virtual void tickOptionComputation(TickerID tickerId, TickType tickType, double impliedVol, double delta, double modelPrice, double pvDividend, double gamma, double vega, double theta, double undPrice)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktData()
tickType	Specifies the type of tick. Possible values are: <ul style="list-style-type: none"> <li>• 10 = Bid</li> <li>• 11 = Ask</li> <li>• 12 = Last</li> </ul>
impliedVol	The implied volatility calculated by the TWS option modeler, using the specified ticktype value.
delta	The option delta value.
optPrice	The option price.
pvDividend	The present value of dividends expected on the options underlying instrument.
gamma	The option gamma value.
vega	The option vega value.
theta	The option theta value.
undPrice	The price of the underlying.

### tickGeneric()

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

**virtual void tickGeneric(TickerId tickerId, TickType tickType, double value)**

Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData().
tickType	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	The value of the specified field.

### tickString()

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

**virtual void tickString(TickerId tickerId, TickType tickType, const CString& value)**



Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData().
field	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 45 will map to lastTimestamp, etc.
value	The value of the specified field.

**tickEFP()**

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

**virtual void tickEFP(TickerId tickerId, TickType tickType, double basisPoints, const CString& formattedBasisPoints, double totalDividends, int holdDays, const CString& futureExpiry, double dividendImpact, double dividendsToExpiry)**

Parameter	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	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	Annualized basis points, which is representative of the financing rate that can be directly compared to broker rates.
formattedBasisPoints	Annualized basis points as a formatted string that depicts them in percentage form.
impliedFuture	Implied futures price.
holdDays	Number of hold days until the expiry of the EFP.
futureExpiry	Expiration date of the single stock future.
dividendImpact	The dividend impact upon the annualized basis points interest rate.
dividendsToExpiry	The dividends expected until the expiration of the single stock future.

**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.

**virtual void tickSnapshotEnd(int reqId)**

Parameter	Description
-----------	-------------

reqID	Id of the data request.
-------	-------------------------

### 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.

#### virtual void marketDataType(TickerId reqId, int marketDataType)

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

### orderStatus()

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

#### virtual void orderStatus(OrderId id, const CString &status, int filled, int remaining, double avgFillPrice, int permId, int parentId, double lastFillPrice, int clientId, const CString& whyHeld)

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

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

status	<p>The order status. Possible values include:</p> <ul style="list-style-type: none"><li>• PendingSubmit - indicates that you have transmitted the order, but have not yet received confirmation that it has been accepted by the order destination. <b>NOTE:</b> This order status is not sent by TWS and should be explicitly set by the API developer when an order is submitted.</li><li>• 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. <b>NOTE:</b> This order status is not sent by TWS and should be explicitly set by the API developer when an order is canceled.</li><li>• 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.</li><li>• Submitted - indicates that your order has been accepted at the order destination and is working.</li><li>• 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.</li><li>• Filled - indicates that the order has been completely filled.</li><li>• 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.</li></ul>
--------	---

filled	Specifies the number of shares that have been executed.  For more information about partial fills, see <a href="#">Order Status for Partial Fills</a> .
remaining	Specifies the number of shares still outstanding.
avgFillPrice	The average price of the shares that have been executed. This parameter is valid only if the <b>filled</b> parameter value is greater than zero. Otherwise, the price parameter will be zero.
permId	The TWS id used to identify orders. Remains the same over TWS sessions.
parentId	The order ID of the parent order, used for bracket and auto trailing stop orders.
lastFilledPrice	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	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	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'.

### openOrder()

This function is called to feed in open orders.

**virtual void openOrder(OrderId orderId, const Contract &contract, const Order &order, const OrderState &orderState)**

For more information, see [Extended Order Attributes](#).

Parameter	Description
orderId	The order ID assigned by TWS. Use to cancel or update the order.
contract	The Contract class attributes describe the contract.
order	The Order class gives the details of the open order.
orderState	The orderState class includes attributes used for both pre and post trade margin and commission data.

### nextValidId()

This function is called after a successful connection to TWS.

**virtual void nextValidId(OrderID orderId)**

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

### updateAccountValue()

This function is called only when ReqAccountUpdates on EClientSocket object has been called.

**virtual void updateAccountValue(const CString& key, const CString& value, const CString& currency, const CString& accountName)**

Parameter	Description
key	<p>A string that indicates one type of account value. Below is a set of keys sent by TWS.</p> <ul style="list-style-type: none"> <li>• CashBalance - Account cash balance</li> <li>• Currency - Currency string</li> <li>• DayTradesRemaining - Number of day trades left</li> <li>• EquityWithLoanValue - Equity with Loan Value</li> <li>• InitMarginReq - Current initial margin requirement</li> <li>• LongOptionValue - Long option value</li> <li>• MaintMarginReq - Current maintenance margin</li> <li>• NetLiquidation - Net liquidation value</li> <li>• OptionMarketValue - Option market value</li> <li>• ShortOptionValue - Short option value</li> <li>• StockMarketValue - Stock market value</li> <li>• UnalteredInitMarginReq - Overnight initial margin requirement</li> <li>• UnalteredMaintMarginReq - Overnight maintenance margin requirement</li> </ul>
value	The value associated with the key.
currency	Defines the currency type, in case the value is a currency type.
account	States the account to which the message applies. Useful for Financial Advisor sub-account messages.

### updatePortfolio()

This function is called only when reqAccountUpdates on EClientSocket object has been called.

**virtual void updatePortfolio(const Contract& contract, int position, double marketPrice, double marketValue, double averageCost, double unrealizedPNL, double realizedPNL, const CString& accountName)**

Parameter	Description
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	This integer indicates the position on the contract. If the position is 0, it means the position has just cleared.
marketPrice	Unit price of the instrument.
marketValue	The total market value of the instrument.
averageCost	The average cost per share is calculated by dividing your cost (execution price + commission) by the quantity of your position.
unrealizedPNL	The difference between the current market value of your open positions and the average cost, or Value - Average Cost.
realizedPNL	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	States the account to which the message applies. Useful for Financial Advisor sub-account messages.

### updateAccountTime()

This function is called only when reqAccountUpdates on EClientSocket object has been called.

**virtual void updateAccountTime(const CString& timeStamp)**

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

### updateNewsBulletin()

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

**virtual void updateNewsBulletin(int msgId, int msgType, const CString& message, const CString& origExchange**

Parameter	Description
msgId	The bulletin ID, incrementing for each new bulletin.
msgType	Specifies the type of bulletin. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Regular news bulletin</li> <li>• 2 = Exchange no longer available for trading</li> <li>• 3 = Exchange is available for trading</li> </ul>

message	The bulletin's message text.
origExchange	The exchange from which this message originated.

### **contractDetails()**

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

#### **virtual void contractDetails(const ContractDetails &contractDetails)**

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

### **contractDetailsEnd()**

This function 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	Description
reqID	The ID of the data request.

### **bondContractDetails()**

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

#### **virtual void bondContractDetails(const contractDetails&contractDetails)**

Parameter	Description
reqId	The ID of the data request.
contractDetails	This structure contains a description of the contract which is being traded. The exchange field in a contract is not set for portfolio update.

### **execDetails()**

This event is fired when the reqExecutions() functions is invoked, or when an order is filled.

#### **virtual void execDetails( OrderId orderId, const Contract& contract, const Execution& execution, long liquidation)**

Parameter	Description
id	The order ID that was specified previously in the call to placeOrder().

contract	This structure contains a full description of the contract that was executed.
execution	This structure contains addition order execution details.

### **execDetailsEnd()**

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

**virtual void execDetailsEnd(int reqId)**

Parameter	Description
reqID	The Id of the data request.

### **commissionReport**

**virtual void commissionReport(const CommissionReport &commissionReport)**

Parameter	Description
commissionReport	The structure that contains commission details.

### **updateMktDepth()**

This function is called when the market depth changes.

**virtual void updateMktDepth(TickerId id, int position, int operation, int side, double price, int size)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktDepth()
position	Specifies the row id of this market depth entry.
operation	Identifies the how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>
side	Identifies the side of the book that this order belongs to. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>
price	The order price.
size	The order size.



**updateMktDepthL2()**

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

**virtual void updateMktDepthL2(TickerId id, int position, CString marketMaker, int operation, int side, double price, int size)**

Parameter	Description
id	The ticker ID that was specified previously in the call to reqMktDepth()
position	Specifies the row id of this market depth entry.
marketMaker	Specifies the exchange hosting this order.
operation	Identifies the how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>
side	Identifies the side of the book that this order belongs to. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>
price	The order price.
size	The order size.

**managedAccounts()**

This function is called when a successful connection is made to a Financial Advisor account. It is also called when the reqManagedAccts() function is invoked.

**virtual void managedAccounts(const CString& accountsList)**

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

**receiveFA()**

This event receives previously requested FA configuration information from TWS.

**virtual receiveFA(long faDataType, string XML)**

Parameter	Description
-----------	-------------

faDataType	Specifies the type of Financial Advisor configuration data being received from TWS. Valid values include: <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 =ACCOUNT ALIASES</li> </ul>
XML	The XML string containing the previously requested FA configuration information.

### **accountSummary()**

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

**virtual void accountSummary(int messageType, int version, int requestId, String account, String tag, String value, String currency)**

Parameter	Type	Description
messageType	Integer	Set to 62.
version	Integer	Set to 1.
requestId	Integer	The ID of the data request.
account	String	The account ID.

Parameter	Type	Description
tag	String	<p>The tag from the data request. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i></li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i></li> <li>• <i>RegTMargin</i></li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i></li> <li>• <i>MaintMarginReq</i></li> <li>• <i>AvailableFunds</i></li> <li>• <i>ExcessLiquidity</i></li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i></li> <li>• <i>FullMaintMarginReq</i></li> <li>• <i>FullAvailableFunds</i></li> <li>• <i>FullExcessLiquidity</i></li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i></li> <li>• <i>LookAheadMaintMarginReq</i></li> <li>• <i>LookAheadAvailableFunds</i></li> <li>• <i>LookAheadExcessLiquidity</i></li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> <li>• <i>DayTradesRemaining</i> — The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1"</li> </ul>

Parameter	Type	Description
		means that the user can put on unlimited day trades. <ul style="list-style-type: none"> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>
value	String	The value of the tag.
currency	String	The currency of the tag.

### accountSummaryEnd

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

**virtual void accountSummaryEnd(int reqId)**

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

### position()

This event returns real-time positions for all accounts in response to the [reqPositions\(\)](#) method.

**virtual void position(int messageId, int version, String account, int conid, String underlying, String securityType, String expiry, String strike, String right, String multiplier, String exchange, String currency, String ibLocalSymbol, double position)**

Parameter	Type	Description
messageId	Integer	Set to 62.
version	Integer	Set to 1.
account	String	The account.
conid	Integer	Unique contract identifier.
underlying	String	The symbol of the underlying asset.
securityType	String	The security type.
expiry	String	The expiration date.
strike	String	The strike price.
right	String	Put or call.
multiplier	String	The multiplier.
exchange	String	The exchange.
currency	String	The currency.
ibLocalSymbol	String	The local symbol.
position	double	The position.

### positionEnd()

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

**virtual void positionEnd(int reqId)**

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

**historicalData()**

This function receives the requested historical data results.

**virtual void historicalData(TickerId reqId, const CString& date, double open, double high, double low, double close, int volume, double WAP, int hasGaps)**

Parameter	Description
reqId	The ticker ID of the request to which this bar is responding.
date	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	The bar opening price.
high	The high price during the time covered by the bar.
low	The low price during the time covered by the bar.
close	The bar closing price.
volume	The volume during the time covered by the bar.
WAP	The weighted average price during the time covered by the bar.
count	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers.
hasGaps	Reports whether or not there are gaps in the data.

**scannerParameters()**

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

**virtual void scannerParameters(String xml)**

Parameter	Description
xml	An XML document that describes the valid parameters for queries.

**scannerData()**

This function receives the requested market scanner data results.

**virtual void scannerData(int reqId, int rank, const ContractDetails &contractDetails, String distance, String benchmark, String projection)**

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

### scannerDataEnd()

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

**virtual void scannerDataEnd(int reqId)**

Parameter	Description
reqId	The ID of the market scanner request being closed by this parameter.

### realtimeBar()

This function receives the real-time bars data results.

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

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

**fundamentalData()**

This function 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.

**virtual void fundamentalData(int reqId, string data)**

Parameter	Description
<b>reqId</b>	The ID of the data request.
<b>data</b>	One of three XML reports: <ul style="list-style-type: none"><li>• Estimates (estimates)</li><li>• Financial statements (finstat)</li><li>• Summary (snapshot)</li></ul>

## SocketClient Properties

The tables below define properties for the Execution, Contract and Order classes, and classes that are closely related to them.

- [Execution](#)
- [ExecutionFilter](#)
- [Contract](#)
- [ContractDetails](#)
- [ComboLeg](#)
- [Order](#)
- [OrderState](#)
- [ScannerSubscription](#)
- [UnderComp](#)
- [CommissionReport](#)

### Execution

Attribute	Description
IBString execId	Unique order execution id.
IBString time	The order execution time.
IBString acctNumber	The customer account number.
IBString exchange	Exchange that executed the order.
IBString side	Specifies if the transaction was a sale or a purchase. Valid values are: <ul style="list-style-type: none"> <li>• BOT</li> <li>• SLD</li> </ul>
int shares	The number of shares filled.
double price	The order execution price, not including commissions.
int permId	The TWS id used to identify orders, remains the same over TWS sessions.
long clientId	The id of the client that placed the order. <b>Note:</b> TWS orders have a fixed client id of 0.
long orderId	The order id. <b>Note:</b> TWS orders have a fixed order id of 0.



int liquidation	Identifies the position as one to be liquidated last should the need arise.
int cumQty	Cumulative quantity. Used in regular trades, combo trades and legs of the combo.
double avgPrice	Average price. Used in regular trades, combo trades and legs of the combo. Includes commissions.
IBString evRule	Contains the Economic Value Rule name and the respective optional argument. The two values should be separated by a colon. For example, aus-sieBond:YearsToExpiration=3. When the optional argument is not present, the first value will be followed by a colon.
double evMultiplier	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.

## ExecutionFilter

Attribute	Description
IBString acctCode	Filter the results of the reqExecutions() function based on an account code. Note: this is only relevant for FA managed accounts.
IBString exchange	Filter the results of the reqExecutions() function based on the order exchange.
IBString secType	Filter the results of the reqExecutions() function based on the order security type. Note: Refer to the Contract struct for the list of valid security types.
IBString side	Filter the results of the reqExecutions() function based on the order action. Note: Refer to the Order struct for the list of valid order actions.
IBString symbol	Filter the results of the reqExecutions() function based on the order symbol.
IBString time	Filter the results of the reqExecutions() function based on execution reports received after the specified time. The format for timeFilter is "yyyymmdd-hh:mm:ss"
long clientId	Filter the results of the reqExecutions() function based on the clientId.

## Contract

Attribute	Description
vector<ComboLeg> comboLegs	Dynamic memory structure used to store the leg definitions for this contract.
IBString comboLegsDescrip	Description for combo legs.
int conId	The unique contract identifier.
IBString currency	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.
IBString exchange	The order destination, such as Smart.
IBString expiry	The expiration date. Use the format YYYYMM.
bool includeExpired	<p>If set to true, contract details requests and historical data queries can be performed pertaining to expired contracts.</p> <p><b>Note:</b> 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.</p>
IBString localSymbol	This is the local exchange symbol of the underlying asset.
IBString multiplier	Allows you to specify a futures or options multiplier. This is only necessary when multiple possibilities exist.;
IBString primaryExchange	To clarify any ambiguity for Smart-routed contracts, include the primary exchange, along with the Smart designation, for the destination.
IBString right	Specifies a Put or Call. Valid values are: P, PUT, C, CALL.
IBString secId	Unique identifier for the secIdType.

IBString secIdType	<p>Security identifier, when querying contract details or when placing orders. Supported identifiers are:</p> <ul style="list-style-type: none"> <li>• ISIN (Example: Apple: US0378331005)</li> <li>• CUSIP (Example: Apple: 037833100)</li> <li>• SEDOL (Consists of 6-AN + check digit. Example: BAE: 0263494)</li> <li>• RIC (Consists of exchange-independent RIC Root and a suffix identifying the exchange. Example: AAPL.O for Apple on NASDAQ.)</li> </ul>
IBString secType	<p>This is the security type. Valid values are:</p> <ul style="list-style-type: none"> <li>• STK</li> <li>• OPT</li> <li>• FUT</li> <li>• IND</li> <li>• FOP</li> <li>• CASH</li> <li>• BAG</li> </ul>
double strike	The strike price.
IBString symbol	This is the symbol of the underlying asset.
IBString tradingClass	The trading class name for this contract.

### ContractDetails

Attribute	Description
bool callable	For Bonds. Values are True or False. If true, the bond can be called by the issuer under certain conditions.
IBString category	The industry category of the underlying. For example, InvestmentSvc.
IBString contractMonth	The contract month. Typically the contract month of the underlying for a futures contract.
bool convertible	For Bonds. Values are True or False. If true, the bond can be converted to stock under certain conditions.
double coupon	For Bonds. The interest rate used to calculate the amount you will receive in interest payments over the course of the year.

IBString industry	The industry classification of the underlying/product. For example, Financial.
IBString liquidHours	The liquid trading hours of the product. For example, 20090507:0930-1600;20090508:CLOSED.
IBString longName	The descriptive name of the asset.
IBString marketName	The market name for this contract.
double minTick	The minimum price tick.
Bool nextOptionPartial	For Bonds, relevant if the bond has embedded options, i.e., is the next option full or partial?
IBString orderTypes	The list of valid order type for this contract
long priceMagnifier	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.
bool putable	For Bonds. Values are True or False. If true, the bond can be sold back to the issuer under certain conditions.
TagValueListSPtr secIdList()	A list of contract identifiers that the customer is allowed to view (CUSIP, ISIN, etc.)
IBString subcategory	The industry subcategory of the underlying. For example, Brokerage.
Contract summary	A contract structure.
IBString tradingHours	The trading hours of the product. For example, 20090507:0700-1830,1830-2330;20090508:CLOSED.
IBString timeZoneId	The ID of the time zone for the trading hours of the product. For example, EST.
IBString underConId	The underlying contract ID.
IBString evRule	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.
double evMultiplier	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.
<b>Bond Values</b>	
IBString bondType	For Bonds. The type of bond, such as "CORP."
IBString couponType	For Bonds. The type of bond coupon, such as "FIXED."

IBString cusip	For Bonds. The nine-character bond CUSIP or the 12-character SEDOL.
IBString descAppend	For Bonds. A description string containing further descriptive information about the bond.
IBString issueDate	For Bonds. The date the bond was issued.
IBString maturity	For Bonds. The date on which the issuer must repay the face value of the bond.
IBString nextOptionDate	For Bonds, relevant if the bond has embedded options.
IBString nextOptionType	For Bonds, relevant if the bond has embedded options.
IBString notes	For Bonds, if populated for the bond in IB's database
IBString ratings	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.
IBString validExchanges	The list of exchanges on which this contract is traded.

### ComboLeg

Attribute	Description
IBString action	The side (buy or sell) for the leg you are constructing.
long conId	The unique contract identifier specifying the security.
IBString designatedLocation	If shortSaleSlot == 2, the designatedLocation must be specified. Otherwise leave blank or orders will be rejected.
IBString exchange	The exchange to which the complete combination order will be routed.
long openClose	Specifies whether the order is an open or close order. Valid values are: <ul style="list-style-type: none"> <li>• Same - (0) same as the parent security. This is the only option for retail customers.</li> <li>• Open - (1) only valid for institutional customers.</li> <li>• Close - (2) only valid for institutional customers.</li> <li>• Unknown</li> </ul>
long ratio	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.

int shortSaleSlot	<p>For institutional customers only.</p> <ul style="list-style-type: none"> <li>• 0 - inapplicable (i.e. retail customer or not short leg)</li> <li>• 1 - clearing broker</li> <li>• 2 - third party. If this value is used, you must enter a designated location.</li> </ul>
-------------------	---

## Order

Attribute	Description
<b>Order Identifiers</b>	
long clientId	The id of the client that placed this order.
long orderId	The id for this order.
long permId	The TWS id used to identify orders, remains the same over TWS sessions.
<b>Main Order Fields</b>	
IBString action	Identifies the side. Valid values are: BUY, SELL, SSHORT
double auxPrice	This is the STOP price for stop-limit orders, and the offset amount for relative orders. In all other cases, specify zero.
double lmtPrice	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.
IBString orderType	<p>Identifies the order type.</p> <p>For more information about supported order types, see <a href="#">Supported Order Types</a>.</p>
long totalQuantity	The order quantity.
<b>Extended Order Fields</b>	
bool allOrNone	0 = no, 1 = yes
bool blockOrder	If set to true, specifies that the order is an ISE Block order.
int displaySize	The publicly disclosed order size, used when placing Iceberg orders.
IBString goodAfterTime	<p>The trade's "Good After Time," format "YYYYMMDD hh:mm:ss (optional time zone)"</p> <p>Use an empty String if not applicable.</p>

IBString goodTillDate	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.
boolean hidden	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.
int minQty	Identifies a minimum quantity order type.
IBString ocaGroup	Identifies an OCA (one cancels all) group.
int ocaType	Tells how to handle remaining orders in an OCA group when one order or part of an order executes. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Cancel all remaining orders with block</li> <li>• 2 = Remaining orders are proportionately reduced in size with block</li> <li>• 3 = Remaining orders are proportionately reduced in size with no block</li> </ul> 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.
IBString orderRef	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.
boolean outsideRth()	If set to true, allows orders to also trigger or fill outside of regular trading hours.
bool overridePercentageConstraints	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 parameter's value to True. Valid values include: <ul style="list-style-type: none"> <li>• 0 = False</li> <li>• 1 = True</li> </ul>
long parentId	The order ID of the parent order, used for bracket and auto trailing stop orders.
double percentOffset	The percent offset amount for relative orders.

IBString rule80A	<p>Values include:</p> <ul style="list-style-type: none"> <li>• Individual = 'I'</li> <li>• Agency = 'A',</li> <li>• AgentOtherMember = 'W'</li> <li>• IndividualPTIA = 'J'</li> <li>• AgencyPTIA = 'U'</li> <li>• AgentOtherMemberPTIA = 'M'</li> <li>• IndividualPT = 'K'</li> <li>• AgencyPT = 'Y'</li> <li>• AgentOtherMemberPT = 'N'</li> </ul>
IBString tif	The time in force. Valid values are: DAY, GTC, IOC, GTD.
bool sweepToFill	If set to true, specifies that the order is a Sweep-to-Fill order.
double trailingPercent	<p>Specify the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field:</p> <ul style="list-style-type: none"> <li>• This field is mutually exclusive with the existing trailing amount. That is, the API client can send one or the other but not both.</li> <li>• This field is read AFTER the stop price (barrier price) as follows: deltaNeutralAuxPrice stopPrice trailingPercent scale order attributes</li> <li>• The field will also be sent to the API in the openOrder message if the API client version is <math>\geq 56</math>. It is sent after the stopPrice field as follows: stopPrice trailingPct basisPoint</li> </ul>
double trailStopPrice	For TRAILLIMIT orders only
bool transmit	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.



int triggerfunction	<p>Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 1 - use "double bid/ask" function, where stop orders are triggered based on two consecutive bid or ask prices.</li> <li>• 2 - "last" function, where stop orders are triggered based on the last price.</li> <li>• 3 double last function.</li> <li>• 4 bid/ask function.</li> <li>• 7 last or bid/ask function.</li> <li>• 8 mid-point function.</li> </ul>
<b>Financial Advisor Fields</b>	
IBString faGroup	The Financial Advisor group the trade will be allocated to -- use an empty String if not applicable.
IBString faMethod	The Financial Advisor allocation function the trade will be allocated with -- use an empty String if not applicable.
IBString faPercentage	The Financial Advisor percentage concerning the trade's allocation -- use an empty String if not applicable.
IBString faProfile	The Financial Advisor allocation profile the trade will be allocated to -- use an empty String if not applicable.
<b>Institutional (non-cleared) Only</b>	
IBString designatedLocation	Used only when shortSaleSlot = 2.
IBString openClose	For institutional customers only. Valid values are O, C.
int origin	The order origin. For institutional customers only. Valid values are 0 = customer, 1 = firm
int shortSaleSlot	Valid values are 1 or 2.
<b>SMART Routing Only</b>	
double discretionaryAmt	The amount off the limit price allowed for discretionary orders.
bool eTradeOnly	Trade with electronic quotes. 0 = no, 1 = yes
bool firmQuoteOnly	Trade with firm quotes. 0 = no, 1 = yes

double nbboPriceCap	Maximum smart order distance from the NBBO.
bool optOutSmartRouting	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.
<b>BOX or VOL Orders Only</b>	
int auctionStrategy	<p>Values include:</p> <ul style="list-style-type: none"> <li>• match = 1</li> <li>• improvement = 2</li> <li>• transparent = 3</li> </ul> <p>For orders on BOX only.</p>
<b>BOX Exchange Orders Only</b>	
double delta	The stock delta. For orders on BOX only.
double startingPrice	The auction starting price. For orders on BOX only.
double stockRefPrice	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.
<b>Pegged-to-Stock and VOL Orders Only</b>	
double stockRangeLower	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
double stockRangeUpper	The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
<b>Volatility Orders Only</b>	
bool continuousUpdate	VOL orders only. Specifies whether TWS will automatically update the limit price of the order as the underlying price moves.
int deltaNeutralAuxPrice	VOL orders only. Use this field to enter a value if the value in the <i>deltaNeutralOrderType</i> field is an order type that requires an Aux price, such as a REL order.
IBString deltaNeutralOrderType	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.

int referencePriceType	VOL orders only. Specifies how you want TWS to calculate the limit price for options, and for stock range price monitoring. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Average of NBBO</li> <li>• 2 = NBB or the NBO depending on the action and right.</li> </ul>
double volatility	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.
int volatilityType	Values include: <ul style="list-style-type: none"> <li>• 1 = Daily volatility</li> <li>• 2 = Annual volatility</li> </ul>
IBString deltaNeutralOpenClose	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.
bool deltaNeutralShortSale	Used when the hedge involves a stock and indicates whether or not it is sold short.
int deltaNeutralShortSaleSlot	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.
IBString deltaNeutralDesignatedLocation	Used only when deltaNeutralShortSaleSlot = 2.
<b>Combo Orders Only</b>	
double basisPoints	For EFP orders only
int basisPointsType	For EFP orders only
<b>Scale Orders Only</b>	
bool scaleAutoReset()	For extended Scale orders.
int scaleInitFillQty()	For extended Scale orders.
int scaleInitLevelSize	For Scale orders: Defines the size of the first, or initial, order component.

int scaleInitPosition()	For extended Scale orders.
int scalePriceAdjustInterval()	For extended Scale orders.
double scalePriceAdjustValue()	For extended Scale orders.
double scalePriceIncrement	For Scale orders: Defines the price increment between scale components. This field is required.
double scaleProfitOffset()	For extended Scale orders.
bool scaleRandomPercent()	For extended Scale orders.
int scaleSubsLevelSize	For Scale orders: Defines the order size of the subsequent scale order components. Used in conjunction with scaleInitLevelSize().
<b>Hedge Orders Only</b>	
IBString hedgeParam	Beta = x for Beta hedge orders, ratio = y for Pair hedge order
IBString hedgeType	For hedge orders. Possible values are: <ul style="list-style-type: none"> <li>• D = Delta</li> <li>• B = Beta</li> <li>• F = FX</li> <li>• P = Pair</li> </ul>
<b>Clearing Information</b>	
IBString account	The account. For institutional customers only.
IBString clearingAccount	For IBExecution customers: Specifies the true beneficiary of the order. This value is required for FUT/FOP orders for reporting to the exchange.
IBString clearingIntent	For IBExecution customers: Valid values are: IB, Away, and PTA (post trade allocation).
IBString settlingFirm	Institutional only.
<b>Algo Orders Only</b>	
IBString algoStrategy	For information about API Algo orders, see <a href="#">IBAlgo Parameters</a> .
Vector<TagValue> algoParams	Support for IBAlgo parameters.

<b>What If</b>	
bool whatIf	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.
<b>Order Combo Legs</b>	
Vector<TagValue> Order-ComboLegs	Holds attributes for all legs in a combo order.
<b>Not Held</b>	
bool m_notHeld	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.

## OrderState

Attribute	Description
double commission	Shows the commission amount on the order.
IBString commissionCurrency	Shows the currency of the commission value.
IBString equityWithLoan	Shows the impact the order would have on your equity with loan value.
IBString initMargin	Shows the impact the order would have on your initial margin.
IBString maintMargin	Shows the impact the order would have on your maintenance margin.
double maxCommission	Used in conjunction with the minCommission field, this defines the highest end of the possible range into which the actual order commission will fall.
double minCommission	Used in conjunction with the maxCommission field, this defines the lowest end of the possible range into which the actual order commission will fall.
IBString status	Displays the order status.
IBString warningText	Displays a warning message if warranted.

## ScannerSubscription

Attribute	Description
double abovePrice	Filter out contracts with a price lower than this value. Can be left blank.

int aboveVolume	Filter out contracts with a volume lower than this value. Can be left blank.
int averageOptionVolumeAbove	Can leave empty.
double belowPrice	Filter out contracts with a price higher than this value. Can be left blank.
double couponRateAbove	Filter out contracts with a coupon rate lower than this value. Can be left blank.
double couponRateBelow	Filter out contracts with a coupon rate higher than this value. Can be left blank.
IBString excludeConvertible	Filter out convertible bonds. Can be left blank.
IBString instrument	Defines the instrument type for the scan.
IBString locationCode	The location.
double marketCapAbove	Filter out contracts with a market cap lower than this value. Can be left blank.
double marketCapBelow	Filter out contracts with a market cap above this value. Can be left blank.
IBString maturityDateAbove	Filter out contracts with a maturity date earlier than this value. Can be left blank.
IBString maturityDateBelow	Filter out contracts with a maturity date later than this value. Can be left blank.
IBString moodyRatingAbove	Filter out contracts with a Moody rating below this value. Can be left blank.
IBString moodyRatingBelow	Filter out contracts with a Moody rating above this value. Can be left blank.
int numberOfRows	Defines the number of rows of data to return for a query.
IBString scanCode	Can be left blank.
IBString scannerSettingPairs	Can leave empty. For example, a pairing "Annual, true" used on the "top Option Implied Vol % Gainers" scan would return annualized volatilities.
IBString spRatingAbove	Filter out contracts with an S&P rating below this value. Can be left blank.
IBString spRatingBelow	Filter out contracts with an S&P rating above this value. Can be left blank.

IBString stockTypeFilter	Valid values are: <ul style="list-style-type: none"> <li>• CORP = Corporation</li> <li>• ADR = American Depositary Receipt</li> <li>• ETF = Exchange Traded Fund</li> <li>• REIT = Real Estate Investment Trust</li> <li>• CEF = Closed End Fund</li> </ul>
--------------------------	---

## UnderComp

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

## CommissionReport

Attribute	Description
double commission	The commission amount.
IBString currency()	The currency.
IBString execId()	Unique order execution id.
double realizedPNL()	The amount of realized Profit and Loss.
double yield()	The yield.
int yieldRedemptionDate()	Takes the YYYYMMDD format.

## 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 for a CLK9 futures contract and a SELL order for a CLM9 futures contract. In this procedure, the customer must invoke reqContractDetails() to obtain the conId for both CLK9 and CLM9 contracts.

#### Leg 1: Buy 1 CLK9 futures contract

#### Leg 2: Sell 1 CLM9 futures contract

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. Request 1 is assigned to CLK9 and Request 2 is assigned to CLM9.

```
con1.localSymbol = "CLK9";
con1.secType = "FUT";
con1.exchange = "NYMEX";
con1.currency = "USD";

m_client->reqContractDetails(1, con1->getContract()); // request 1

con2.m_localSymbol = "CLM9";
con2.m_secType = "FUT";
con2.m_exchange = "NYMEX";
con2.m_currency = "USD";

m_client->reqContractDetails(2, con2->getContract()); // request 2
```

The conId values are delivered by the following event. If reqId is equal to 1, then the conId is for the CLK9 contract. If reqId is equal to 2, then the conId is for CLM9.

```
::contractDetails( int reqId, const ContractDetails &contractDetails)
{
    // to obtain conId for CLK9
    if (reqId == 1)
    ...
}
```



```
// to obtain conid for CLM9
if (reqId == 2)
...
}
```

2. Assign all the related values for combo orders and combine them:

```
leg1.conId = Leg1_conId;
leg1.ratio = 1;
leg1.action = "BUY";
leg1.exchange = "NYMEX";
leg1.openClose = 0;
leg1.shortSaleSlot = 0;
leg1.designatedLocation = "";

leg2.conId = Leg2_conId;
leg2.ratio = 1;
leg2.action = "SELL";
leg2.exchange = "NYMEX";
leg2.openClose = 0;
leg2.shortSaleSlot = 0;
leg2.designatedLocation = "";
```

3. Invoke the placeOrder() method with the appropriate contract and order objects. As shown below, it includes the addAllLegs declaration in the contract object.

```
contract.symbol = "USD"; // arbitrary value only combo orders
contract.secType = "BAG"; // BAG is the security type for COMBO order
contract.exchange = "NYMEX";
contract.currency = "USD";
contract.comboLegs = addAllLegs; //including combo order in contract object

order.m_action = "BUY";
order.m_totalQuantity = 1;
order.m_orderType = "MKT";

m_client->placeOrder(Orderid, contract->getContract(), order->getOrder());
```



# Java

This chapter describes the Java API, including the following topics:

- [Linking to TWS using the Java API](#)
- [Running the Java Test Client Sample Program](#)
- [Running the Java Test Client Program with Eclipse](#)
- [Java Test Client Overview](#)
- [Java API Overview](#)
- [Java EClientSocket Methods](#)
- [Java EWrapper Methods](#)
- [Java SocketClient Properties](#)
- [Placing a Combination Order](#)
- [Java Code Samples: Contract Parameters](#)

## Linking to TWS using the Java API

### To link to TWS using the Java API

1. Import `com.ib.client.*` into your source code file.
2. Implement the `EWrapper` interface. This class will receive messages from the socket.
3. Override the following methods:

Ewrapper Method	Description
<code>tickPrice()</code>	Handles market data.
<code>tickSize()</code>	
<code>tickOptionComputation()</code>	
<code>tickGeneric()</code>	
<code>tickString()</code>	
<code>tickEFP()</code>	
<code>orderStatus()</code>	Receives order status.
<code>openOrder()</code>	Receives open orders.
<code>error()</code>	Receives error information.
<code>connectionClosed()</code>	Notifies when TWS terminates the connection.
<code>updateAccountValue()</code>	Receives current account values.
<code>updateAccountTime()</code>	Receives the last time account information was updated.
<code>updatePortfolio()</code>	Receives current portfolio information.
<code>nextValidId()</code>	Receives the next valid order ID upon connection.
<code>contractDetails()</code>	Receives contract information.
<code>contractDetailsEnd()</code>	Identifies the end of a given contract details request.

bondContractDetails()	Receives bond contract information.
execDetails()	Receives execution report information.
updateMktDepth()	Receives market depth information.
updateMktDepthL2()	Receives Level II market depth information.
updateNewsBulletin()	Receives IB news bulletins.
managedAccounts()	Receives a list of Financial Advisor (FA) managed accounts.
receiveFA()	Receives FA configuration information.
historicalData()	Receives historical data results.
scannerParameters()	Receives an XML document that describes the valid parameters of a scanner subscription.
scannerData()	Receives market scanner results.
realTimeBar()	Receives real-time bars.
currentTime()	Receives the current system time on the server.
fundamentalData()	Receives Reuters global fundamental market data.

4. Instantiate the EClientSocket class. This object will be used to send messages to TWS.
5. Call the following methods:

EClientSocket Method	Description
eConnect()	Connects to TWS.
eDisconnect()	Disconnects from TWS.
reqMktData()	Requests market data.
cancelMktData()	Cancels market data.
reqMktDepth()	Requests market depth.
cancelMktDepth()	Cancels market depth.
reqContractDetails()	Requests contract details.
placeOrder()	Places an order.
cancelOrder()	Cancels an order.
reqAccountUpdates()	Requests account values, portfolio, and account update time information.

reqExecutions()	Requests a list of the day's execution reports.
reqOpenOrders()	Requests a list of current open orders for the requesting client and associates TWS open orders with the client. The association only occurs if the requesting client has a Client ID of 0.
reqAllOpenOrders()	Requests a list of all open orders.
reqAutoOpenOrders()	Automatically associates a new TWS with the client. The association only occurs if the requesting client has a Client ID of 0.
reqNewsBulletin()	Requests IB news bulletins.
cancelNewsBulletins()	Cancels IB news bulletins.
setServerLogLevel()	Sets the level of API request and processing logging.
reqManagedAccts()	Requests a list of Financial Advisor (FA) managed account codes.
requestFA()	Requests FA configuration information from TWS.
replaceFA()	Modifies FA configuration information from the API.
reqScannerParameters()	Requests an XML document that describes the valid parameters of a scanner subscription.
reqScannerSubscription()	Requests market scanner results.
cancelScannerSubscription()	Cancels a scanner subscription.
reqHistoricalData()	Requests historical data.
cancelHistoricalData()	Cancels historical data.
reqRealTimeBars()	Requests real-time bars.
cancelRealTimeBars()	Cancels real-time bars.
exerciseOptions()	Exercises options.
reqCurrentTime()	Requests the current server time.
serverVersion()	Returns the version of the TWS instance to which the API application is connected.
TwConnectionTime()	Returns the time the API application made a connection to TWS.

reqFundamentalData()	Requests Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.
cancelFundamentalData()	Cancels Reuters global fundamental data.

## Running the Java Test Client Sample Program

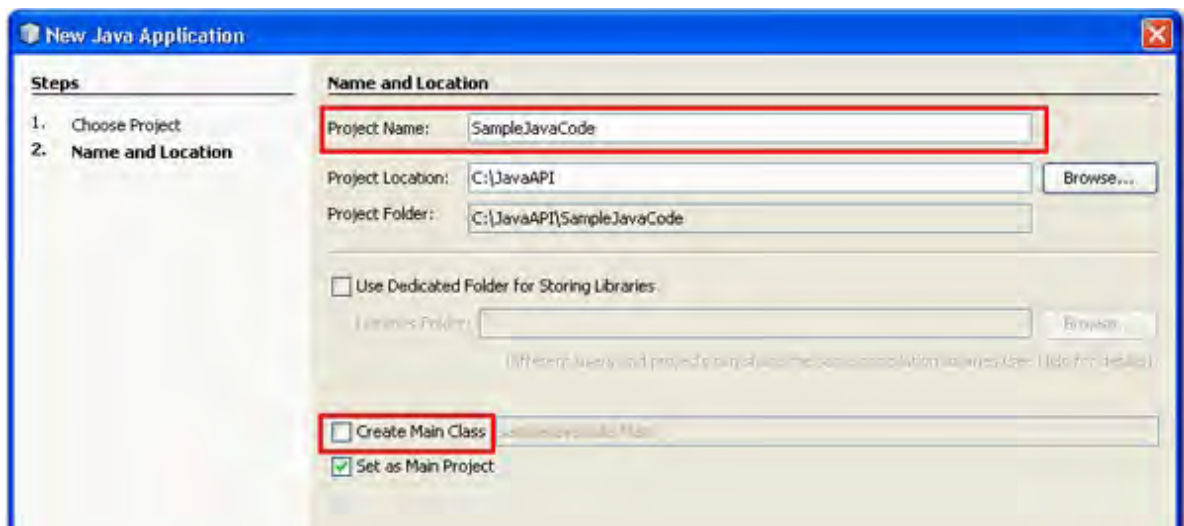
You can access the IB trading system via TWS or the IB Gateway through a Java application using the socket client component. Before you can connect to TWS, you must:

### To run the Java Test Client sample program on Windows

1. From Windows Explorer, navigate to the Jts:\Java folder.
2. Run the file *run.bat*.

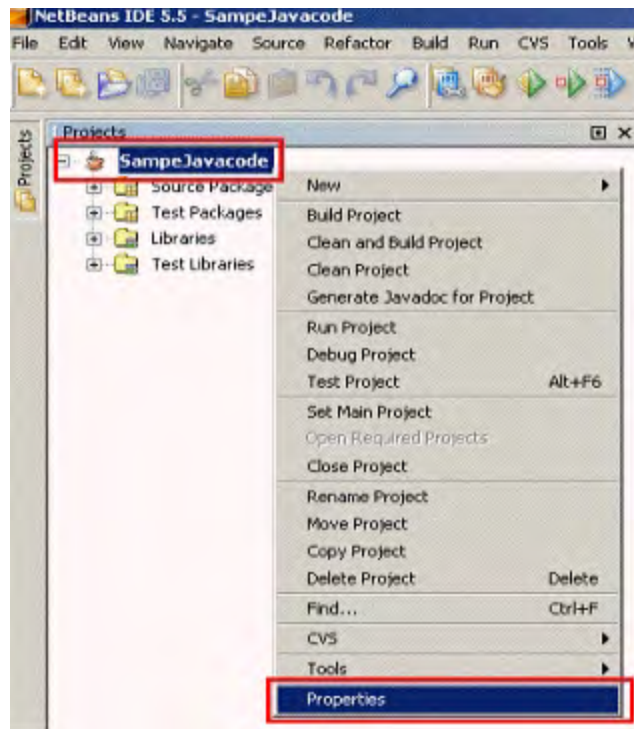
### To run the Java Test Client sample program from a new project in NetBeans

1. Open NetBeans and click *New Project* to start the wizard.
2. In the Projects area, select *Java Application* and click **Next**.
3. In the **New Java Application** window, name your project, choose a location, and uncheck the check box for *Create Main Class*.
4. Click **Finish**.

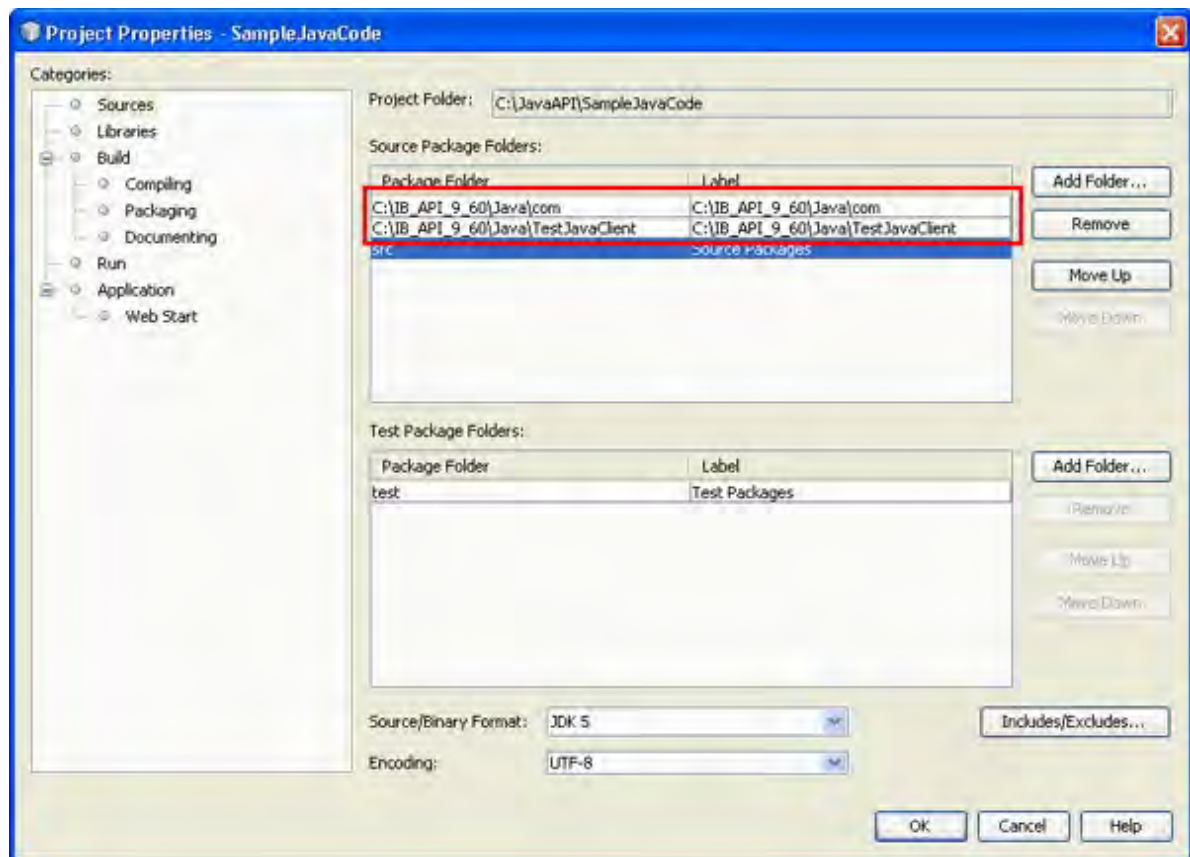




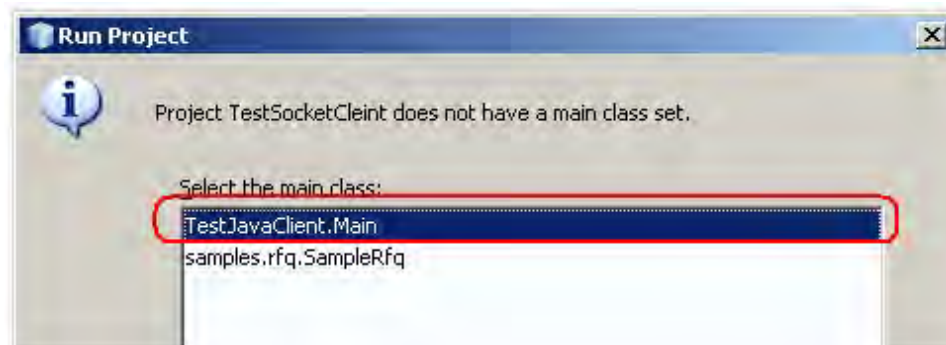
5. To set up Java to use the API, right-click the project you just created, then select **Properties**.



6. From the source category click *Add Folder*.
7. Navigate to the folder where the TWS API is installed. The folders you want to add are called *Java\com* and *Java\TestJavaClient*. Click **OK**.



8. Press **F6** to run the sample java project. When the message says "Project Samplejavacode does not have main class set" select *TestJavaclient.Main* and click **OK**.



The Java Test Client's sample application window is pictured below.



## Running the Java Test Client Program with Eclipse

This section describes how to run the Java Test Client Program with the Eclipse IDE. The following steps assume that you have already downloaded and installed the TWS API software.

**Note:** This procedure assumes that you are using Eclipse Helios 3.6.2, but other versions of Eclipse should also work.

### To run the Java Test Client Program with Eclipse

1. Download the Eclipse IDE from <http://www.eclipse.org/downloads/>.
  - Download the Windows 32-bit or Windows 64-bit version, depending on your operating system.
  - The download process will suggest the most appropriate mirror for your download automatically. Click on the link and save the zip file to your computer.
2. Unzip the downloaded Eclipse file.
3. Launch the Eclipse IDE by running the file **eclipse.exe**, which will be unzipped to the eclipse directory.
4. Start a new project:
  - a. To start a new project in Eclipse, select **File > New > Java Project**.
  - b. Type the project name. For this example, name the project *My API Program*.
  - c. Optionally, change various settings such as JRE and workspace location.
  - d. Click **Finish**.
5. Import the TWS API source files:
  - a. Expand the project you just created in the Package Explorer panel on the left.
  - b. Right-click the **src** folder, then select **New > Package**.
  - c. Enter *com.ib.client* as the package com, then click **Finish**.
  - d. Right-click the package **com.ib.client** in the Package Explorer panel, then select **Import**.
  - e. Select **General > File System**, then click **Next**.
  - f. Click **Browse...**, then locate the folder where the API is installed (typically Java\com\ib\client). Select the **client** folder (for example, C:\Jts\Java\com\ib\client\), then click **OK**.
  - g. Put a check mark on the **client** folder, then click **Finish**.
8. Import the Java sample test client files:
  - a. Expand the project you just created in the Package Explorer panel on the left.
  - b. Right-click the **src** folder, then select **New > Package**.
  - c. Enter *TestJavaClient* as the package com, then click **Finish**.

- d. Right-click the package **TestJavaClient** in the Package Explorer panel, then select **Import**.
  - e. Select **General > File System**, then click **Next**.
  - f. Click **Browse...**, then locate the folder where the Java Test Client is installed (typically Java\TestJavaClient). Select the **TestJavaClient** folder (for example, C:\Jts\Java\TestJavaClient), then click **OK**.
  - g. Put a check mark on the **client** folder, then click **Finish**.
8. Run the sample test client:
- a. Right-click the TestJavaClient package and select **Run As... > Java Application**.
  - b. This is what you should see and you are ready to create your own customized program:



## Java Test Client Overview

This section describes the package, classes and methods used in the Java Test Client, the sample Java program included in the TWS API.

**Note:** The classes and methods used in the Java Test Client were developed for the sample program and are not part of the TWS Java API.

### Package

The Java Test Client includes the package `TestJavaClient`. This package includes all the classes and methods required by the Java Test Client sample program.

### TestJavaClient Classes

`TestJavaClient` includes the following classes:

Class	Description
<code>AccountDlg</code>	Defines the Account/Portfolio dialog, which the Java Test Client sample program uses to display a customer's account and portfolio information.
<code>AcctUpdatesDlg</code>	Defines the Account Updates dialog, which the Java Test Client sample program uses to let FA customers subscribe to account updates.
<code>ComboLegDlg</code>	Defines the Combination Legs dialog, which the Java Test Client sample program uses to let customers add and remove combo legs.
<code>ConnectDlg</code>	Defines the Connect dialog, which the Java Test Client sample program uses to let customers connect to TWS
<code>ExecFilterDlg</code>	Defines the Execution Report Filter dialog, which the Java Test Client sample program uses to let customers enter filter criteria for execution reports.
<code>ExtOrdDlg</code>	Defines the Extended Order Info dialog, which the Java Test Client sample program uses to let customers enter values for extended order attributes.
<code>FAAllocationInfoDlg</code>	Defines the FA Allocation Info dialog, which the Java Test Client sample program uses to let Financial Advisor customers enter information about allocation profiles and account groups.
<code>FinancialAdvisorDlg</code>	Defines the Financial Advisor dialog, which Financial Advisor customers use in the Java Test Client sample program.

LogConfigDlg	Defines the Log Configuration dialog, which the Java Test Client sample program uses to let customers specify the level of log entries in the log file.
MktDepthDlg	Defines the Market Depth dialog, which the Java Test Client sample program uses to let customers view market depth for a specified contracts.
NewsBulletinDlg	Defines the IB News Bulletin Subscription dialog, which the Java Test Client sample program uses to let customers subscribe and unsubscribe to news bulletins.
OrderDlg	Defines the Sample dialog, which the Java Test Client sample program uses to let customers enter contract and order information for orders and requests for contract data, market depth, market data, options exercise, and historical data queries.
SampleFrame	Defines the Java Test Client sample program main window. All of the text panels and buttons on the main window are defined in this class.
ScannerDlg	Defines the Market Scanner dialog (also called the Sample dialog in the test client), which the Java Test Client sample program uses to let customers subscribe and unsubscribe to markets scans, as well as request market scan parameters.

**Note:** For more information about the Java code in the Java Test Client sample program, see the [Getting Started with the TWS Java API guide](#).

## Java API Overview

The TWS Java API contains the package **com.ib.client**, which contains the following classes:

Class	Description
<a href="#">EWrapper</a>	This interface is responsible for receiving messages from TWS.
<a href="#">ComboLeg</a>	This class contains attributes used to describe combo legs.
<a href="#">Contract</a>	This class contains attributes used to describe a contract.
<a href="#">ContractDetails</a>	This class contains attributes used to describe contract details, including bond information.
<a href="#">EClientSocket</a>	This class is responsible for sending messages to TWS.
<a href="#">Execution</a>	This class contains attributes used to describe a trade.

<a href="#"><u>ExecutionFilter</u></a>	This class contains attributes used to describe execution filter criteria.
<a href="#"><u>Order</u></a>	This class contains attributes used to describe an order.
<a href="#"><u>OrderState</u></a>	This class contains attributes used to describe the status of an order.
<a href="#"><u>ScannerSubscription</u></a>	This class contains attributes used to describe the elements of a market scan.
<a href="#"><u>TickType</u></a>	This class defines the generic tick types and their tick values.

The methods and attributes of these classes are described in the rest of this chapter.



## Java EClientSocket Methods

This section describes the class EClientSocket methods you use when connecting to TWS. The list of methods includes:

<p><b>Connection and Server</b></p> <p><a href="#"><u>EClientSocket()</u></a>  <a href="#"><u>eConnect()</u></a>  <a href="#"><u>eDisconnect()</u></a>  <a href="#"><u>isConnected()</u></a>  <a href="#"><u>setServerLogLevel()</u></a>  <a href="#"><u>reqCurrentTime()</u></a>  <a href="#"><u>serverVersion()</u></a>  <a href="#"><u>TwsConnectionTime()</u></a></p> <p><b>Market Data</b></p> <p><a href="#"><u>reqMktData()</u></a>  <a href="#"><u>cancelMktData()</u></a>  <a href="#"><u>calculateImpliedVolatility()</u></a>  <a href="#"><u>cancelCalculateImpliedVolatility()</u></a>  <a href="#"><u>calculateOptionPrice()</u></a>  <a href="#"><u>cancelCalculateOptionPrice()</u></a>  <a href="#"><u>reqMarketDataType()</u></a></p> <p><b>Orders</b></p> <p><a href="#"><u>placeOrder()</u></a>  <a href="#"><u>cancelOrder()</u></a>  <a href="#"><u>reqOpenOrders()</u></a>  <a href="#"><u>reqAllOpenOrders()</u></a>  <a href="#"><u>reqAutoOpenOrders()</u></a>  <a href="#"><u>reqIDs()</u></a>  <a href="#"><u>exerciseOptions()</u></a></p> <p><b>Account</b></p> <p><a href="#"><u>reqAccountUpdates()</u></a></p> <p><b>Executions</b></p> <p><a href="#"><u>reqExecutions()</u></a></p>	<p><b>Contract Details</b></p> <p><a href="#"><u>reqContractDetails()</u></a></p> <p><b>Market Depth</b></p> <p><a href="#"><u>reqMktDepth()</u></a>  <a href="#"><u>cancelMktDepth()</u></a></p> <p><b>News Bulletins</b></p> <p><a href="#"><u>reqNewsBulletins()</u></a>  <a href="#"><u>cancelNewsBulletins()</u></a></p> <p><b>Financial Advisors</b></p> <p><a href="#"><u>reqManagedAccts()</u></a>  <a href="#"><u>requestFA()</u></a>  <a href="#"><u>replaceFa()</u></a>  <a href="#"><u>reqAccountSummary()</u></a>  <a href="#"><u>cancelAccountSummary()</u></a>  <a href="#"><u>reqPositions()</u></a>  <a href="#"><u>cancelPositions()</u></a></p> <p><b>Market Scanners</b></p> <p><a href="#"><u>reqScannerParameters()</u></a>  <a href="#"><u>reqScannerSubscription()</u></a>  <a href="#"><u>cancelScannerSubscription()</u></a></p> <p><b>Historical Data</b></p> <p><a href="#"><u>reqHistoricalData()</u></a>  <a href="#"><u>cancelHistoricalData()</u></a></p> <p><b>Real Time Bars</b></p> <p><a href="#"><u>reqRealTimeBars()</u></a>  <a href="#"><u>cancelRealTimeBars()</u></a></p> <p><b>Fundamental Data</b></p> <p><a href="#"><u>reqFundamentalData()</u></a>  <a href="#"><u>cancelFundamentalData()</u></a></p>
---	---

**EClientSocket()**

This is the constructor.

**EClientSocket(AnyWrapper anyWrapper)**

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

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

**eDisconnect()**

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

**void eDisconnect()****isConnected()**

Call this method to check if there is a connection with TWS.

**void isConnected()****setServerLogLevel()**

The default level is ERROR. Refer to the [API logging](#) page for more details.

**void setServerLogLevel(int logLevel)**

Parameter	Description
-----------	-------------

logLevel	<p>Specifies the level of log entry detail used by the server (TWS) when processing API requests. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = SYSTEM</li> <li>• 2 = ERROR</li> <li>• 3 = WARNING</li> <li>• 4 = INFORMATION</li> <li>• 5 = DETAIL</li> </ul>
----------	---

### reqCurrentTime()

Returns the current system time on the server side via the [currentTime\(\)](#) EWrapper method.

**void reqCurrentTime()**

### serverVersion()

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

**void serverVersion()**

### TwsConnectionTime()

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

**void TwsConnectionTime ()**

### 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)**

Parameter	Description
tickerId	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.
<a href="#">contract</a>	This class contains attributes used to describe the contract.
genericTicklist	A comma delimited list of generic tick types. Tick types can be found in the <a href="#">Generic Tick Types</a> page.
snapshot	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.

### cancelMktData()

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

**void cancelMktData(int tickerId)**

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

**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	Description
reqId	The ticker id.
optionContract	Describes the contract.
optionPrice	The price of the option.
underPrice	Price of the underlying.

**cancelCalculateImpliedVolatility()**

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

**calculateImpliedVolatility(int reqId)**

Parameter	Description
reqId	The ticker id.

**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	Description
conid	The ticker ID.
contract	Describes the contract.
volatility	The volatility.
underPrice	Price of the underlying.

**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	Description
reqId	The ticker id.

### reqMarketDataType()

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

### placeOrder()

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

Parameter	Description
id	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	This class contains attributes used to describe the contract.
order	This structure contains the details of the order. Note: Each client MUST connect with a unique clientId.

### cancelOrder()

Call this method to cancel an order.

**void cancelOrder(int id)**

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

### 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()****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()****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	Description
bAutoBind	If set to TRUE, newly created TWS orders will be implicitly associated with the client. If set to FALSE, no association will be made.

**reqIDs()**

Call this function 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	Description
numIds	Set to 1.

**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	Description
-----------	-------------

tickerId	The Id for the exercise request
contract	This class contains attributes used to describe the contract.
exerciseAction	Specifies whether to exercise the specified option or let the option lapse. Valid values are: <ul style="list-style-type: none"> <li>• 1 = exercise</li> <li>• 2 = lapse</li> </ul>
exerciseQuantity	The number of contracts to be exercised
account	For institutional orders. Specifies the IB account.
override	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: <ul style="list-style-type: none"> <li>• 0 = do not override</li> <li>• 1 = override</li> </ul>

### 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	Description
subscribe	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	The account code for which to receive account and portfolio updates.

### 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)**

Parameter	Description
filter	The filter criteria used to determine which execution reports are returned.

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

**reqMktDepth()**

Call this method to request market depth for a specific contract. The market depth will be returned by the [updateMktDepth\(\)](#) and [updateMktDepthL2\(\)](#) methods.

**void reqMktDepth(int tickerId, Contract contract, int numRows)**

Parameter	Description
tickerId	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	This class contains attributes used to describe the contract.
numRows	Specifies the number of market depth rows to return.

**cancelMktDepth()**

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

**void cancelMktDepth(int id)**

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

**reqNewsBulletins()**

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

**void reqNewsBulletins(boolean allMsgs)**

Parameter	Description
allMsgs	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.



**cancelNewsBulletins()**

Call this method to stop receiving news bulletins.

**void cancelNewsBulletins()**

**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()**

**requestFA()**

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

**void requestFA(long faDataType)**

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being requested. Valid values include: <ul style="list-style-type: none"><li>• 1 = GROUPS</li><li>• 2 = PROFILE</li><li>• 3 = ACCOUNT ALIASES</li></ul>

**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(long faDataType, string xml)**

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being requested. Valid values include: <ul style="list-style-type: none"><li>• 1 = GROUPS</li><li>• 2 = PROFILE</li><li>• 3 = ACCOUNT ALIASES</li></ul>
xml	The XML string containing the new FA configuration information.

## 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 messageType, int version, int reqId, String groupName, String tags)**

Parameter	Type	Description
messageType	Integer	Set this to 62.
version	Integer	Set this to 1.
reqId	Integer	
groupName	String	Set to <i>All</i> 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.

Parameter	Type	Description
tags	String	<p>A comma-separated list of account tags. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>NetLiquidation</i>,</li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i>,</li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i>,</li> <li>• <i>RegTMargin</i>,</li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i>,</li> <li>• <i>MaintMarginReq</i>,</li> <li>• <i>AvailableFunds</i>,</li> <li>• <i>ExcessLiquidity</i>,</li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i>,</li> <li>• <i>FullMaintMarginReq</i>,</li> <li>• <i>FullAvailableFunds</i>,</li> <li>• <i>FullExcessLiquidity</i>,</li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i>,</li> <li>• <i>LookAheadMaintMarginReq</i>,</li> <li>• <i>LookAheadAvailableFunds</i>,</li> <li>• <i>LookAheadExcessLiquidity</i>,</li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> </ul>

Parameter	Type	Description
		<ul style="list-style-type: none"> <li><i>DayTradesRemaining</i> — 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.</li> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>

### 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.

### reqPositions()

Requests real-time position data for all accounts.

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

**void reqPositions(int messageId, int version)**

Parameter	Type	Description
messageId	Integer	Set this to 62
version	Integer	Set this to 1.

### cancelPositions()

Cancels real-time position updates.

**void cancelPositions(in messageId, int version)**

Parameter	Type	Description
messageId	Integer	Set this to 64.
version	Integer	Set this to 1.

### reqScannerParameters()

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

**void reqScannerParameters()**

**reqScannerSubscription()**

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

**void reqScannerSubscription(int tickerId, ScannerSubscription subscription)**

Parameter	Description
tickerId	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	Summary of the scanner subscription parameters including filters.

**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().

**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)**

Parameter	Description
tickerId	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	This class contains attributes used to describe the contract.
endDateTime	Use the format yyyyymmdd hh:mm:ss tmz, where the time zone is allowed (optionally) after a space at the end.

durationStr	<p>This is the time span the request will cover, and is specified using the format: &lt;integer&gt; &lt;unit&gt;, i.e., 1 D, where valid units are:</p> <ul style="list-style-type: none"> <li>• " S (seconds)</li> <li>• " D (days)</li> <li>• " W (weeks)</li> <li>• " M (months)</li> <li>• " Y (years)</li> </ul> <p>If no unit is specified, seconds are used. Also, note "years" is currently limited to one.</p>
barSizeSetting	<p>Specifies the size of the bars that will be returned (within IB/TWS limits). Valid bar size values include:</p> <ul style="list-style-type: none"> <li>• 1 sec</li> <li>• 5 secs</li> <li>• 15 secs</li> <li>• 30 secs</li> <li>• 1 min</li> <li>• 2 mins</li> <li>• 3 mins</li> <li>• 5 mins</li> <li>• 15 mins</li> <li>• 30 mins</li> <li>• 1 hour</li> <li>• 1 day</li> </ul>
whatToShow	<p>Determines the nature of data being extracted. Valid values include:</p> <ul style="list-style-type: none"> <li>• TRADES</li> <li>• MIDPOINT</li> <li>• BID</li> <li>• ASK</li> <li>• BID_ASK</li> <li>• HISTORICAL_VOLATILITY</li> <li>• OPTION_IMPLIED_VOLATILITY</li> </ul>

useRTH	<p>Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 - all data is returned even where the market in question was outside of its regular trading hours.</li> <li>• 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.</li> </ul>
formatDate	<p>Determines the date format applied to returned bars. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 - dates applying to bars returned in the format: yyymmdd{space}{space}hh:mm:dd</li> <li>• 2 - dates are returned as a long integer specifying the number of seconds since 1/1/1970 GMT.</li> </ul>

**Note:** For more information about historical data request limitations, see [Historical Data Limitations](#).

### cancelHistoricalData()

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

**void cancelHistoricalData (int tickerId)**

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

### 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)**

Parameter	Description
tickerId	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	This class contains attributes used to describe the contract.
barSize	Currently only 5 second bars are supported, if any other value is used, an exception will be thrown.

whatToShow	Determines the nature of the data extracted. Valid values include: <ul style="list-style-type: none"> <li>• TRADES</li> <li>• BID</li> <li>• ASK</li> <li>• MIDPOINT</li> </ul>
useRTH	Regular Trading Hours only. Valid values include: <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> </ul>

### cancelRealTimeBars()

Call this method to stop receiving real time bar results.

**void cancelRealTimeBars (int tickerId)**

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

### reqFundamentalData()

Call this method to receive Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.

**void reqFundamentalData(int reqId, Contract contract, string reportType)**

Parameter	Description
reqId	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.
contract	This structure contains a description of the contract for which Reuters Fundamental data is being requested.
reportType	Identifies the report type, which is one of the following: <ul style="list-style-type: none"> <li>• Estimates</li> <li>• Financial Statements</li> <li>• Summary</li> </ul>



**cancelFundamentalData()**

Call this method to stop receiving Reuters global fundamental data.

**void cancelFundamentalData(int reqId)**

Parameter	Description
reqId	The ID of the data request.

## Java EWrapper Methods

This section describes the class EWrapper methods you can use when connecting to TWS. The list of methods includes:

<b>Connection and Server</b> <a href="#"><u>currentTime()</u></a> <a href="#"><u>error()</u></a> <a href="#"><u>connectionClosed()</u></a>  <b>Market Data</b> <a href="#"><u>tickPrice()</u></a> <a href="#"><u>tickSize()</u></a> <a href="#"><u>tickOptionComputation()</u></a> <a href="#"><u>tickGeneric()</u></a> <a href="#"><u>tickString()</u></a> <a href="#"><u>tickEFP()</u></a> <a href="#"><u>tickSnapshotEnd()</u></a> <a href="#"><u>marketDataType()</u></a>  <b>Orders</b> <a href="#"><u>orderStatus()</u></a> <a href="#"><u>openOrder()</u></a> <a href="#"><u>nextValidId()</u></a>  <b>Account and Portfolio</b> <a href="#"><u>updateAccountValue()</u></a> <a href="#"><u>updatePortfolio()</u></a> <a href="#"><u>updateAccountTime()</u></a>  <b>Contract Details</b> <a href="#"><u>contractDetails()</u></a> <a href="#"><u>contractDetailsEnd()</u></a> <a href="#"><u>bondContractDetails()</u></a>	<b>Executions</b> <a href="#"><u>execDetails()</u></a> <a href="#"><u>execDetailsEnd()</u></a> <a href="#"><u>commissionReport()</u></a>  <b>Market Depth</b> <a href="#"><u>updateMktDepth()</u></a> <a href="#"><u>updateMktDepthL2()</u></a>  <b>News Bulletins</b> <a href="#"><u>updateNewsBulletin()</u></a>  <b>Financial Advisors</b> <a href="#"><u>managedAccounts()</u></a> <a href="#"><u>receiveFA()</u></a> <a href="#"><u>accountSummary()</u></a> <a href="#"><u>accountSummaryEnd()</u></a> <a href="#"><u>position()</u></a> <a href="#"><u>positionEnd()</u></a>  <b>Historical Data</b> <a href="#"><u>historicalData()</u></a>  <b>Market Scanners</b> <a href="#"><u>scannerParameters()</u></a> <a href="#"><u>scannerData()</u></a> <a href="#"><u>scannerDataEnd()</u></a>  <b>Real Time Bars</b> <a href="#"><u>realtimeBar()</u></a>  <b>Fundamental Data</b> <a href="#"><u>fundamentalData()</u></a>
--	---

### currentTime()

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

**void** currentTime(long time)

Parameter	Description
time	The current system time on the server side

**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	Description
id	This is the orderId or tickerId of the request that generated the error.
errorCode	For information on error codes, see <a href="#">Error Codes</a> .
errorString	The textual description of the error.

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

**void error(Exception e)**

Parameter	Description
e	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	Description
str	This is the textual description of the error

**connectionClosed()**

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

**void connectionClosed()****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	Description
-----------	-------------

tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	<p>Specifies the type of price. Pass the field value into <code>TickType.getField(int tickType)</code> to retrieve the field description. For example, a field value of 1 will map to <code>bidPrice</code>, a field value of 2 will map to <code>askPrice</code>, etc.</p> <ul style="list-style-type: none"> <li>• 1 = bid</li> <li>• 2 = ask</li> <li>• 4 = last</li> <li>• 6 = high</li> <li>• 7 = low</li> <li>• 9 = close</li> </ul>
price	Specifies the price for the specified field
canAutoExecute	<p>Specifies whether the price tick is available for automatic execution. Possible values are:</p> <ul style="list-style-type: none"> <li>• 0 = not eligible for automatic execution</li> <li>• 1 = eligible for automatic execution</li> </ul>

### 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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	<p>Specifies the type of price.</p> <p>Pass the field value into <code>TickType.getField(int tickType)</code> to retrieve the field description. For example, a field value of 0 will map to <code>bidSize</code>, a field value of 3 will map to <code>askSize</code>, etc.</p> <ul style="list-style-type: none"> <li>• 0 = bid size</li> <li>• 3 = ask size</li> <li>• 5 = last size</li> <li>• 8 = volume</li> </ul>
size	Specifies the size for the specified field

**tickOptionComputation()**

This method is called when the market in an option or its underlier moves. TWS's 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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	<p>Specifies the type of option computation.</p> <p>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.</p> <ul style="list-style-type: none"> <li>• 10 = Bid</li> <li>• 11 = Ask</li> <li>• 12 = Last</li> </ul>
impliedVol	The implied volatility calculated by the TWS option modeler, using the specified ticktype value.
delta	The option delta value.
optPrice	The option price.
pvDividend	The present value of dividends expected on the options underlier
gamma	The option gamma value.
vega	The option vega value.
theta	The option theta value.
undPrice	The price of the underlying.

**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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
tickType	<p>Specifies the type of price.</p> <p>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.</p>
value	The value of the specified field

**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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	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 45 will map to lastTimestamp, etc.
value	The value of the specified field

**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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktData()
field	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	Annualized basis points, which is representative of the financing rate that can be directly compared to broker rates
formattedBasisPoints	Annualized basis points as a formatted string that depicts them in percentage form
impliedFuture	Implied futures price
holdDays	The number of hold days until the expiry of the EFP
futureExpiry	The expiration date of the single stock future
dividendImpact	The dividend impact upon the annualized basis points interest rate
dividendsToExpiry	The dividends expected until the expiration of the single stock future

**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	Description
reqId	Id of the data request.

**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 type)**

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

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

status	<p>The order status. Possible values include:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 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.</li> <li>• 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 .</li> <li>• Submitted - indicates that your order has been accepted at the order destination and is working.</li> <li>• 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.</li> <li>• Filled - indicates that the order has been completely filled.</li> <li>• 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.</li> </ul>
filled	<p>Specifies the number of shares that have been executed.</p> <p>For more information about partial fills, see <a href="#">Order Status for Partial Fills</a>.</p>
remaining	Specifies the number of shares still outstanding.
avgFillPrice	The average price of the shares that have been executed. This parameter is valid only if the <b>filled</b> parameter value is greater than zero. Otherwise, the price parameter will be zero.
permId	The TWS id used to identify orders. Remains the same over TWS sessions.
parentId	The order ID of the parent order, used for bracket and auto trailing stop orders.
lastFilledPrice	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	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	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'.

### **openOrder()**

This method is called to feed in open orders.

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

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

### **nextValidId()**

This method is called after a successful connection to TWS.

**void nextValidId(int orderId)**

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

### **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	Description
-----------	-------------

key	<p>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:</p> <ul style="list-style-type: none"> <li>• CashBalance - account cash balance</li> <li>• DayTradesRemaining - number of day trades left</li> <li>• EquityWithLoanValue - equity with Loan Value</li> <li>• InitMarginReq - current initial margin requirement</li> <li>• MaintMarginReq - current maintenance margin</li> <li>• NetLiquidation - net liquidation value</li> </ul>
value	The value associated with the key.
currency	Defines the currency type, in case the value is a currency type.
account	States the account the message applies to. Useful for Financial Advisor sub-account messages.

### 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	Description
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	This integer indicates the position on the contract. If the position is 0, it means the position has just cleared.
marketPrice	The unit price of the instrument.
marketValue	The total market value of the instrument.
averageCost	The average cost per share is calculated by dividing your cost (execution price + commission) by the quantity of your position.
unrealizedPNL	The difference between the current market value of your open positions and the average cost, or Value - Average Cost.
realizedPNL	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	The name of the account the message applies to. Useful for Financial Advisor sub-account messages.

**updateAccountTime()**

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

**void updateAccountTime(String timeStamp)**

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

**contractDetails()**

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

**void contractDetails(int ReqId, ContractDetails contractDetails)**

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

**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	Description
reqID	The Id of the data request.

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

**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	Description
reqId	The reqID that was specified previously in the call to reqExecution().
contract	This structure contains a full description of the contract that was executed.  <b>Note:</b> Refer to the <a href="#">Java SocketClient Properties</a> page for more information.
execution	This structure contains addition order execution details.

### execDetailsEnd()

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

**void execDetailsEnd(int reqId)**

Parameter	Description
reqId	The Id of the data request.

### commissionReport()

**void commissionReport(CommissionReport commissionReport)**

Parameter	Description
commissionReport	The structure that contains commission details.

### 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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktDepth()
position	Specifies the row Id of this market depth entry.
operation	Identifies how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>

side	Identifies the side of the book that this order belongs to. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>
price	The order price.
size	The order size.

### **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	Description
tickerId	The ticker Id that was specified previously in the call to reqMktDepth()
position	Specifies the row id of this market depth entry.
marketMaker	Specifies the exchange hosting this order.
operation	Identifies the how this order should be applied to the market depth. Valid values are: <ul style="list-style-type: none"> <li>• 0 = insert (insert this new order into the row identified by 'position')</li> <li>• 1 = update (update the existing order in the row identified by 'position')</li> <li>• 2 = delete (delete the existing order at the row identified by 'position')</li> </ul>
side	Identifies the side of the book that this order belongs to. Valid values are: <ul style="list-style-type: none"> <li>• 0 = ask</li> <li>• 1 = bid</li> </ul>
price	The order price.
size	The order size.

### **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	Description
msgId	The bulletin ID, incrementing for each new bulletin.

msgType	Specifies the type of bulletin. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Regular news bulletin</li> <li>• 2 = Exchange no longer available for trading</li> <li>• 3 = Exchange is available for trading</li> </ul>
message	The bulletin's message text.
origExchange	The exchange from which this message originated.

### managedAccounts()

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

**void managedAccounts(String accountsList)**

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

### receiveFA()

This method receives previously requested FA configuration information from TWS.

**receiveFA(long faDataType, string xml)**

Parameter	Description
faDataType	Specifies the type of Financial Advisor configuration data being received from TWS. Valid values include: <ul style="list-style-type: none"> <li>• 1 = GROUPS</li> <li>• 2 = PROFILE</li> <li>• 3 = ACCOUNT ALIASES</li> </ul>
xml	The XML string containing the previously requested FA configuration information.

### accountSummary()

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

**void accountSummary(int messageType, int version, int requestId, String account, String tag, String value, String currency)**

Parameter	Type	Description
messageType	Integer	Set to 62.
version	Integer	Set to 1.

Parameter	Type	Description
requestId	Integer	The ID of the data request.
account	String	The account ID.

Parameter	Type	Description
tag	String	<p>The tag from the data request. Available tags are:</p> <ul style="list-style-type: none"> <li>• <i>AccountType</i></li> <li>• <i>TotalCashValue</i> — Total cash including futures pnl</li> <li>• <i>SettledCash</i> — For cash accounts, this is the same as Total-CashValue</li> <li>• <i>AccruedCash</i> — Net accrued interest</li> <li>• <i>BuyingPower</i> — The maximum amount of marginable US stocks the account can buy</li> <li>• <i>EquityWithLoanValue</i> — Cash + stocks + bonds + mutual funds</li> <li>• <i>PreviousEquityWithLoanValue</i></li> <li>• <i>GrossPositionValue</i> — The sum of the absolute value of all stock and equity option positions</li> <li>• <i>RegTEquity</i></li> <li>• <i>RegTMargin</i></li> <li>• <i>SMA</i> — Special Memorandum Account</li> <li>• <i>InitMarginReq</i></li> <li>• <i>MaintMarginReq</i></li> <li>• <i>AvailableFunds</i></li> <li>• <i>ExcessLiquidity</i></li> <li>• <i>Cushion</i> — Excess liquidity as a percentage of net liquidation value</li> <li>• <i>FullInitMarginReq</i></li> <li>• <i>FullMaintMarginReq</i></li> <li>• <i>FullAvailableFunds</i></li> <li>• <i>FullExcessLiquidity</i></li> <li>• <i>LookAheadNextChange</i> — Time when look-ahead values take effect</li> <li>• <i>LookAheadInitMarginReq</i></li> <li>• <i>LookAheadMaintMarginReq</i></li> <li>• <i>LookAheadAvailableFunds</i></li> <li>• <i>LookAheadExcessLiquidity</i></li> <li>• <i>HighestSeverity</i> — A measure of how close the account is to liquidation</li> <li>• <i>DayTradesRemaining</i> — The Number of Open/Close trades a user could put on before Pattern Day Trading is detected. A value of "-1"</li> </ul>



Parameter	Type	Description
		means that the user can put on unlimited day trades. <ul style="list-style-type: none"> <li><i>Leverage</i> — <math>\text{GrossPositionValue} / \text{NetLiquidation}</math></li> </ul>
value	String	The value of the tag.
currency	String	The currency of the tag.

### accountSummaryEnd

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

**void accountSummaryEnd(int reqId)**

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

### position()

This event returns real-time positions for all accounts in response to the [reqPositions\(\)](#) method.

**void position(int messageId, int version, String account, int conid, String underlying, String securityType, String expiry, String strike, String right, String multiplier, String exchange, String currency, String ibLocalSymbol, double position)**

Parameter	Type	Description
messageId	Integer	Set to 62.
version	Integer	Set to 1.
account	String	The account.
conid	Integer	Unique contract identifier.
underlying	String	The symbol of the underlying asset.
securityType	String	The security type.
expiry	String	The expiration date.
strike	String	The strike price.
right	String	Put or call.
multiplier	String	The multiplier.
exchange	String	The exchange.
currency	String	The currency.
ibLocalSymbol	String	The local symbol.
position	double	The position.

### 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(int reqId)**

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

**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	Description
reqId	The ticker Id of the request to which this bar is responding.
date	The date-time stamp of the start of the bar. The format is determined by the reqHistoricalData() formatDate parameter.
open	The bar opening price.
high	The high price during the time covered by the bar.
low	The low price during the time covered by the bar.
close	The bar closing price.
volume	The volume during the time covered by the bar.
count	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers
WAP	The weighted average price during the time covered by the bar.
hasGaps	Whether or not there are gaps in the data.

**scannerParameters()**

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

**void scannerParameters(String xml)**

Parameter	Description
xml	A document describing available scanner subscription parameters.

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

### **scannerDataEnd()**

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

**void scannerDataEnd(int reqId)**

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

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

**fundamentalData()**

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	Description
reqId	The ID of the data request.
data	One of three XML reports: <ul style="list-style-type: none"><li>• Estimates (estimates)</li><li>• Financial statements (finstat)</li><li>• Summary (snapshot)</li></ul>

## Java SocketClient Properties

The tables below define attributes for the following classes:

- [Execution](#)
- [ExecutionFilter](#)
- [CommissionReport](#)
- [Contract](#)
- [ContractDetails](#)
- [ComboLeg](#)
- [Order](#)
- [OrderState](#)
- [ScannerSubscription](#)
- [UnderComp](#)

### Execution

Attribute	Description
String m_acctNumber	The customer account number.
double m_avgPrice	Average price. Used in regular trades, combo trades and legs of the combo. Includes commissions.
int m_clientId	The id of the client that placed the order. Note: TWS orders have a fixed client id of "0."
int m_cumQty	Cumulative quantity. Used in regular trades, combo trades and legs of the combo.
String m_exchange	Exchange that executed the order.
String m_execId	Unique order execution id.
int m_liquidation	Identifies the position as one to be liquidated last should the need arise.
int m_orderId	The order id.  <b>Note:</b> TWS orders have a fixed order id of "0."
int m_permId	The TWS id used to identify orders, remains the same over TWS sessions.
double m_price	The order execution price, not including commissions.

int m_shares	The number of shares filled.
String m_side	Specifies if the transaction was a sale or a purchase. Valid values are: <ul style="list-style-type: none"> <li>• BOT</li> <li>• SLD</li> </ul>
String m_time	The order execution time.
String m_evRule	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.
double m_evMultiplier	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.

## ExecutionFilter

Attribute	Description
String m_acctCode	Filter the results of the reqExecutions() method based on an account code. Note: this is only relevant for Financial Advisor (FA) accounts.
int m_clientId	Filter the results of the reqExecutions() method based on the clientId.
String m_exchange	Filter the results of the reqExecutions() method based on the order exchange.
String m_secType	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.
String m_side	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.
String m_symbol	Filter the results of the reqExecutions() method based on the order symbol.
String m_time	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"

## CommissionReport

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

## Contract

Attribute	Description
Vector m_comboLegs	Dynamic memory structure used to store the leg definitions for this contract.
String m_comboLegsDescrip	Description for combo legs
int m_conId	The unique contract identifier.
String m_currency	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.
String m_exchange	The order destination, such as Smart.
String m_expiry	The expiration date. Use the format YYYYMM.
boolean m_includeExpired	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,
String m_localSymbol	This is the local exchange symbol of the underlying asset.
String m_multiplier	Allows you to specify a future or option contract multiplier. This is only necessary when multiple possibilities exist.
String m_primaryExch	Identifies the listing exchange for the contract (do not list SMART).

String m_right	Specifies a Put or Call. Valid values are: P, PUT, C, CALL.
String m_secId	Unique identifier for the secIdType.
String m_secIdType	Security identifier, when querying contract details or when placing orders. Supported identifiers are: <ul style="list-style-type: none"> <li>• SIN (Example: Apple: US0378331005)</li> <li>• CUSIP (Example: Apple: 037833100)</li> <li>• SEDOL (Consists of 6-AN + check digit. Example: BAE: 0263494)</li> <li>• RIC (Consists of exchange-independent RIC Root and a suffix identifying the exchange. Example: AAPL.O for Apple on NASDAQ.)</li> </ul>
String m_secType	This is the security type. Valid values are: <ul style="list-style-type: none"> <li>• STK</li> <li>• OPT</li> <li>• FUT</li> <li>• IND</li> <li>• FOP</li> <li>• CASH</li> <li>• BAG</li> </ul>
double m_strike	The strike price.
String m_symbol	This is the symbol of the underlying asset.
String m_tradingClass	The trading class name for this contract.

### ContractDetails

Attribute	Description
String m_category	The industry category of the underlying. For example, InvestmentSvc.
String m_contractMonth	The contract month. Typically the contract month of the underlying for a futures contract.
String m_industry	The industry classification of the underlying/product. For example, Financial.
String m_liquidHours	The liquid trading hours of the product. For example, 20090507:0930-1600;20090508:CLOSED.
String m_longName	Descriptive name of the asset.



String m_marketName	The market name for this contract.
double m_minTick	The minimum price tick.
String m_orderTypes	The list of valid order types for this contract.
String m_priceMagnifier	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.
Vector<TagValue> m_secId-List()	A list of contract identifiers that the customer is allowed to view (CUSIP, ISIN, etc.)
String m_subcategory	The industry subcategory of the underlying. For example, Brokerage.
Contract m_summary	A contract summary.
String m_timeZoneId	The ID of the time zone for the trading hours of the product. For example, EST.
String m_tradingHours	The trading hours of the product. For example, 20090507:0700-1830,1830-2330;20090508:CLOSED.
String m_underConId	The underlying contract ID.
String m_validExchanges	The list of exchanges this contract is traded on.
String m_evRule	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.
double m_evMultiplier	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.
<b>Bond Values</b>	
String m_bondType	For Bonds. The type of bond, such as "CORP."
boolean m_callable	For Bonds. Values are True or False. If true, the bond can be called by the issuer under certain conditions.
boolean m_convertible	For Bonds. Values are True or False. If true, the bond can be converted to stock under certain conditions.
double m_coupon	For Bonds. The interest rate used to calculate the amount you will receive in interest payments over the course of the year.
String m_couponType	For Bonds. The type of bond coupon.

String m_cusip	For Bonds. The nine-character bond CUSIP or the 12-character SEDOL.
String m_descAppend	For Bonds. A description string containing further descriptive information about the bond.
String m_issueDate	For Bonds. The date the bond was issued.
String m_maturity	For Bonds. The date on which the issuer must repay the face value of the bond.
String m_nextOptionDate	For Bonds, only if bond has embedded options.
boolean m_nextOptionPartial	For Bonds, only if bond has embedded options.
boolean m_putable	For Bonds. Values are True or False. If true, the bond can be sold back to the issuer under certain conditions.
String m_ratings	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.
String m_nextOptionType	For Bonds, only if bond has embedded options.
String m_notes	For Bonds, if populated for the bond in IB's database

### ComboLeg

Attribute	Description
String m_action	The side (buy or sell) for the leg you are constructing.
int m_conId	The unique contract identifier specifying the security.
String m_designatedLocation	If shortSaleSlot == 2, the designatedLocation must be specified. Otherwise leave blank or orders will be rejected.
String m_exchange	The exchange to which the complete combination order will be routed.
int m_openClose	Specifies whether the order is an open or close order. Valid values are: <ul style="list-style-type: none"> <li>• 0 - Same as the parent security. This is the only option for retail customers.</li> <li>• 1 - Open. This value is only valid for institutional customers.</li> <li>• 2 - Close. This value is only valid for institutional customers.</li> <li>• Unknown - (3)</li> </ul>

int m_ratio	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.
int m_shortSaleSlot	For institutional customers only. <ul style="list-style-type: none"> <li>• 0 - inapplicable (i.e. retail customer or not short leg)</li> <li>• 1 - clearing broker</li> <li>• 2 - third party. If this value is used, you must enter a designated location.</li> </ul>

### OrderComboLeg

Attribute	Description
double m_price	Order-specific leg price.

### Order

Attribute	Description
<b>Order Identifiers</b>	
int m_clientId	The id of the client that placed this order.
int m_orderId	The id for this order.
int m_permid	The TWS id used to identify orders, remains the same over TWS sessions.
<b>Main Order Fields</b>	
String m_action	Identifies the side. Valid values are: BUY, SELL, SSHORT
double m_auxPrice	This is the STOP price for stop-limit orders, and the offset amount for relative orders. In all other cases, specify zero.
double m_lmtPrice	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.
String m_orderType	Identifies the order type.  For more information about supported order types, see <a href="#">Supported Order Types</a> .
long m_totalQuantity	The order quantity.
<b>Extended Order Fields</b>	

boolean m_allOrNone	0 = no, 1 = yes
boolean m_blockOrder	If set to true, specifies that the order is an ISE Block order.
int m_displaySize	The publicly disclosed order size, used when placing Iceberg orders.
String m_goodAfterTime	The trade's "Good After Time," format "YYYYMMDD hh:mm:ss (optional time zone)" Use an empty String if not applicable.
String m_goodTillDate	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.
boolean hidden	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.
int m_minQty	Identifies a minimum quantity order type.
String m_ocaGroup	Identifies an OCA (one cancels all) group.
int m_ocaType	Tells how to handle remaining orders in an OCA group when one order or part of an order executes. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Cancel all remaining orders with block</li> <li>• 2 = Remaining orders are proportionately reduced in size with block</li> <li>• 3 = Remaining orders are proportionately reduced in size with no block</li> </ul> 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.
String m_orderRef	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.
boolean m_outsideRth	If set to true, allows orders to also trigger or fill outside of regular trading hours.
int m_parentId	The order ID of the parent order, used for bracket and auto trailing stop orders.
double m_percentOffset	The percent offset amount for relative orders.

boolean overridePercentageConstraints	<p>Precautionary constraints are defined on the TWS Pre-sets 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 parameter's value to True.</p> <p>Valid values include:</p> <ul style="list-style-type: none"> <li>• 0 = False</li> <li>• 1 = True</li> </ul>
string m_rule80A	<p>Values include:</p> <ul style="list-style-type: none"> <li>• Individual = 'I'</li> <li>• Agency = 'A',</li> <li>• AgentOtherMember = 'W'</li> <li>• IndividualPTIA = 'J'</li> <li>• AgencyPTIA = 'U'</li> <li>• AgentOtherMemberPTIA = 'M'</li> <li>• IndividualPT = 'K'</li> <li>• AgencyPT = 'Y'</li> <li>• AgentOtherMemberPT = 'N'</li> </ul>
boolean m_sweepToFill	If set to true, specifies that the order is a Sweep-to-Fill order.
String m_tif	The time in force. Valid values are: DAY, GTC, IOC, GTD.
bool m_transmit	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.

int m_triggerMethod	<p>Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are:</p> <ul style="list-style-type: none"> <li>• 0 - The default value. The "double bid/ask" function will be used for orders for OTC stocks and US options. All other orders will use the "last" function.</li> <li>• 1 - use "double bid/ask" function, where stop orders are triggered based on two consecutive bid or ask prices.</li> <li>• 2 - "last" function, where stop orders are triggered based on the last price.</li> <li>• 3 double last function.</li> <li>• 4 bid/ask function.</li> <li>• 7 last or bid/ask function.</li> <li>• 8 mid-point function.</li> </ul>
double m_trailStopPrice	For TRAILLIMIT orders only
double m_trailingPercent	<p>Specify the trailing amount of a trailing stop order as a percentage. Observe the following guidelines when using the trailingPercent field:</p> <ul style="list-style-type: none"> <li>• This field is mutually exclusive with the existing trailing amount. That is, the API client can send one or the other but not both.</li> <li>• This field is read AFTER the stop price (barrier price) as follows: deltaNeutralAuxPrice stopPrice trailingPercent scale order attributes</li> <li>• The field will also be sent to the API in the openOrder message if the API client version is <math>\geq 56</math>. It is sent after the stopPrice field as follows: stopPrice trailingPct basisPoint</li> </ul>
<b>Financial Advisor Fields</b>	
String m_faGroup	The Financial Advisor group the trade will be allocated to -- use an empty String if not applicable.
String m_faMethod	The Financial Advisor allocation function the trade will be allocated with -- use an empty String if not applicable.

String m_faPercentage	The Financial Advisor percentage concerning the trade's allocation -- use an empty String if not applicable.
String m_faProfile	The Financial Advisor allocation profile the trade will be allocated to -- use an empty String if not applicable.
<b>Institutional (non-cleared) Only</b>	
String m_designatedLocation	Used only when shortSaleSlot = 2.
String m_openClose	For institutional customers only. Valid values are O, C.
int m_origin	The order origin. For institutional customers only. Valid values are 0 = customer, 1 = firm
int m_shortSaleSlot	Valid values are 1 or 2.
<b>SMART Routing Only</b>	
double m_discretionaryAmt	The amount off the limit price allowed for discretionary orders.
boolean m_eTradeOnly	Trade with electronic quotes. 0 = no, 1 = yes
boolean m_firmQuoteOnly	Trade with firm quotes. 0 = no, 1 = yes
double m_nbboPriceCap	Maximum smart order distance from the NBBO.
boolean m_optOutSmartRouting	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.
<b>BOX or VOL Orders Only</b>	
int m_auctionStrategy	Values include: <ul style="list-style-type: none"> <li>• match = 1</li> <li>• improvement = 2</li> <li>• transparent = 3</li> </ul> For orders on BOX only.
<b>BOX Exchange Orders Only</b>	
double m_delta	The stock delta. For orders on BOX only.
double m_startingPrice	The auction starting price. For orders on BOX only.

double m_stockRefPrice	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.
<b>Pegged-to-Stock and VOL Orders Only</b>	
double m_stockRangeLower	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
double m_stockRangeUpper	The upper value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
<b>Volatility Orders Only</b>	
boolean m_continuousUpdate	VOL orders only. Specifies whether TWS will automatically update the limit price of the order as the underlying price moves.
String m_deltaNeutralOrderType	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.
int m_deltaNeutralAuxPrice	VOL orders only. Use this field to enter a value if the value in the <i>deltaNeutralOrderType</i> field is an order type that requires an Aux price, such as a REL order.
int m_referencePriceType	VOL orders only. Specifies how you want TWS to calculate the limit price for options, and for stock range price monitoring. Valid values include: <ul style="list-style-type: none"> <li>• 1 = Average of NBBO</li> <li>• 2 = NBB or the NBO depending on the action and right.</li> </ul>
double m_volatility	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.
int m_volatilityType	Values include: <ul style="list-style-type: none"> <li>• 1 = Daily volatility</li> <li>• 2 = Annual volatility</li> </ul>



String m_deltaNeutralOpenClose	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.
boolean m_deltaNeutralShortSale	Used when the hedge involves a stock and indicates whether or not it is sold short.
int m_deltaNeutralShortSaleSlot	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.
String m_deltaNeutralDesignatedLocation	Used only when deltaNeutralShortSaleSlot = 2.
<b>Combo Orders Only</b>	
double m_basisPoints	For EFP orders only
int m_basisPointsType	For EFP orders only
<b>Scale Orders Only</b>	
boolean m_scaleAutoReset()	For extended Scale orders.
int m_scaleInitFillQty()	For extended Scale orders.
int m_scaleInitLevelSize	For Scale orders: Defines the size of the first, or initial, order component.
int m_scaleInitPosition()	For extended Scale orders.
int m_scalePriceAdjustInterval()	For extended Scale orders.
double m_scalePriceAdjustValue()	For extended Scale orders.
double m_scalePriceIncrement	For Scale orders: Defines the price increment between scale components. This field is required.
double m_scaleProfitOffset()	For extended Scale orders.
boolean m_scaleRandomPercent()	For extended Scale orders.
int m_scaleSubsLevelSize	For Scale orders: Defines the order size of the subsequent scale order components. Used in conjunction with scaleInitLevelSize().
<b>Hedge Orders Only</b>	

String m_hedgeParam	Beta = x for Beta hedge orders, ratio = y for Pair hedge order
String m_hedgeType	For hedge orders. Possible values are: <ul style="list-style-type: none"> <li>• D = Delta</li> <li>• B = Beta</li> <li>• F = FX</li> <li>• P = Pair</li> </ul>
<b>Clearing Information</b>	
String m_account	The account. For institutional customers only.
String m_clearingAccount	For IBExecution customers: Specifies the true beneficiary of the order. This value is required for FUT/FOP orders for reporting to the exchange.
String m_clearingIntent	For IBExecution customers: Valid values are: IB, Away, and PTA (post trade allocation).
String m_settlingFirm	Institutional only.
<b>Algo Orders Only</b>	
String m_algoStrategy	For information about API Algo orders, see <a href="#">IBAlgo Parameters</a> .
Vector<TagVlue> m_algoParams	Support for IBAlgo parameters.
<b>What If</b>	
boolean m_whatIf	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() call-back.
<b>Smart Combo Routing</b>	
Vector<TagValue> m_smartComboRoutingParams	Support for <a href="#">Smart Combo Routing</a> .
<b>Order Combo Legs</b>	
OrderComboLegs() As Object	Holds attributes for all legs in a combo order.
<b>Not Held</b>	

boolean m_notHeld	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.
-------------------	--

## OrderState

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

## ScannerSubscription

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

double m_couponRateBelow	Filter out contracts with a coupon rate higher than this value. Can be left blank.
String m_excludeConvertible	Filter out convertible bonds. Can be left blank.
String m_instrument	Defines the instrument type for the scan.
String m_locationCode	The location.
String m_maturityDateAbove	Filter out contracts with a maturity date earlier than this value. Can be left blank.
String m_maturityDateBelow	Filter out contracts with a maturity date later than this value. Can be left blank.
double m_marketCapAbove	Filter out contracts with a market cap lower than this value. Can be left blank.
double m_marketCapBelow	Filter out contracts with a market cap above this value. Can be left blank.
String m_moody-RatingAbove	Filter out contracts with a Moody rating below this value. Can be left blank.
String m_moody-RatingBelow	Filter out contracts with a Moody rating above this value. Can be left blank.
int m_numberOfRows	Defines the number of rows of data to return for a query.
String m_scanCode	Can be left blank.
String m_scannerSettingPairs	Can leave empty. For example, a pairing "Annual, true" used on the "top Option Implied Vol % Gainers" scan would return annualized volatilities.
String m_spRatingAbove	Filter out contracts with an S&P rating below this value. Can be left blank.
String m_spRatingBelow	Filter out contracts with an S&P rating above this value. Can be left blank.
String m_stockTypeFilter	Valid values are: <ul style="list-style-type: none"> <li>• CORP = Corporation</li> <li>• ADR = American Depositary Receipt</li> <li>• ETF = Exchange Traded Fund</li> <li>• REIT = Real Estate Investment Trust</li> <li>• CEF = Closed End Fund</li> </ul>

**UnderComp**

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

## 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 = "100"
con2.m_exchange = "SMART";
```

```

con2.m_currency = "USD";

.reqContractDetails(2, con2);

//All conId numbers are delivered by the ContractDetail()

static public String contractDetails(int reqId, ContractDetails con-
tractDetails) {

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:

```

Contract contract = new Contract();
Order order = new Order();

contract.m_symbol = "USD";      // For combo order use "USD" 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 = "BUY";
order.m_totalQuantity = 1;
order.m_orderType = "MKT"
.placeOrder(OrderId, contract, order);
```

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



## 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](#)

### 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);  
}
```

## 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);
}
```



## Advisors

This chapter describes API functionality for users with Financial Advisor accounts, including the following topics:

- [Financial Advisor Orders and Account Configuration](#)
- [Excel DDE Support](#)
- [Support by Other API Technologies](#)
- [Improved Financial Advisor Execution Reporting](#)
- [Allocation Methods for Account Groups](#)
- [Java Code Samples for Financial Advisor API Orders](#)

## 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.

## Excel DDE Support

Starting with TWS release 824, DDE orders now have six Extended Order Attributes: *Good After Time*, *Good Till Date*, *FA Group*, *FA Method*, *FA Percentage*, and *FA Profile*. These can be left empty if they do not apply to an order. TWS Financial Advisor account configuration should be done manually for DDE access.

You can place FA orders on the Advisors page in the most recent release of the TwsDde.xls DDE for Excel API spreadsheet. For more information, see the [Advisors Page](#) topic.

## 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.



## 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 will serve to explain the new reporting scheme:

- 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."

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:

## 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!

<b>BUY ORDER</b>	<b>Positive Per- cent</b>	<b>Negative Per- cent</b>
<b>Long Posi- tion</b>	<b>Increases posi- tion</b>	<b>No effect</b>
<b>Short Posi- tion</b>	<b>No effect</b>	<b>Decreases posi- tion</b>

<b>SELL ORDER</b>	<b>Positive Per- cent</b>	<b>Negative Per- cent</b>
<b>Long Posi- tion</b>	<b>No effect</b>	<b>Decreases posi- tion</b>
<b>Short Posi- tion</b>	<b>Increases posi- tion</b>	<b>No effect</b>

## 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 ;
}
```

# ActiveX for Excel

This chapter describes the ActiveX for Excel sample spreadsheet, including the following topics:

- [Getting Started with the ActiveX for Excel API](#)
- [Using the ActiveX for Excel Sample Spreadsheet](#)

The ActiveX for Excel sample spreadsheet, **TwsActiveX.xls**, duplicates the functionality of the ActiveX for Excel API spreadsheet but internally uses an ActiveX component, **Tws.ocx**. One of the benefits of using this spreadsheet is that it can connect to a TWS or IB Gateway session that is running on a remote PC. The DDE for Excel API spreadsheet cannot do this.

**Note:** The methods, events and COM objects used in the code for the ActiveX for Excel sample spreadsheet are the same as those used in the ActiveX API. See the [ActiveX](#) chapter for complete details about the ActiveX API.

The following figure shows the Tickers page in the ActiveX for Excel API sample spreadsheet.

Symbol	Type	Expiry	Strike	Bid	Ask	Bid Price	Ask Price	Bid Size	Ask Size
MSFT	STK					26.42	26.43	250	10
YHOO	STK					20.12	20.13	135	10
GE	STK					28.17	28.18	18	10
Q000	STK					432.47	432.76	6	10
Q000	STK					46.21	46.22	505	10
IBM	STK					127.26	127.37	1	10
MOT	STK					7.41	7.42	184	10
Q000	OPT	200612	273	C	133	0.3755722	0.3903132	42	215.9
Q000	OPT	200612	273	P	133	0.4899657	0.5174134	58	1.24
IBM	OPT	200610	135	P	133	0.2355833	0.2460139	181	10.5
IBM	OPT	200610	73	C	133	0	0	447	58.2
MSFT	OPT	200610	23	C	133	0.2538075	0.2691957	3300	6.25
MSFT	OPT	200610	23	P	133	0.3939432	0.4053312	9447	0.15
Z	FUT	200612							
SPX	FUT	200612							
SPX	FUT	200612							
2B	FUT	200612							
2B	FUT	200612							

## Getting Started with the ActiveX for Excel API

We have created a sample Excel spreadsheet, **TwsActiveX.xls**, that uses an ActiveX control, **Tws.ocx**. You can use this spreadsheet with TWS as is, or use it create your own custom TWS API Excel application. It's easy to get started:

- [Download the API components](#), which includes the ActiveX for Excel sample spreadsheet, TwsActiveX.xls.
- Ensure that:
  - the application server is running and that it is configured to support ActiveX or
  - the IB Gateway is running.
- [Open the spreadsheet](#) and start using the ActiveX for Excel API.

The sample spreadsheet currently comprises several pages complete with sample data and action buttons that make it easy for you to get market data, send orders and view your activity.

### Download the API Components and Spreadsheet

We recommend using the sample Excel spreadsheet that we provide as a starting point toward creating your own ActiveX for Excel API. Follow the steps below to download the sample spreadsheet.

#### To install the ActiveX for Excel sample spreadsheet

1. From the [IB homepage](#), select *API Solutions* from the **Trading** menu.
2. Click the *IB API* button, then on the API Software page, find the column appropriate to your operating system and click *Download latest version*.

Windows users can download the beta test version of the API by using the **Windows Beta** column, or revert to the previous production version by selecting *Downgrade to Previous Version*.

3. Save the installation program to your computer, and if desired, select a different directory. Click **Save**.
4. Close any versions of TWS and Excel that you have running.
5. Locate the installation program you just saved to your computer, then double-click the file to begin the API installation.
6. Follow the instructions in the installation wizard.

**Note:** Before you can [use the spreadsheet](#), you must have TWS running and configured to support the ActiveX API. See [Run the API through TWS](#) for detailed instructions.

### Running the ActiveX for Excel API on 64-bit Windows XP Systems

To run the ActiveX for Excel API on 64-bit Windows XP systems, do the following:

1. Install Microsoft Visual C++ 2005 SP1 Redistributable Package (x86).
2. Install Microsoft Visual J# 2.0 Redistributable Package.
3. Download and install the API software.



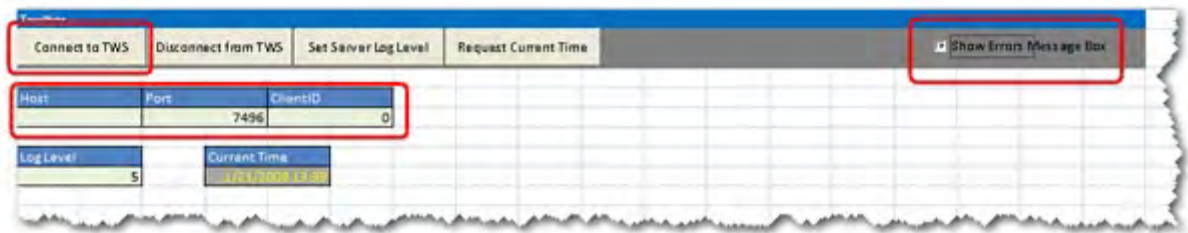
## Open the Sample Spreadsheet

After you have downloaded the sample spreadsheet and configured the application to allow the ActiveX for Excel API to link to it, open the spreadsheet and save it as your personal file.

**Note:** Note that not more than one API application can simultaneously access a single instance. The API application does not need to be running on the same computer on which the application is running.

### To open the sample spreadsheet

1. Go to the API installation folder in which the Excel API sample spreadsheet was installed (typically C:\Jts\Excel) and double-click **TwsActiveX.xls**.
2. Save the spreadsheet with a different file name. This lets you customize the spreadsheet without changing the original.
3. Click the **General** tab.
4. Modify the default values in the *Host*, *Port*, and *ClientID* cells, then click **Connect to TWS** on the Toolbar.
  - If you select the **Show Errors Message Box** check box, error messages display when you connect to TWS. In this case, you must click **OK** to dismiss any messages that appear.



## Using the ActiveX for Excel Sample Spreadsheet

The ActiveX for Excel API sample spreadsheet, TwsDde.xls, includes the following pages (tabs):

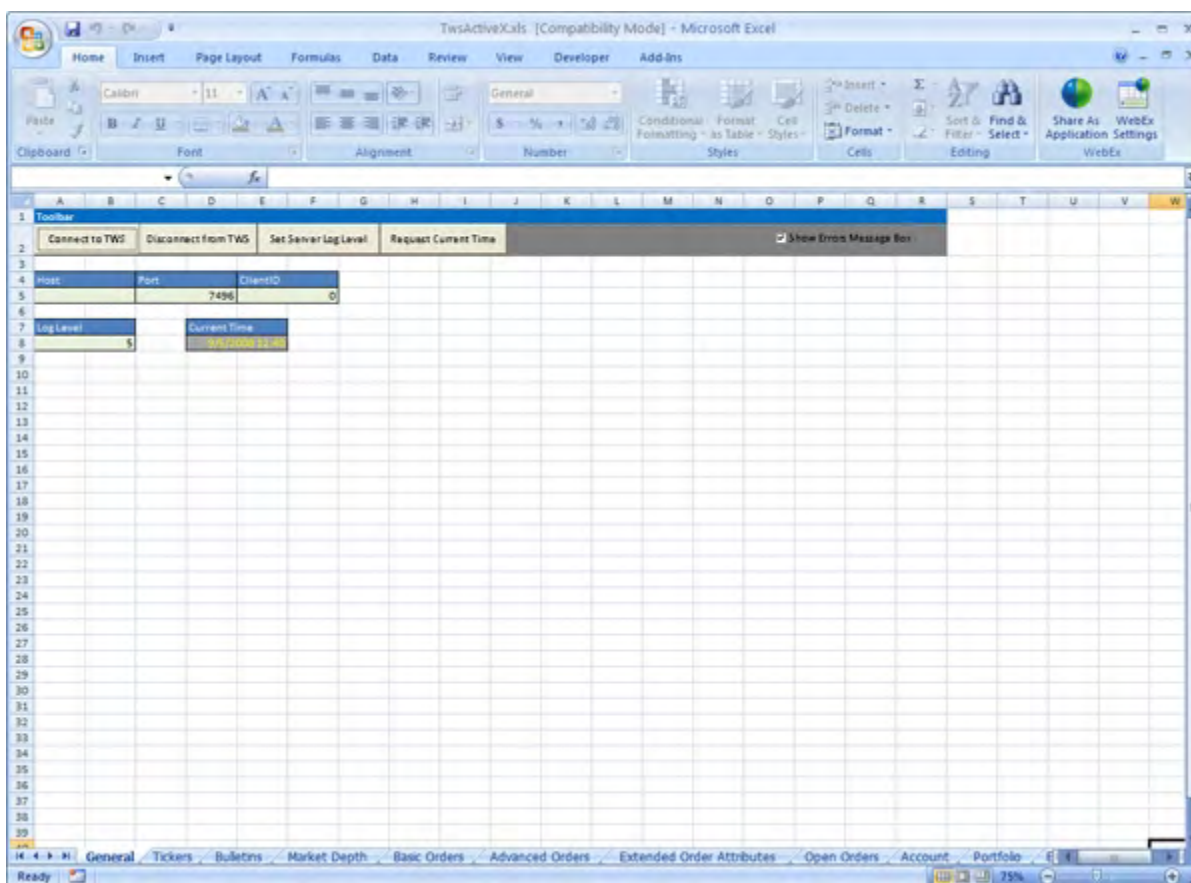
Page	Description
<a href="#"><u>General</u></a>	Lets you connect to and disconnect from TWS, set the server log level and request the current time.
<a href="#"><u>Tickers</u></a>	Lets you set up your ticker lines and request market data. You can view market data for all asset types including EFPs and combination orders.
<a href="#"><u>Bulletins</u></a>	Lets you subscribe to and view IB News Bulletins.
<a href="#"><u>Market Depth</u></a>	Lets you view market depth for selected quotes.
<a href="#"><u>Basic Orders</u></a>	Lets you send and modify orders, set up combination orders and EFPs, and request open orders.
<a href="#"><u>Conditional Orders</u></a>	Lets you create an order whose submission is contingent on other conditions being met, for example an order based on a prior fill.
<a href="#"><u>Advanced Orders</u></a>	Lets you send and modify advanced orders types that require the use of extended order attributes, such as Bracket, Scale and Trailing Stop Limit orders.
<a href="#"><u>Extended Order Attributes</u></a>	Used in conjunction with the Basic Orders, Advanced Orders, Conditional Orders and Advisors pages, this page lets you change the time in force, create Hidden or Iceberg orders and apply many other order attributes.
<a href="#"><u>Open Orders</u></a>	Shows you all transmitted orders, including those that have been accepted by the IB system, and those that are working at an exchange.
<a href="#"><u>Account</u></a>	Provides up to date account information and displays your portfolio.
<a href="#"><u>Portfolio</u></a>	Displays all your current positions.
<a href="#"><u>Executions</u></a>	Lets you view all execution reports, and includes a filtering box so you can limit your results.
<a href="#"><u>Historical Data</u></a>	Request historical data for an instrument based on data you enter in a query.
<a href="#"><u>Contract Details</u></a>	Lets you collect contract-specific information you will need for other actions, including the conid and supported order types for a contract
<a href="#"><u>Bond Contract Details</u></a>	Lets you collect bond contract-specific information you will need for other actions, including bond coupon and maturity date.

<a href="#">Real Time Bars</a>	Lets you request and view real time bars from TWS.
<a href="#">Market Scanner</a>	Lets you view market scanner parameters and subscribe to TWS market scanners.
<a href="#">Fundamentals</a>	Lets you request and view Fundamentals data from TWS.
<a href="#">Advisors</a>	Lets Financial Advisors send and modify FA orders.
<a href="#">Log</a>	Lets you view all error messages.

## General Page

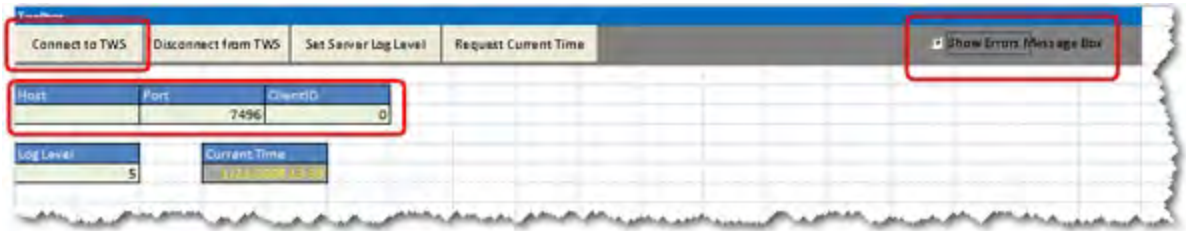
Use the General page to:

- Connect to TWS.
- Disconnect from TWS.
- Set the level of log entry detail used by the server when processing API requests.
- Request the current server time.



## To connect to TWS

1. Click **Connect to TWS** in the toolbar.
  - If required, change the values in the *Host*, *Port* and *ClientID* cells.
  - Select the **Show Errors Message Box** to display errors when connecting to TWS.



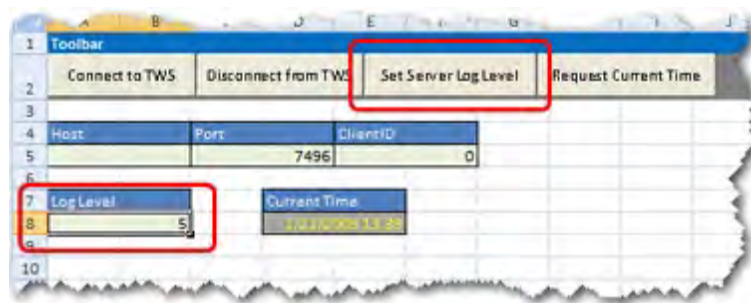
### To disconnect from TWS

1. Click **Disconnect from TWS** in the toolbar.

### To set the server log level

1. Type one of the following values in the *Log Level* cell:
  - 1 = SYSTEM
  - 2 = ERROR
  - 3 = WARNING
  - 4 = INFORMATION
  - 5 = DETAIL

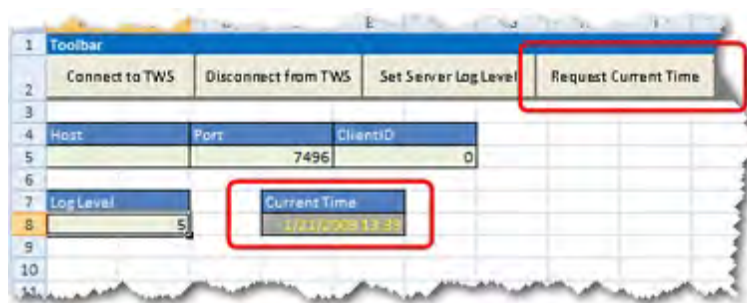
The higher the number, the greater the level of detail and performance overhead.



2. Click **Set Server Log Level** in the toolbar.

### To request the current time

1. Click **Request Current Time** in the toolbar.



### General Page Toolbar Buttons

The toolbar on the General page includes the buttons described below.

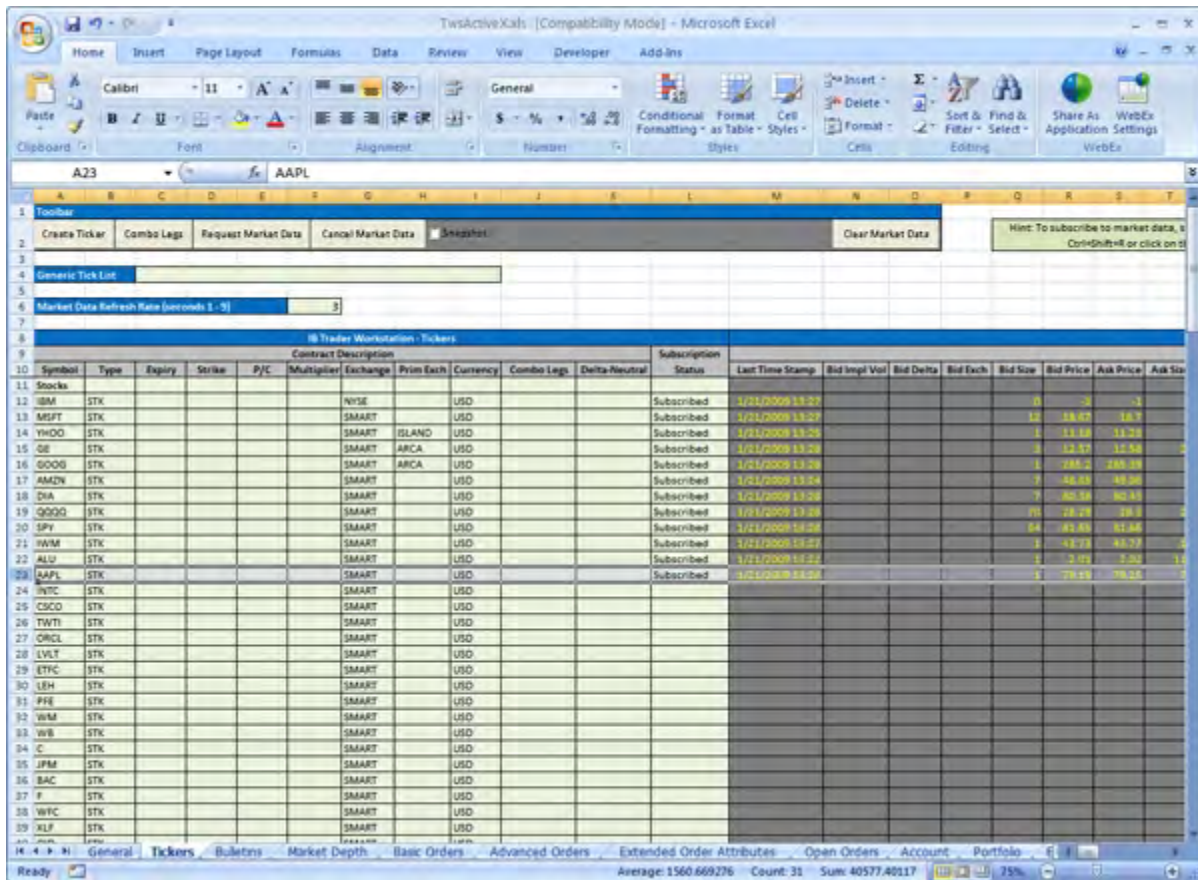
Button	Description
Connect to TWS	Connects to TWS.
Disconnect from TWS	Disconnects from TWS.
Set Server Log Level	Sets the level of detail of entries in the log.txt log file.
Request Current Time	Requests the current server time.

The toolbar also includes the **Show Errors Message Box** check box, which when selected, displays error when connecting to TWS.

### Tickers Page

Use the Tickers page to:

- Create market data (ticker) lines.
- Request market data.
- Create a combination order for options.
- Create market data line for Exchange for Physical (EFP) combination orders.



## Using the Tickers Page

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To create a ticker using the Create Ticker button

1. Click the **Tickers** tab at the bottom of the spreadsheet.
2. Click the line number to the left of a blank row to select the row. You must have a blank row selected to create a ticker line.
3. Click the **Create Ticker** button on the toolbar and enter information in the Create Tickers dialog.
4. Click **OK**.

For stocks, you only need to specify the *Symbol*, *Type*, *Exchange* (usually SMART), and *Currency*.



### To create a ticker on the spreadsheet

1. Select a blank cell in the *Symbol* column and enter a symbol.
2. Tab through the all contract description fields and enter data where necessary, for example if you are entering a stock ticker, you don't need values in the Expiry, Strike, P/C and Multiplier fields.

**Note:** The Exchange field accepts the following values: SMART (for smart order routing), and any valid exchange acronym.

### To request market data for a ticker

1. Select the ticker row for which you want to request market data by clicking the row number.
2. Enter a comma-separated list of generic tick values in the *Generic Tick List* cell. For details about generic tick values, see [Generic Tick Types](#).
3. Optionally, click the **Snapshot** check box to request a single snapshot of market data.
4. Click **Request Market Data** on the toolbar.

To get market data for a group of tickers, select multiple ticker rows while holding down the **Shift** key, then click **Request Market Data** multiple times until all rows are showing data.

### To set the refresh rate

The market data refresh rate determines how often the link to TWS is refreshed.

- Enter the refresh rate value (in whole numbers, in seconds) in the Market Data Refresh Rate cell.

TWS market data updates every 3 seconds by default.

### Tickers Page Toolbar Buttons

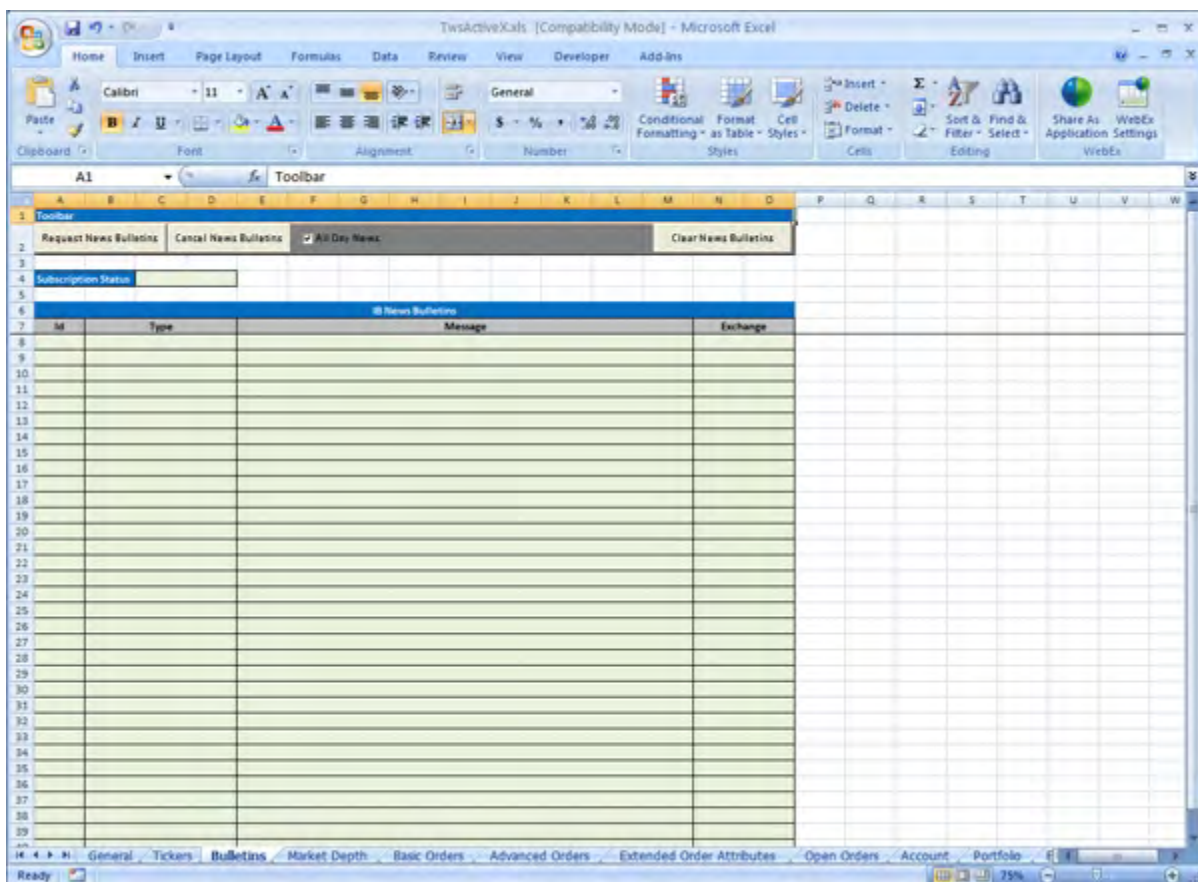
The toolbar on the Tickers page includes the buttons described below.

Button	Description
Create Ticker	Opens the Ticker box. Enter information to create a market data line.
Combo Legs	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
Request Market Data	Select a line and click to get market data for the selected contract.
Cancel Market Data	Cancel market data for the selected ticker.
Clear Market Data	Clears all market data from the page.

The toolbar also includes the **Snapshot** check box, which when selected, requests only a single snapshot of market data.

## Bulletins Page

Use the Bulletins page to request and view IB news bulletins. Simply click **Request News Bulletins** in the toolbar. News bulletins display in the table on the page. To request all the existing bulletins for the current day and any new ones, select the **All Day News** check box on the toolbar. If this check box is not selected, you will receive only new bulletins.





## Bulletins Page Toolbar Buttons

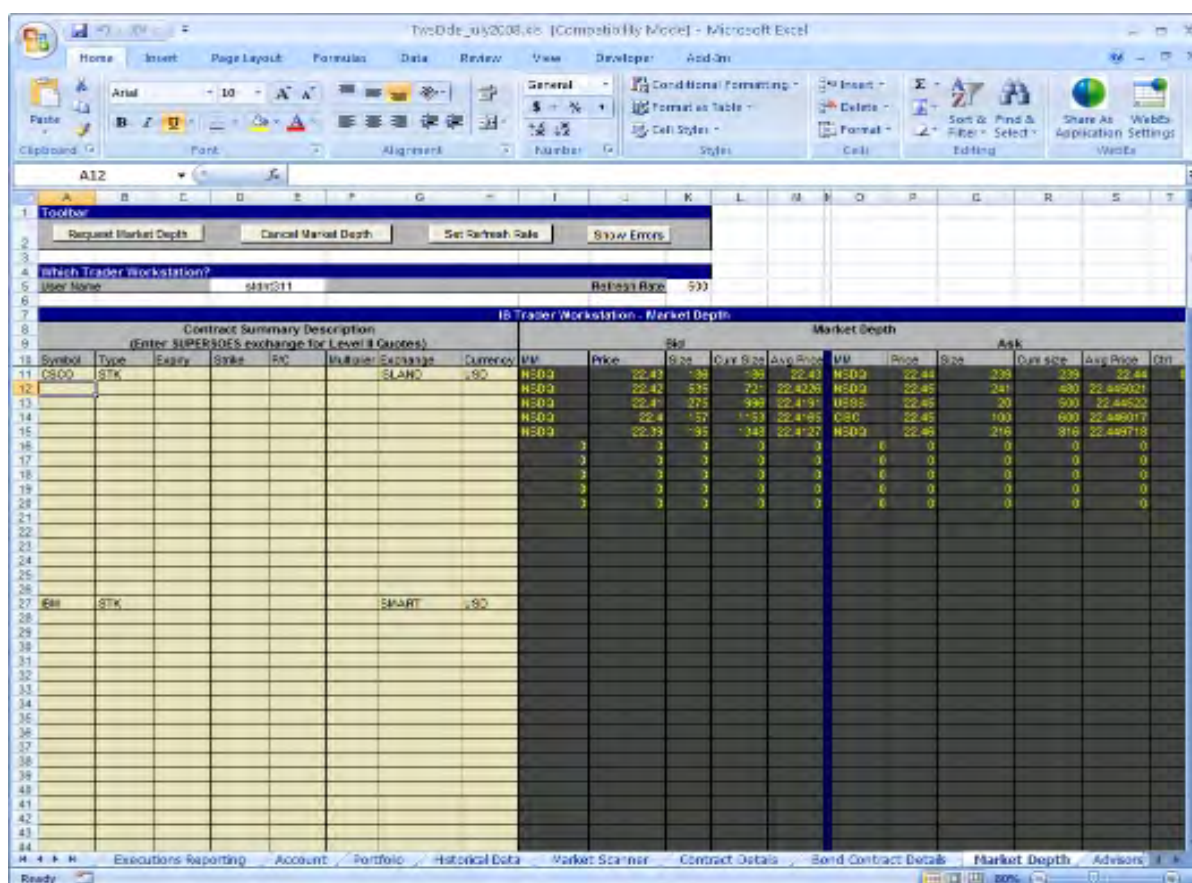
The toolbar on the Tickers page includes the buttons described below.

Button	Description
Request News Bulletins	Requests all the new news bulletins.
Cancel News Bulletins	Cancels receipt of all news bulletins.
Clear News Bulletins	Clears all news bulletins from the page.

The toolbar also includes the **All Day News** check box, which when selected, requests all the existing bulletins for the current day and any new ones.

## Market Depth Page

Use the Market Depth page to view market depth for selected contracts. You can also view market depth for NYSE-listed products through the Open Book Market Data Subscription, and NASDAQ-listed products through the TotalView Market Data Subscriptions, if you have signed up for those subscriptions.



## Using the Market Depth Page

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To request market depth for a contract

1. Click the **Market Depth** tab at the bottom of the spreadsheet to open the Market Depth page.
2. Select the ticker symbol for which you want to request the market depth, or enter a new ticker on a blank line.
3. Click the **Request Market Depth** button on the toolbar.

### To reset the market data refresh rate for market depth

1. Type the desired market data refresh rate in seconds in the *Market Depth Refresh Rate* cell. The value must be in whole numbers from 1 to 9.

## Market Depth Page Toolbar Buttons

The toolbar on the Market Depth page includes the following buttons:

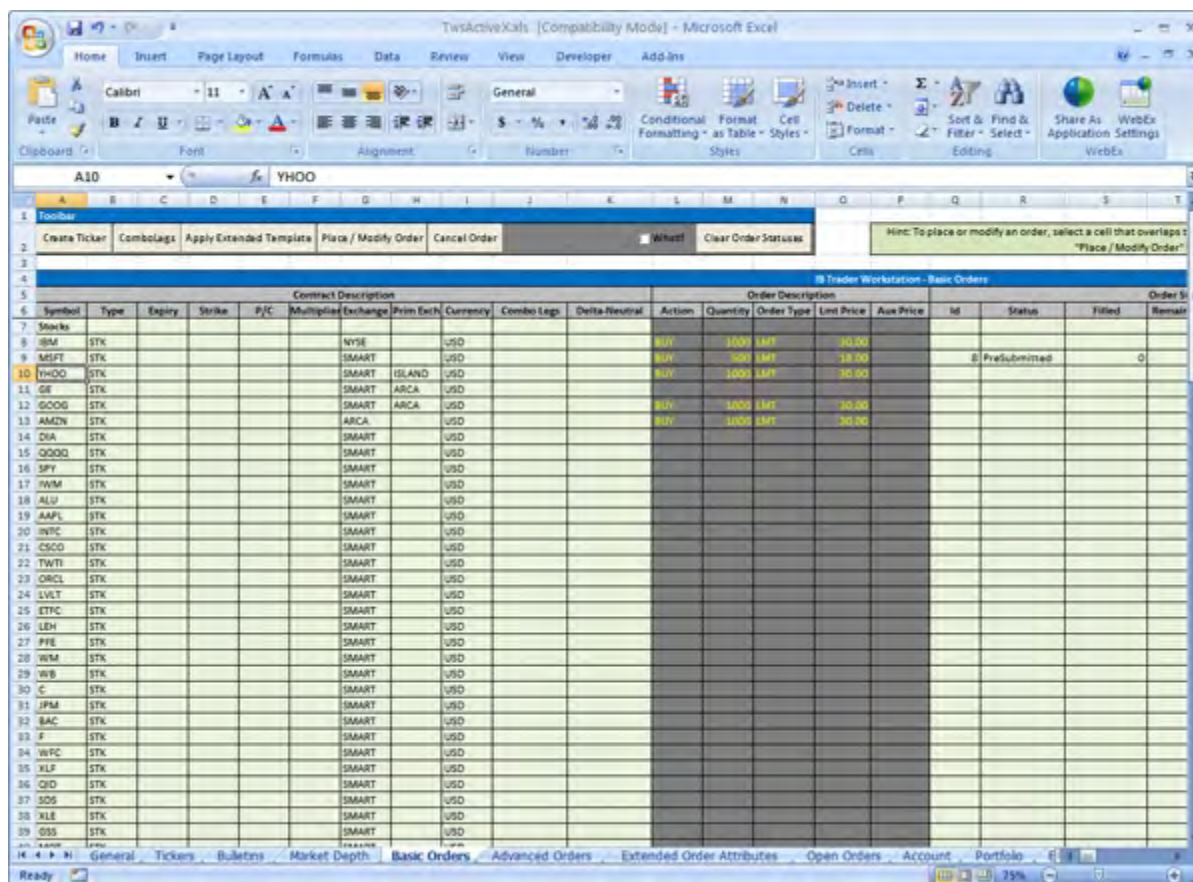
Button	Description
Request Market Depth	View bid/ask depth prices for the selected contract.
Cancel Market Depth	Cancel market depth for the selected contract.
Clear Market Depth	Clears all market depth data from the page.

The page also includes a *Market Depth Refresh Rate* cell, which lets you reset the market depth refresh rate in seconds, in whole numbers from 1 - 9. The default refresh rate is 3 seconds.

## Basic Orders Page

Use the Basic Orders page to:

- Create an order.
- Place a “what if” order, which shows you the margin and commission information before you place an order.
- Create a "basket" of orders.
- Modify and cancel orders.
- Create combination orders.



## Placing Orders

This topic describes how to place the following types of orders on the Orders page:

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To place an order

1. Click the **Basic Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.

You must define the *Action* (Buy, Sell or Short Sell), *Quantity*, *Order Type*, *Limit Price* (unless it's a market order) and if necessary, the *Aux. Price* for order types that require it.

4. If desired, select the contract (the ticker row) and apply extended order attributes by clicking the **Apply Extended Template** button on the toolbar. This applies all attributes you have defined on the [Extended Order Attributes](#) page to the selected contract.
5. Click the **Place/Modify Order** button on the toolbar.

### To place a "basket" of orders

1. Click the **Basic Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using fields in the *Order Description* section.
4. Repeat Steps 1 and 2 for additional orders.
5. Select a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.
6. Click the **Place/Modify Order** button.

#### To modify an order (or group of orders)

1. On the Basic Orders page, change any necessary parameters in an order or group of orders.
2. Select the order or a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.
3. Click the **Place/Modify Order** button.

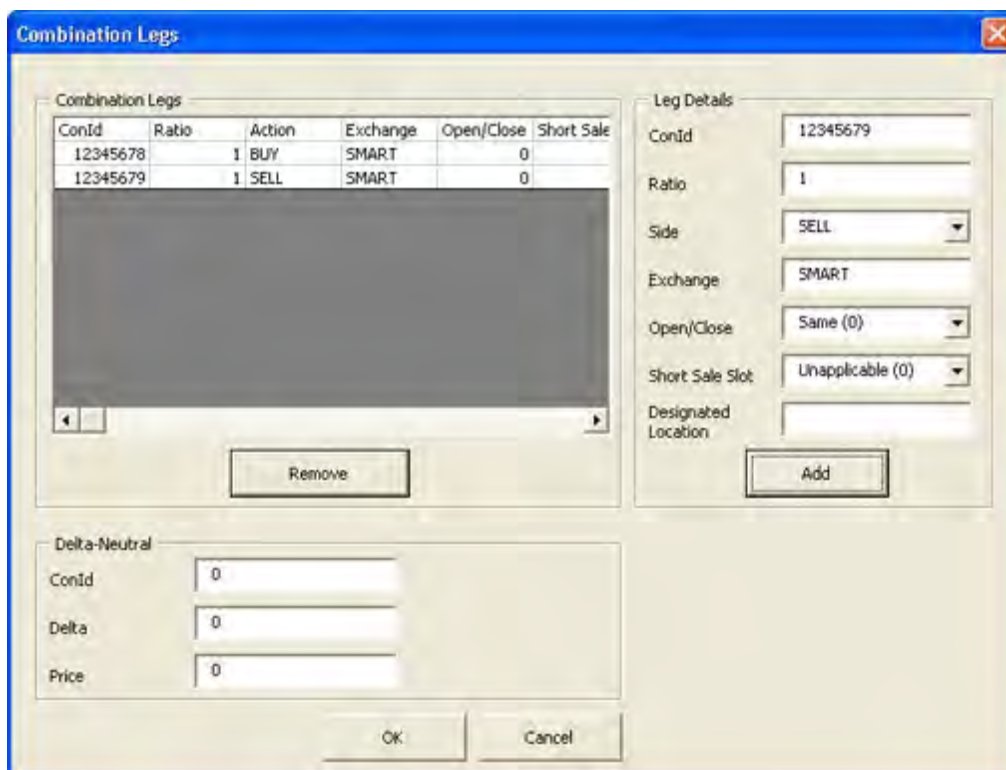
#### Placing a Combination Order

A combination order is a special type of order that is constructed of many separate legs but executed as a single transaction.

To buy a calendar spread, you would:

#### To create a calendar spread order

The following example walks you through the process of placing a hypothetical calendar spread order for XYZ on ISE.



1. Use the [Contract Details](#) page to get the contract id for both of the leg definitions.
  - The conid for XYZ option JUL08 17.5 CALL on ISE is "12345678".
  - The conid for XYZ option AUG08 17.5 CALL on ISE is "12345679".
2. Click the **Basic Orders** tab to build the combo leg definitions. Click the **Combo Legs** button on the Basic Orders page toolbar and enter leg information. Your leg information is translated into the format:

[CMBLGS]\_[NumOfLegs]\_[Combo Leg Definitions]\_[CMBLGS]

where:

- [CMBLGS] is the delimiter used to identify the start and end of the leg definitions
- [NumOfLegs] is the number of leg definitions
- [Combo Leg Definitions] defines N leg definitions, and each leg definition consists of [conid]\_[ratio]\_[action]\_[exchange]\_[openClose], so the resulting combo substring looks as follows:

CMBLGS\_2\_17496957\_1\_BUY\_EMPTY\_0\_15910089\_1\_SELL\_EMPTY\_0\_CMBLGS

3. The combination leg definitions must occur before the extended order attributes. The full place order DDE request string will look like this:

```
=acctName|ord!id12345?place?BUY_1_XYZ_BAG_ISE_LMT_1_CMBLGS_2_12345678_1_BUY_EMPTY_0_12345679_1_SELL_EMPTY_0_CMBLGS_DAY_EMPTY_0_O_0_EMPTY_0_EMPTY_0_0_EMPTY_0_0
```

If the order legs do not constitute a valid combination, one of the following errors will be returned:

- 312 = The combo details are invalid.
- 313 = The combo details for '<leg number>' are invalid.
- 314 = Security type 'BAG' requires combo leg details.
- 315 = Stock combo legs are restricted to SMART exchange.

**Note:** 1. The exchange for the leg definition must match that of the combination order. The exception is for a STK leg definition, which must specify the SMART exchange.

2. The openClose leg definition value is always 'SAME' (i.e.0) for retail accounts. For institutional accounts, the value may be any of the following: (SAME, OPEN, CLOSE).

## Supported Order Types

The order types currently supported through the ActiveX for Excel API are:

- Limit (LMT)
- Market (MKT)
- Limit if Touched (LIT)
- Market if Touched (MIT)
- Market on Close (MOC)
- Limit on Close (LOC)
- Pegged to Market (PEGMKT)
- Relative (REL)
- Stop (STP)
- Stop Limit (STPLMT)
- Trailing Stop (TRAIL)
- Trailing Stop Limit (TRAILLIMIT)
- Volume-Weighted Average Price (VWAP)
- Volatility orders (VOL)

## Basic Orders Page Toolbar Buttons

The toolbar on the Basic Orders page includes the following buttons:

Button	Description
Create Ticker	Opens the Ticker box. Enter information to create a market data line.
Combo Legs	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one. You can also enter Delta Neutral information.



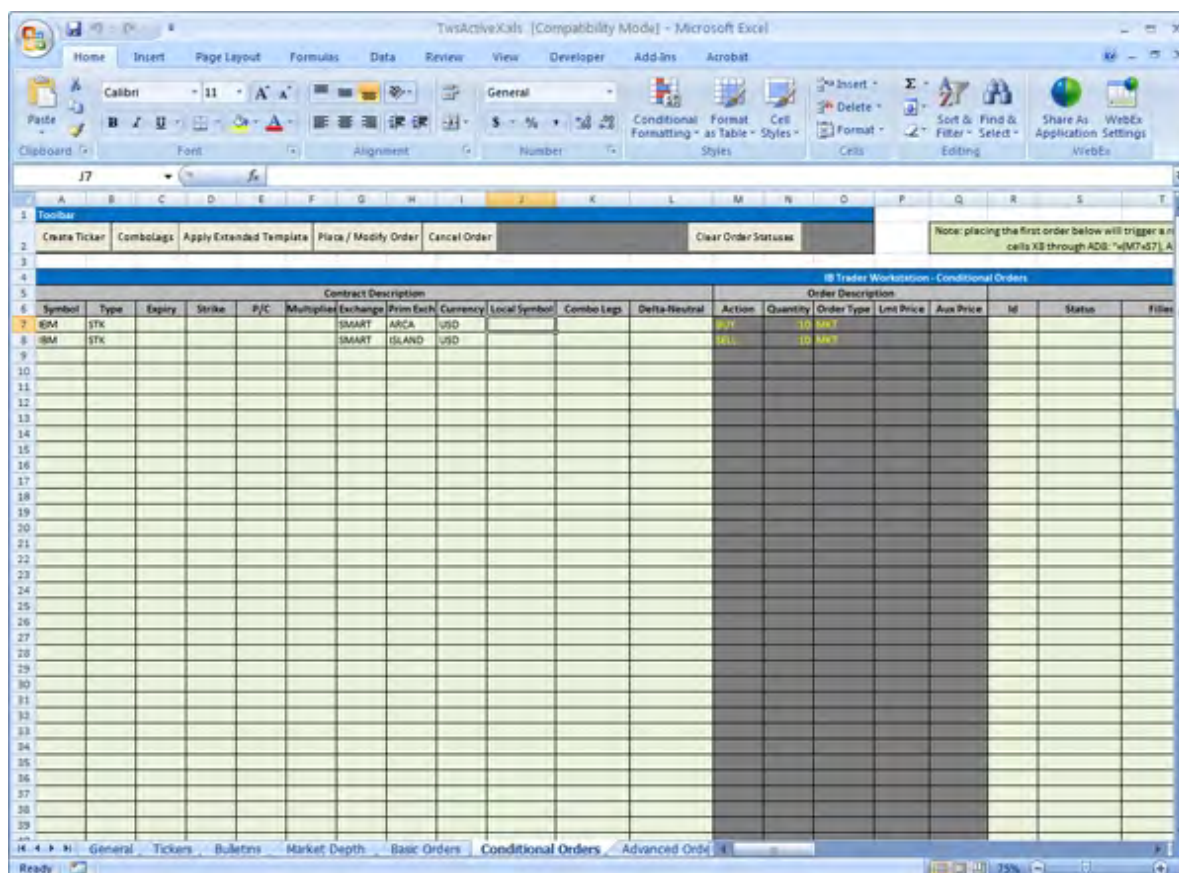
Apply Extended Template	Applies the current values on the Extended Order Attributes page to the highlighted order row.
Place/Modify Orders	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
Cancel Order	This button cancels the selected order(s).
Clear Order Statuses	Clears all order status information from the page.

There is also a **WhatIf** check box on the toolbar. When checked, you will receive margin and commission data as if the order were placed, but the order will NOT be placed.

### Conditional Orders Page

Use the Conditional Orders page to create an order whose submission is contingent on other conditions being met, for example, an order based on a prior fill or a change in the bid or ask price.

To see the Conditional Statement fields (in blue), use the scroll bar on the bottom of the page to scroll to the right.



## Setting Up Conditional Orders

**Note:** Ensure that TWS is running, and that you have entered your user name in the *User Name* field in the *Which Trader Workstation?* section of all pages in the Excel spreadsheet to properly connect to TWS.

### To set up a conditional order

1. On the **Conditional Orders** page, first create the order you want transmitted when a condition is met by defining the contract in the *Contract Description* fields, and then using the *Order Description* area to set up the order parameters.
2. In the blue Condition Statements area, use the *Statement* field to set the criteria which must be met to trigger the order. When the Statement = TRUE, your order will be submitted.

The sample spreadsheet includes a pair of orders, with the second orders transmission depending on the first order being completely filled. In this case, the Statement field trigger is that the value in cell T10 (the *Filled* field) must be equal to the value in M10 (the order *Quantity* field).

3. Type **ADD** in the *ADD/MOD* field because you are creating a one-time order.
4. Define the remaining order parameters just as you did in the *Order Description* area.

IB Trader Workstation - Conditional Orders												
Option			Order Description						Order			
Primary Exchange	Currency	Comb Legs	Leave this empty	Action	Quantity	Order Type	Lmt Price	Aux Price	Ctr Id	Id	St Filled	Remainin
ARCA	USD			BUY	10 MKT				0	id1	Fi	10 0
ISLAND	USD			SELL	10 MKT				0	id1	Fi	10 0
	usd			buy	200 lmt		81.00					

5. Complete the necessary fields on the **Conditional Orders** page according to the syntax in the following table.

Field	Description
Statement	An Excel function which returns a true or false. When true, the order will be submitted; when false, nothing happens.
ADD/MOD	Use ADD for a one-time order. Use MOD to continue checking and modifying the order until it is completely filled. This is the field that activates a conditional order, and orders will be activated only with the "ADD" or "MOD" tags.
Action	BUY SELL
Quantity	Enter the quantity of the order.
Order Type	Refer to <a href="#">list of supported order types</a> .
Lmt Price	The limit price for Limit and Stop Limit order types.
Aux. Price	The stop-election price for Stop and Stop Limit order types, or the offset for relative orders.



All of the fields described above may be variables that depend on other cells, so any type of conditional order may be created.

## Conditional Order Examples

### If-Filled order

An if-filled order is an order that executes if a prior order executes. To create an if-filled order with the condition "If a Buy order fully executes, enter a sell limit order at a price of \$50.00":

Field	Value
Statement	Filled cell = 100
ADD/MOD	ADD
Action	SELL
Quantity	100
Order Type	LMT
Lmt Price	50
Aux. Price	empty

### Price-change order

A price-change order will be triggered if a specific bid or ask price is greater than, less than or equal to a specific price. To create a price change order with the condition "If the bid price drops below 81.20, submit a buy limit order for 200 shares with a limit price of \$81.10:

Field	Value
Statement	<p>On the <b>Tickers</b> page, put your cursor in the bid price field you want to use, then copy the value that appears in the formula bar ("=" entry field) at the top of the spreadsheet. This value looks something like this:</p> <p>=username tik!id4?bid</p> <p>where "4" identifies the bid price for a specific contract. Paste this in the formula bar ("=" entry field) for the Statement, and add your qualifier, "=" "&gt;" or "&lt;" followed by the price. In this example, the formula would be:</p> <p>=username tik!id4?bid&lt;81.20</p>
ADD/MOD	ADD

Action	BUY
Quantity	200
Order Type	LMT
Lmt Price	81.10
Aux. Price	Not used in this example.

### To modify an order (or basket of orders)

1. Select the order or a group of orders.
  - To select a group of contiguous orders, highlight the first order, hold down the **Shift** key, then highlight the last order of the group.
  - To select a group of non-contiguous orders, hold the **Ctrl** key down as you select each order.
2. Click the **Place/Modify Order** button.
3. Change any necessary parameters, then click the **Place/Modify Order** button.

### Conditional Orders Page Toolbar Buttons

The toolbar on the Conditional Orders page includes the following buttons:

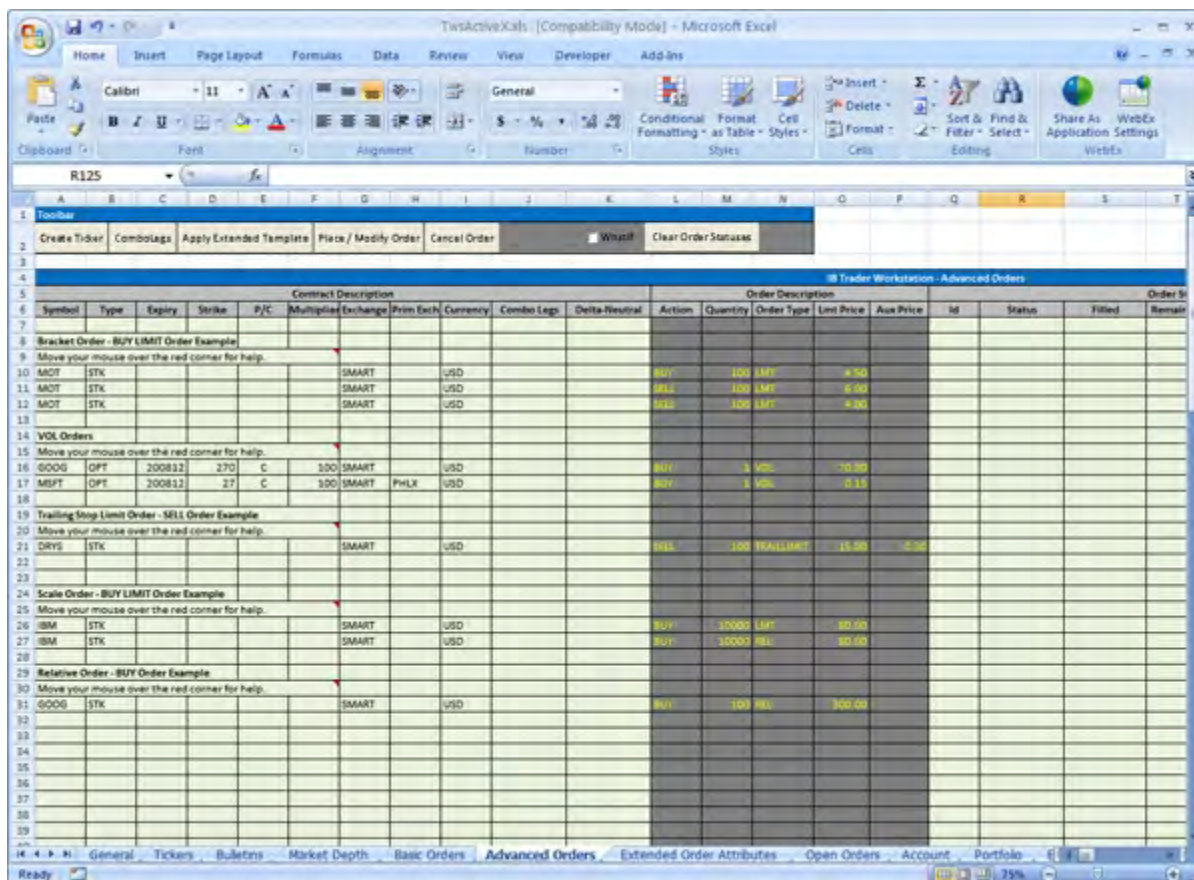
Button	Description
Combo Legs	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
Place/Modify Order	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
Apply Extended Template	Applies all attributes on the <a href="#">Extended Order Attributes page</a> to the selected order(s).
Cancel Order	This button cancels the order(s) you have highlighted.
Show Errors	Jumps to the Error Code field and shows the error code.

### Advanced Orders Page

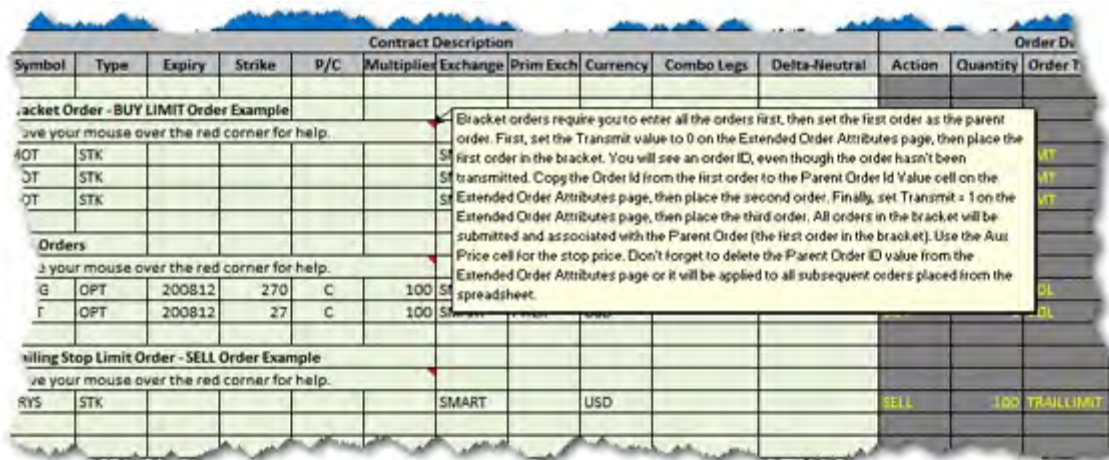
Use the Advanced Orders page to:

- Bracket orders
- VOL orders
- Trailing Stop Limit Orders
- Scale Orders

- Relative Orders



This page includes several example orders with mouseover help to assist you in learning how to place these orders. Simply move your mouse over the red triangle of the corner of cells on the page to display pop-up help.



For more information about using extended order attributes for individual orders or groups of orders, see [Apply Extended Order Attributes to Individual Orders and Groups of Orders](#)

## Placing a Bracket Order

Bracket orders in the ActiveX for Excel sample spreadsheet require the use of the extended order attributes *Transmit* and *Parent Order Id*. You must turn *Transmit* off until the order is completely set up, and you must identify the first order in the bracket as the Parent Order.

### To place a Buy-Limit bracket order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Enter the contract descriptions and order descriptions for all three orders on three contiguous rows:
  - The first order should be a BUY LMT order.
  - The second order should be a SELL STP order.
  - The third order should be a SELL LMT order.
3. Click the **Extended Order Attributes** tab. Change the value for *Transmit* to **0** (row 13 on the Extended Order Attributes page).

This ensures that your orders are not transmitted until you have completed the order setup.

4. Click the **Advanced Orders** tab, highlight the first order in the bracket order, then click the **Place/Modify Order** button.

The order is not executed, but the system generates an Order ID.

5. Copy the Order ID for the first order, omitting the “id” prefix, then click the **Extended Order Attributes** tab and paste the Order ID into the *Value* field for *Parent Order Id* (row 14). This value will be applied to all subsequent orders until you remove it from the Extended Order Attributes page.

The first order of the bracket order is now the primary order.

6. Click the **Advanced Orders** tab, highlight the second order, then click the **Place/Modify Order** button.

The order is not executed but is now associated with the primary order by means of the Parent Order Id extended order attribute.

7. Click the **Extended Order Attributes** tab and change the value for *Transmit* back to **1** (row 13).
8. Click the **Advanced Orders** tab, highlight the third order in the bracket order, then click the **Place/Modify Order** button. The entire bracket order is transmitted.
9. When you are done placing your bracket order, go to the **Extended Order Attributes** page and delete the *Parent Order Id* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Volatility Order

In the ActiveX for Excel sample spreadsheet, you place VOL orders by entering values for the following extended order attributes:

- Volatility
- Volatility Type

- Reference Price Type
- Continuous Update
- Underlying Range (Low) - optional
- Underlying Range (High) - optional
- Hedge Delta Order Type - optional
- Hedge Delta Aux Price - optional

### To place a VOL order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.
  - Enter **VOL** in the *Order Type* field.
4. Click the **Extended Order Attributes** tab. Enter values in the *Value* field for the following extended order attributes:
  - Volatility - This value represents the volatility to use in calculating a limit price for the option. Enter this value as a percentage, not as the market data is displayed. For example, enter 17.12 instead of .1712.
  - Volatility Type - Enter 1 for daily volatility or 2 for annual volatility.
  - Reference Price Type - This value is used to compute the limit price sent to an exchange and for stock range price monitoring. Enter 1 to use the average of the best bid and ask; or 2 to use NBB (bid) when buying a call or selling a put, or the NBO (ask) when selling a call or buying a put.
  - Continuous Update - Enter 1 to automatically update the option price as the underlying stock price (or futures price, for index options) moves. Enter 0 if you do not want to use this feature.
5. On the **Extended Order Attributes** page, enter values in the *Value* field for the following optional extended order attributes:
  - Underlying Range (Low) - Enter a low-end acceptable stock price relative to the selected option order. If the price of the underlying instrument falls below the lower stock range price, the option order will be canceled.
  - Underlying Range (High) - Enter a high-end acceptable stock price relative to the selected option order. If the price of the underlying instrument rises above the higher stock range price, the option order will be canceled.
  - Hedge Delta Order Type - Enter LMT, MKT or REL. Enter NONE if you do not want to use delta hedging.
  - Hedge Delta Aux Price - If you have entered LMT or REL as the Hedge Delta Order Type, enter the price as the value for this attribute.
6. Click the **Advanced Orders** tab, then highlight the order row.
7. Click the **Apply Extended Template** button. The values you entered for the extended order attributes are applied to the order row and displayed in the *Extended Order Attributes* section of the page.
8. With the order row highlighted, click the **Place/Modify Order** button.

9. When you are done placing VOL orders, go to the **Extended Order Attributes** page and delete the VOL order values you entered. If you do not, these values will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Trailing Stop Limit Order

In TWS, there are four values that make up a trailing stop limit order:

- trailing amount
- stop price
- limit price
- limit offset

In the ActiveX for Excel API spreadsheet, you enter the trailing amount, stop price and limit price. There is no field or extended order attribute for the limit offset value. You must include the limit offset in the stop price (the *Trail Stop Price* extended order attribute).

### To create a Trailing Stop Limit Order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.
  - Enter **BUY** or **SELL** in the *Action* field.
  - Enter the limit price in the *Lmt Price* field.
  - Enter **TRAILLIMIT** in the *Order Type* field.
  - Enter the trailing amount in the *Aux Price* field.
4. Click the **Extended Order Attributes** tab. Specify the trailing stop price as an extended order attribute. Type this value in the *Trail Stop Price Value* field.
  - The Trail Stop Price value must include the limit offset.  
For a sell order:

$$\text{Trail Stop Price} = \text{Limit Price} - \text{Trailing Amount} - \text{Limit Offset}$$

For a buy order:

$$\text{Trail Stop Price} = \text{Limit Price} + \text{Trailing Amount} + \text{Limit Offset}$$

5. On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The *Trail Stop Price* value is applied to the selected order and displayed in the *Trail Stop Price* field in the *Extended Order Attributes* section of the page.
6. Click the **Place/Modify Order** button.
7. When you are done placing your order, go to the **Extended Order Attributes** page and delete the *Trail Stop Price* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Scale Order

In the ActiveX for Excel sample spreadsheet, you place scale orders by entering values for the following extended order attributes:

- Scale Component Size
- Scale Price Increment

### To place a scale order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields. The order type should be LMT or REL.
4. Click the **Extended Order Attributes** tab. Enter values in the *Value* field for the following extended order attributes:
  - Scale Component Size - Enter the size of the first, or initial, order component. For example, if you submit a 10,000-share order with a Scale Component Size value of 1000, the first component will be for 1000 shares.
  - Scale Price Increment - Enter the amount used to calculate the per-unit price of each component in the scale ladder. This cannot be a negative number.

**Note:** As of API Release 9.41, the Scale Num Components not supported.

5. On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The scale order values are applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
6. Click the **Place/Modify Order** button.
7. When you are done placing your order, go to the **Extended Order Attributes** page and delete the scale order values you entered. If you do not, these values will be applied to all subsequent orders that you place in the spreadsheet.

## Placing a Relative Order

In the ActiveX for Excel sample spreadsheet, you place relative orders by entering a value for the *Percent Offset* extended order attribute.

### To place a relative order

1. Click the **Advanced Orders** tab at the bottom of the spreadsheet.
2. Define a contract by typing a symbol in a blank *Symbol* field, then entering information in the relevant contract description fields.
3. Select a contract and set up the order using the *Order Description* fields.
  - Enter **REL** as the order type.
  - Enter the price cap in the *Lmt Price* cell.
4. Click the **Extended Order Attributes** tab. Enter a percentage in decimal form in the *Value* field for the *Percent Offset* extended order attribute.

5. On the **Advanced Orders** page, select the order row and click the **Apply Extended Template** button. The percent offset value is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
6. Click the **Place/Modify Order** button.
7. When you are done placing your order, go to the **Extended Order Attributes** page and delete the *Percent Offset* value you entered. If you do not, this value will be applied to all subsequent orders that you place in the spreadsheet.

### Advanced Orders Page Toolbar Buttons

The toolbar on the Advanced Orders page includes the following buttons:

Button	Description
Create Ticker	Opens the Ticker box. Enter information to create a market data line.
Combo Legs	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
Apply Extended Template	Applies the current values on the Extended Order Attributes page to the highlighted order row.
Place/Modify Orders	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
Cancel Order	This button cancels the selected order(s).
Clear Order Statuses	Clears all order status information from the page.

There is also a **WhatIf** check box on the toolbar. When checked, you will receive margin and commission data as if the order were placed, but the order will NOT be placed.

### Extended Order Attributes Page

The Extended Order Attributes page includes all of the optional attributes you can use when you send an order, such as setting a display size to create an iceberg order, adding orders to an OCA group, and setting the transmit date for a Good After Time order. Once you define the attributes on this page, you can apply them to a single order or selected group of orders using the **Apply Extended Template** button, which occurs on both the Orders page and the Conditional Orders page. The attributes populate the extended order attributes fields that follow the *Order Status* fields to the far right of the page.



The screenshot displays the Microsoft Excel interface with the 'TwsActiveXcals [Compatibility Mode] - Microsoft Excel' title bar. The ribbon includes Home, Insert, Page Layout, Formulas, Data, Review, View, Developer, and Add-ins. The active sheet is titled 'Extended Order Attributes'. A yellow callout box contains the text: 'Note: This page has a new ability as of API & 9.1. It is now a template that can be applied to any order.'

Attributes	Value	Note
Time in Force (DAY, GTC, etc.)	DAY	The time in force. Valid values are: DAY, GTC, IOC, STL.
OCA Group		"One Cancels All" group name.
Account (Institutional only)		
Open/Closed(OnOpen, OnClose (Institutional only)	O	
Origin (OnCustomer, 1= Firm) (Institutional only)	0	
Order Ref	HellWorld	
Transmit (Default, 1=true)	1	
Parent Order Id		
Block Order (Default, 1=true)	0	
Sweep To Fill (Default, 1=true)	0	
Display Size		
Trigger Method	0	OnDefault, 1=Double_Bid_Ask, 2=Last, 3=Cumulative, 4=Mid_Ask, 7=Last, or Bid_Ask, 8=Midpoint
Hidden (Default, 1=true)	0	
Discretionary Amount (SMART Routing)		
Good After Time	FORMAT: 20080707 08:00:00 (time zone)	
Good Till Date	FORMAT: 20080707 08:00:00 (time zone)	
TA Group (Financial Advisors only)		NOTE:
TA Method (Financial Advisors only)		THESE FOUR FIELDS
TA Percentage (Financial Advisors only)		ONLY APPLY TO FINANCIAL ADVISOR ORDERS
TA Profile (Financial Advisors only)		LEAVE THEM EMPTY OTHERWISE
Short Sale Slot (Institutional only)		1 - If you hold the shares, 2 - if they will be delivered elsewhere. Only for action "SHORT"
Short Sale Location (Institutional only)		If shared will be delivered elsewhere, specify where it can be comma-delimited list, no spaces!
OCA Type		1-Cancel on Fill with Block, 2-Reduce on Fill with Block, 3-Reduce or Fill without Block
Rule BOA		"1"-Indicates "AC" Agency, "N"-AgencyOrderMember, "F"-In-housePTIA, "M"-AgentOrderMemberPTIA, "R"-In-houseRT, "P"-AgencyPT, "W"-AgentOrderMember
Settling Firm (Institutional only)		
All Or None (Default, 1=true)		
Mainstem Qty		
Percent Offset		Reflexive Orders Only
Electronic Trade Only (Default, 1=true) (SMART Routing)		
Firm Quote Only (Default, 1=true) (SMART Routing)		
NBSO Price Cap (SMART Routing)		
Auction Strategy		BOL Exchange: 1-match, 2-improvement, 3-transparent
Starting Price		BOL Exchange
Stock Ref Price		BOL Exchange

## Manually Program Extended Order Attributes

- When appended to orderDescription, the number and order of attributes cannot be changed.
- For any attribute that is not defined, use the value 'EMPTY' or {}. Since a string length is limited to 255 characters, we recommend using the open/close curly braces {}.
- A place order message for a simple stock limit day order looks as follows, with the primary exchange "ISLAND" separating the extended attributes:

[illegible]

Normally, values that you enter on the Extended Order Attributes page apply to all subsequent orders. However, you also can apply selected attributes to an individual order or a group of orders on the Orders page.

**Note:** You can also use this procedure to apply extended order attributes to orders on the Conditional Orders page.

### To apply extended order attributes to individual orders or a group of orders

1. Enter the value or values on the Extended Order Attributes page that you want to apply to an individual order or group of orders.
2. On the Orders page, select the order or group of orders.
3. Click the **Apply Extended Template** button.

The extended order attributes are applied to the order(s) and the values you entered on the Extended Order Attributes page are added to the corresponding fields in the *Extended Order Attributes* section of the Orders page.

When you place the order or group of orders, the extended order attribute values you entered are applied to the order.

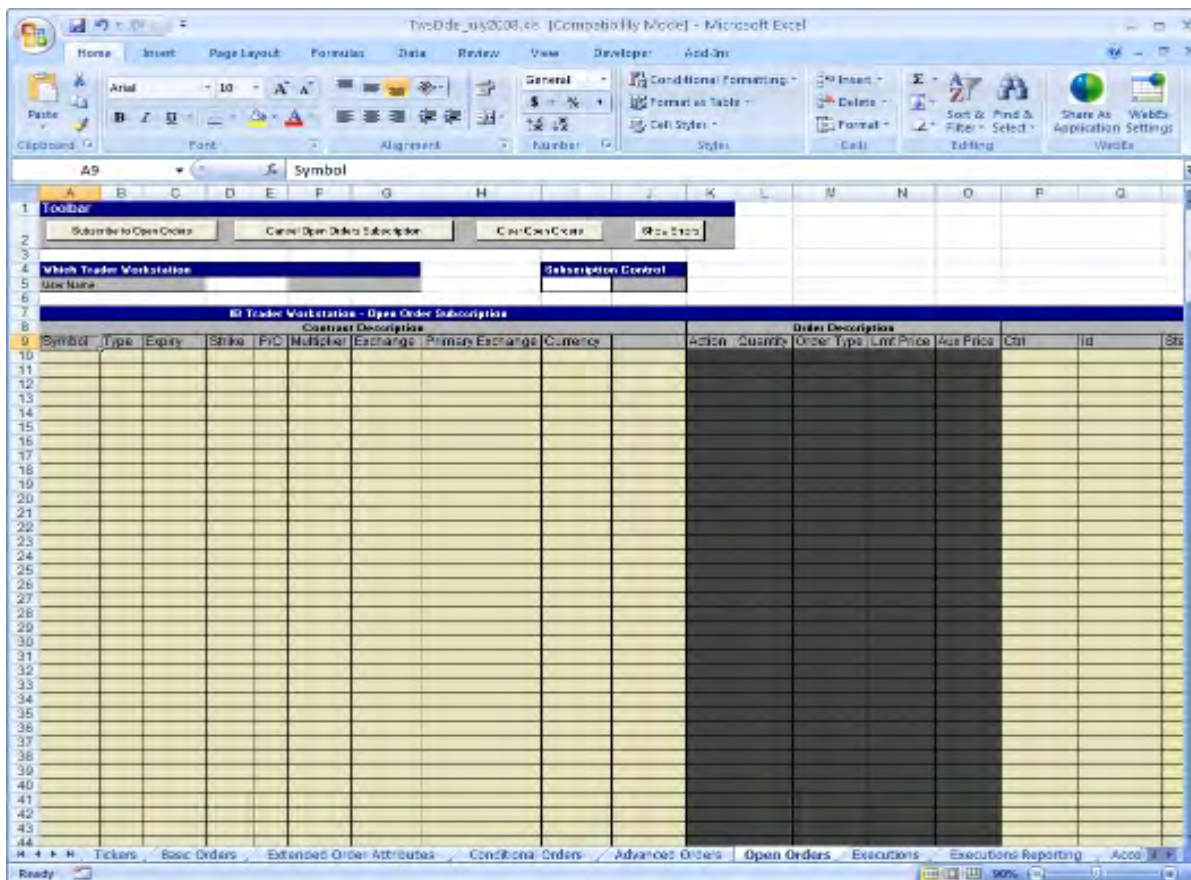
For example, you might want to assign a unique Order Ref number to a group or basket of orders. To do this, you would enter the number for the Order Ref attribute on the Extended Order Attributes page, then select all the orders in the group on the Orders page and click Apply Extended Template.

4. Delete the value of the extended order attributes you used for the order from the Extended Order Attributes page. These values will still apply to all subsequent orders that you place from the ActiveX for Excel API spreadsheet unless you remove the value.

### Open Orders Page

The Open Orders page shows you all transmitted orders, including those that have been accepted by the IB system, and those that are working at an exchange. Once you have subscribed, the page is updated each time you submit a new order, either through the API or in TWS.

Once an order executes, it remains on the Open Orders page for 30 seconds, with the Status value changed to FILLED. Then the filled order is cleared and you can see it on the Executions page if you subscribed to real-time executions.



## Viewing Open Orders

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To view open orders:

1. Click the **Open Orders** tab at the bottom of the spreadsheet.
2. Do one of the following:
  - To request open orders from the current ActiveX for Excel spreadsheet, click **Subscribe to Open Orders** on the toolbar.
  - To request all open orders for the current account, click **Request All Open Orders** on the toolbar.
  - To associate all newly created TWS orders with the current client, click **Request Auto Open Orders** on the toolbar. Note that the Client ID must be 0.

All of the requested open orders are displayed on the page, including orders you enter in the spreadsheet and in TWS.

Orders that fill remain on the page for 30 seconds with a value of *Fill* in the *Status* field.

### To remove open orders

1. Click the **Cancel Open Orders Subscription** button on the toolbar.
2. Click the **Clear Open Orders** button.

## Open Orders Tab Toolbar

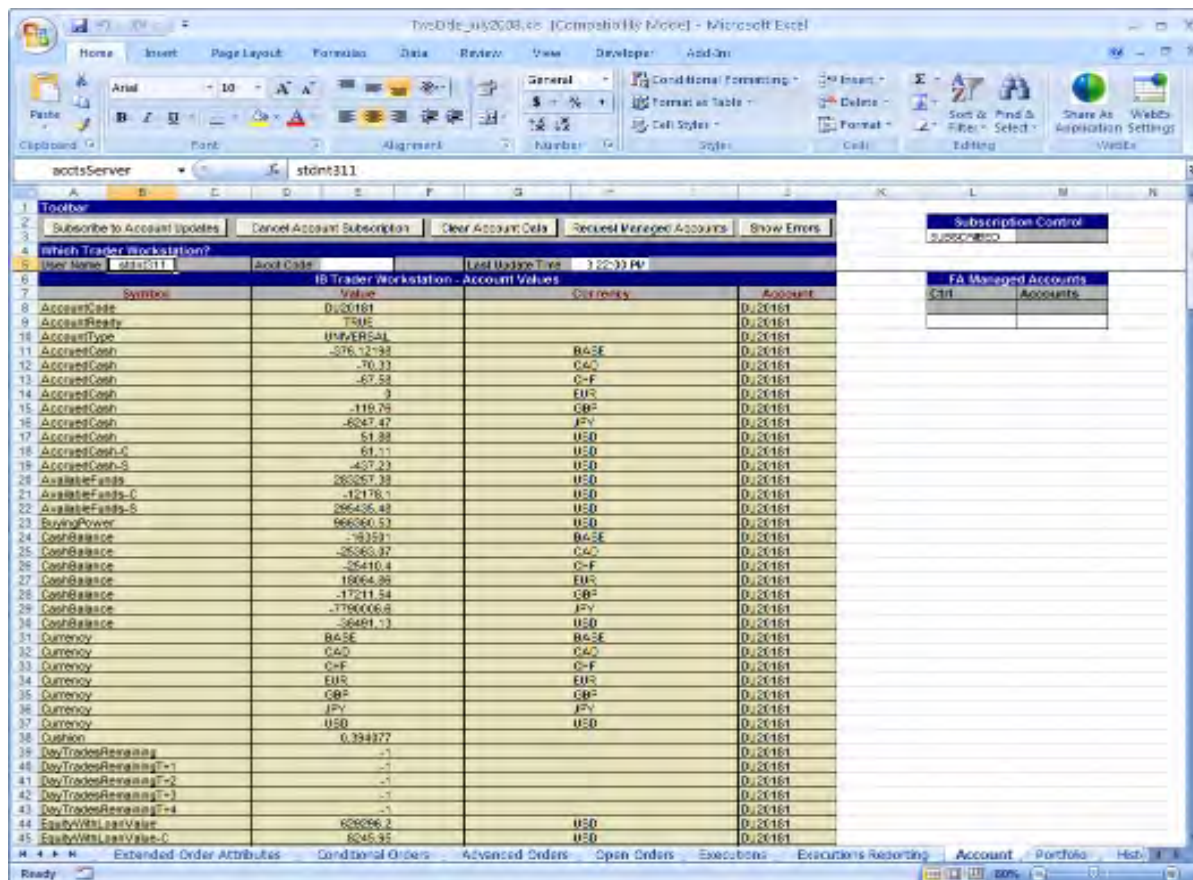
The toolbar on the Open Orders page includes the following buttons:

Button	Description
Request Open Orders	Queries TWS and returns all open orders. Once you subscribe to open orders, this page updates each time there is a new open order.
Request All Open Orders	Queries TWS and returns all open orders from the current account. Once you subscribe to open orders, this page updates each time there is a new open order.
Request Auto Open Orders	Queries TWS and associate all newly created TWS orders with the current client, which must be Client ID 0.
Cancel Auto Open Orders	Cancels association of newly created TWS orders with the client
Clear Open Orders	Removes all open orders from the page.

## Account Page

Use the Account page to:

- View account details including your current Equity with Loan Value and Available funds.
- View list of advisor-managed account codes.
- Financial Advisors can view FA information.
- View your current portfolio.



## Using the Account Page

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To view account information

1. Click the **Account** tab at the bottom of the spreadsheet.
2. Click **Request Account Updates** on the toolbar.

### To remove account information

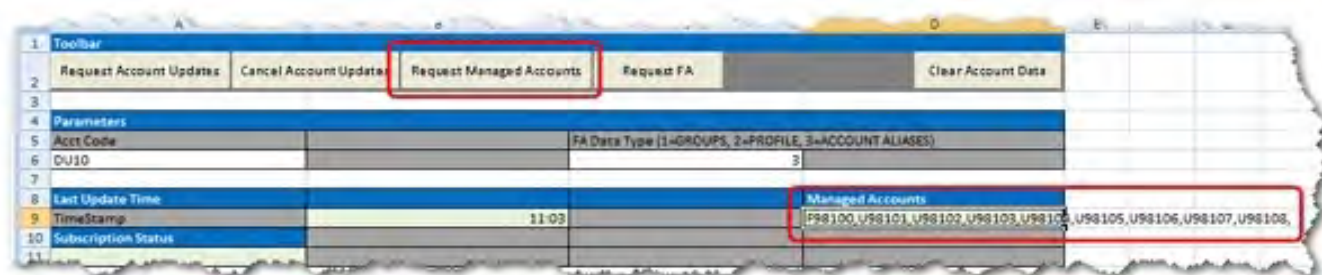
1. Click the **Account** tab at the bottom of the spreadsheet.
2. Click **Cancel Account Updates** on the toolbar to stop receiving account updates.
3. Click **Clear Account Data** on the toolbar to clear all data from the page.

### To request the list of Financial Advisor (FA) managed account codes

1. Click the **Account** tab at the bottom of the spreadsheet.
2. Click **Request Managed Accounts** on the toolbar.

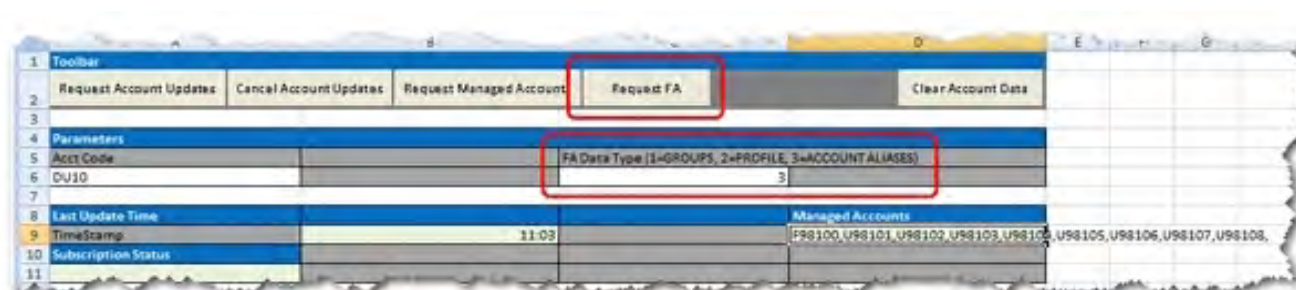
A comma-separated list of all managed account numbers displays in the *Managed Accounts* cell.





### To request Financial Advisor (FA) information

1. Click the **Account** tab at the bottom of the spreadsheet.
2. In the *Account Code* cell, type the account code for which you want details.
3. In the *FA Data Type* cell, enter a numeric value representing the type of data you wish you receive:
  - Type **1** for groups.
  - Type **2** for profiles.
  - Type **3** for account aliases.
4. Click **Request Managed Accounts** on the toolbar.



### Account Page Toolbar Buttons

The toolbar on the Account page includes the following buttons.

Button	Description
Request Account Updates	Each click gives you data for a specific account value. All blank lines that precede the Account Portfolio section will hold data. Continue to click until all lines are populated.
Cancel Account Updates	Click this button one time for each position you hold. When you get a line of "0's" you know you have downloaded all current positions. These values continue to update in real-time.
Request Managed Accounts	For advisor accounts, receives a list of managed accounts and displays them as a comma-separated list in the <i>Management Accounts</i> cell.

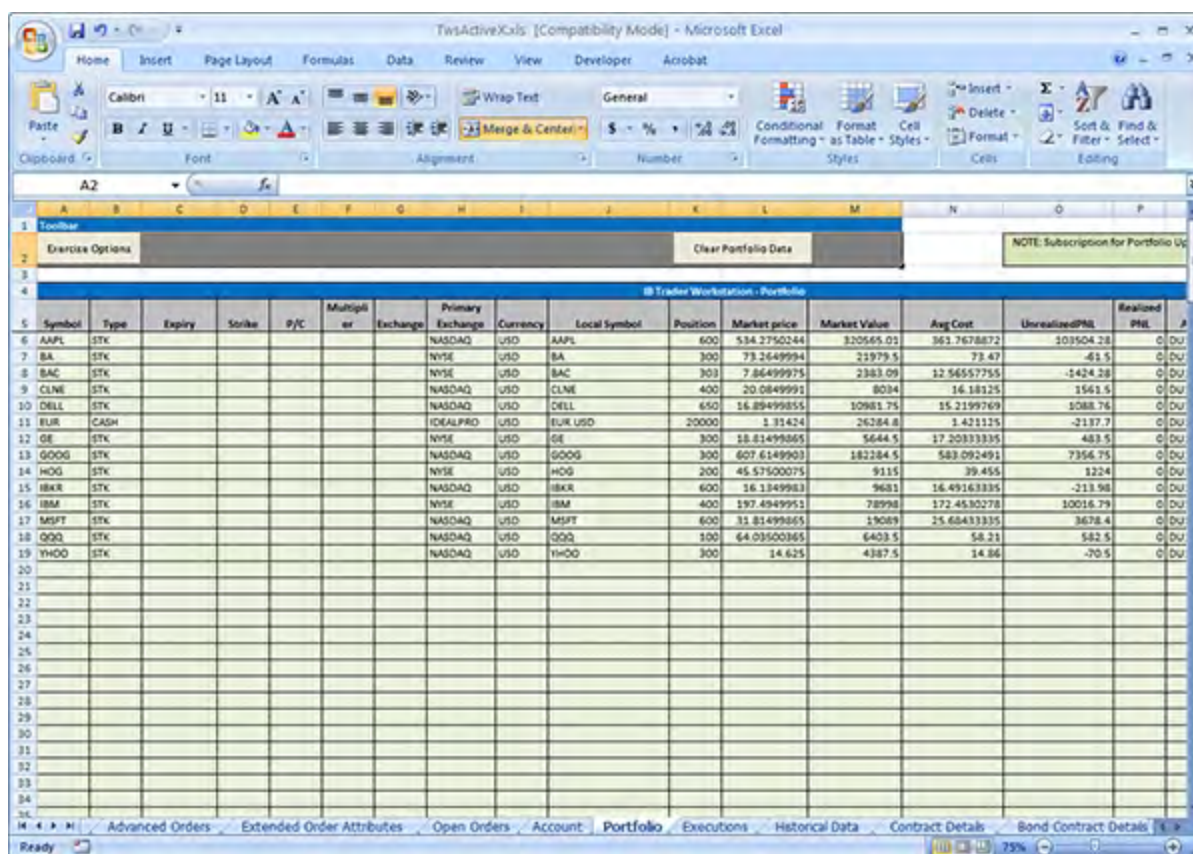
Request FA	For advisor accounts, displays FA data of the type specified in the <i>FA Data Type</i> cell.
Clear Account Data	Clears all information from the page. You must first cancel your subscription before you can clear the data.

## Portfolio Page

Use the Portfolio page to:

- Displays all of your current positions.
- Exercise options.

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.



## Viewing Your Portfolio

### To view your portfolio

1. Click the **Account** tab on the bottom of the worksheet, then click **Request Account Updates** on the toolbar.
2. Click the **Portfolio** tab at the bottom of the worksheet to view your portfolio.

### To remove portfolio information

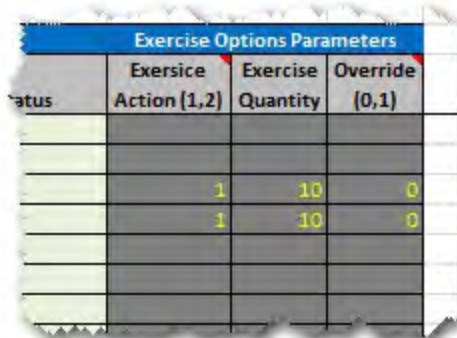
1. Click the **Account** tab on the bottom of the worksheet, then click **Cancel Account Updates** on the toolbar to stop receiving portfolio updates.
2. Click the **Clear Portfolio Data** button to clear all data from the page.

### Exercising Options

You can exercise options or let options lapse on the Portfolio page.

#### To exercise an option or let an option lapse

1. Click the **Portfolio** tab at the bottom of the worksheet.
2. Enter values in the *Exercise Options Parameters* cells at the far right side of the page:
  - *Exercise Action* - Enter **1** to exercise the selected option; **2** to let the option lapse.
  - *Exercise Quantity* - Enter the number of contracts you wish to exercise or let lapse.
  - *Override* - Enter **1** to override the system's natural action; **2** to not override.



Exercise Options Parameters			
Status	Exercise Action (1,2)	Exercise Quantity	Override (0,1)
	1	10	0
	1	10	0

3. Click **Exercise Options** in the toolbar. The *Status* column updates.

### Portfolio Page Toolbar Buttons

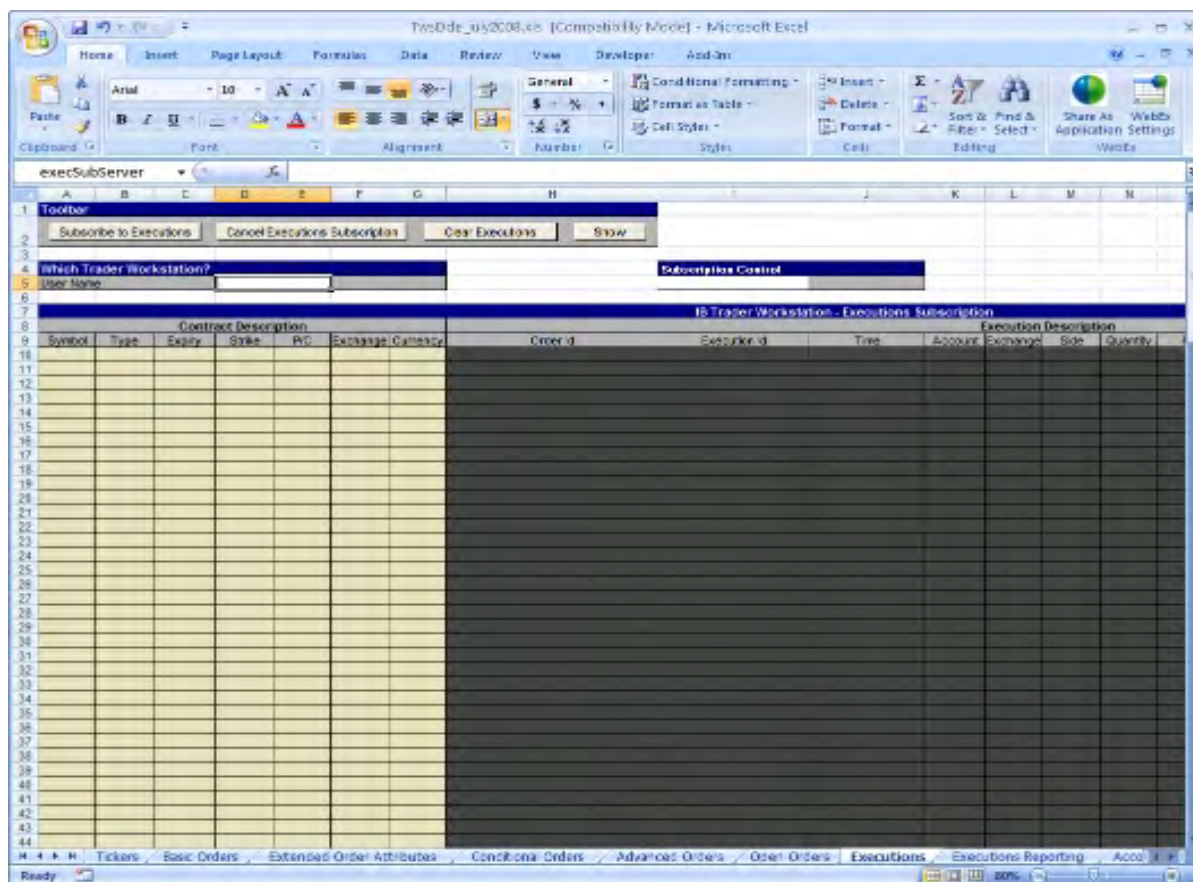
The toolbar on the Portfolio page includes the following buttons.

Button	Description
Exercise Options	Exercises the selected option or lets the selected option lapse, depending on the value in the <i>Exercise Action</i> cell.
Clear Portfolio Data	Removes all data from the page.

### Executions Page

When you subscribe to executions, the Executions page displays information about all completed trades.



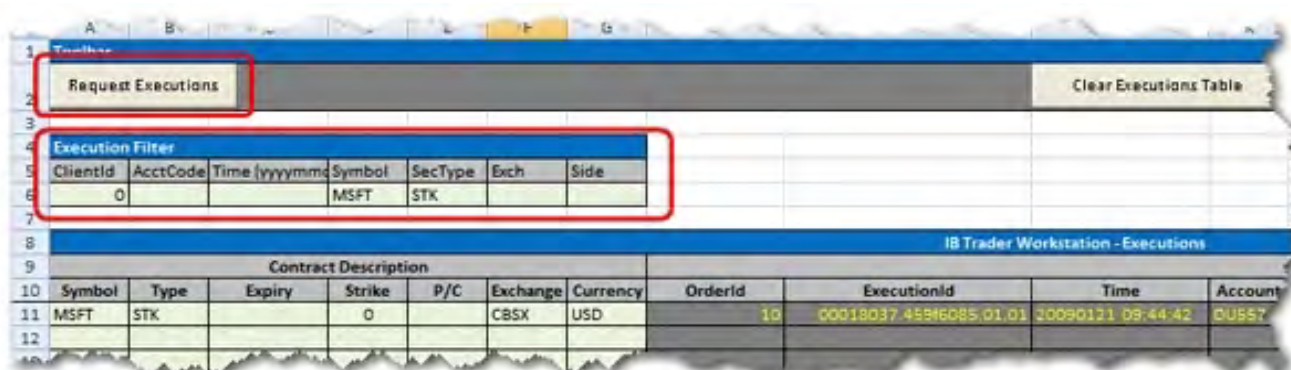


## Viewing Executions

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To view executions

1. Click the **Executions** tab at the bottom of the spreadsheet.
2. Optionally, filter your executions by entering values in the *Execution Filter* cells:
  - Filter executions by client ID, account code, date/time, symbol, security type, exchange or side.



3. Click **Request Executions** on the toolbar.

### To remove execution data

1. Click **Clear Executions Table** on the toolbar. All data is removed from the page.

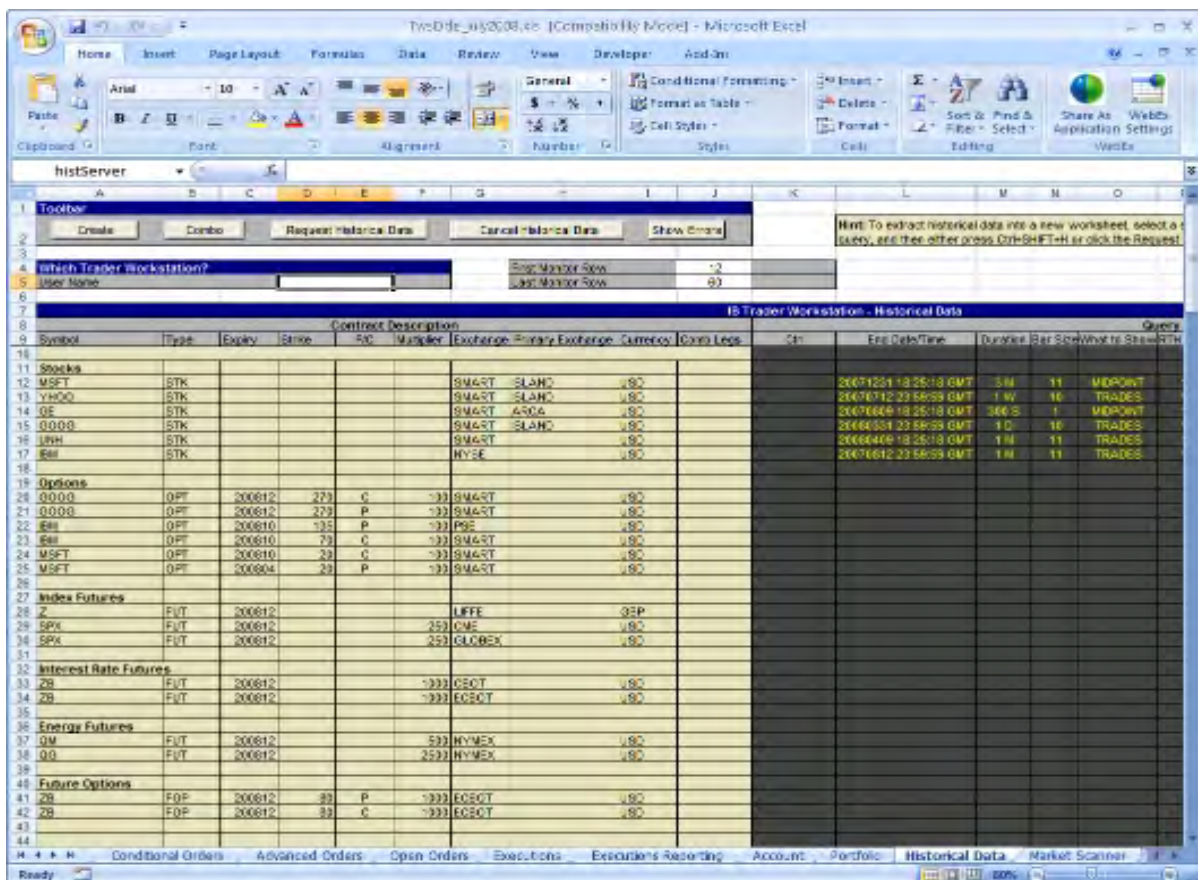
## Executions Page Toolbar Buttons

The toolbar on the Executions page includes the following buttons:

Button	Description
Request Executions	Queries TWS and returns information about all valid executions. After you subscribe to executions, this page updates each time an order executes.
Clear Executions Table	Removes all execution reports from the page.

## Historical Data Page

Use the Historical Data page to request historical data for an instrument based on data you enter in query fields. The query results display on a separate worksheet page and creates a new page for the results if the page doesn't currently exist.



**Note:** For a information about historical data request limitations, see [Historical Data Limitations](#).

## Viewing Historical Data

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To request historical data

1. Click the Historical Data tab at the bottom of the spreadsheet.
2. Create a ticker by filling in the fields in the *Contract Description* section of the page, or by clicking the **Create Ticker** button on the toolbar and [entering the required information in the Ticker box](#).
3. Enter the parameters of your query in the *Query Specification* fields. For complete descriptions of the query fields, [Historical Data Page Query Specification Fields](#).
4. Select the line, then click the **Request Historical Data** button. The status of your request displays in the *Request Status* cell.

In the *Activate Page* cell, enter **TRUE** to display the results page on top of the current window. Enter **FALSE** to display the page on a separate tab in the spreadsheet without displaying on top of the current window.

The results are displayed on a new tabbed page in the spreadsheet, the name of which is specified in the *Page Name* cell.

### To request historical data for expired contracts

1. On the Historical Data page, create a ticker by filling in the fields in the *Contract Description* section of the page, or by clicking the **Create Ticker** button on the toolbar and entering the required information in the Ticker box.
2. Enter the parameters of your query in the *Query Specification* fields.
3. In the *Incl Expired* cell in the Query Specification section, enter **TRUE**.
4. Select the line, then click **Request Historical Data** on the toolbar. The status of your request displays in the *Request Status* cell.

In the *Activate Page* cell, enter **TRUE** to display the results page on top of the current window. Enter **FALSE** to display the page on a separate tab in the spreadsheet without displaying on top of the current window.

The results are displayed on a new tabbed page in the spreadsheet, the name of which is specified in the *Page Name* cell.

5. Historical data queries on expired contracts are limited to the last year of the life of the contract.

The following figure shows a typical historical data results page.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1																	
2	Date/Time	Open	High	Low	Close	Volume	Count	WAP	HasGaps								
3	20081003	105.19	110.94	103.05	103.76	-1	-1	-1	0								
4	20081006	100.76	102.89	96.79	101.35	-1	-1	-1	0								
5	20081007	102.03	102.39	94.24	95.81	-1	-1	-1	0								
6	20081008	91.81	97	90.01	90.11	-1	-1	-1	0								
7	20081009	94.5	95.05	88.28	89.48	-1	-1	-1	0								
8	20081010	88.79	91.42	83.55	88.2	-1	-1	-1	0								
9	20081013	90.12	93.27	86.98	93.04	-1	-1	-1	0								
10	20081014	97.57	98.63	92.02	93.56	-1	-1	-1	0								
11	20081015	94.14	94.83	87.96	88.06	-1	-1	-1	0								
12	20081016	88.33	91.99	84.36	91.46	-1	-1	-1	0								
13	20081017	91.14	95.89	89.54	90.73	-1	-1	-1	0								
14	20081020	92.29	93.29	89.33	92.7	-1	-1	-1	0								
15	20081021	92.13	92.34	88.65	88.67	-1	-1	-1	0								
16	20081022	87.15	87.25	80.86	83.36	-1	-1	-1	0								
17	20081023	84.64	85.87	81.01	84.42	-1	-1	-1	0								
18	20081024	80	84.3	78.86	81.89	-1	-1	-1	0								
19	20081027	79.94	84.46	79.04	79.61	-1	-1	-1	0								
20	20081028	81.9	87.56	79.55	87.51	-1	-1	-1	0								
21	20081029	87.6	90.99	86.28	86.38	-1	-1	-1	0								
22	20081030	90.35	92.41	88.92	90.78	-1	-1	-1	0								
23	20081031	90.95	94.17	89.08	93.15	-1	-1	-1	0								
24	20081103	93.01	94.64	92.25	92.71	-1	-1	-1	0								
25	20081104	93.85	94.23	92.09	93.44	-1	-1	-1	0								
26	20081105	92.79	93.39	89.73	89.88	-1	-1	-1	0								
27	20081106	88.25	88.83	84.29	85.22	-1	-1	-1	0								
28	20081107	86.01	86.71	84.28	86.31	-1	-1	-1	0								
29	20081110	87.81	88.04	82.74	84.01	-1	-1	-1	0								
30	20081111	83.2	83.97	80.26	82.65	-1	-1	-1	0								

## Historical Data Page Query Specification Fields

Parameter	Description
End Date/Time	Use the format <code>yyyymmdd {space}hh:mm:ss{space}tmz</code> where the time zone is allowed (optionally) after a space at the end.
Duration	<p>This is the time span the request will cover, and is specified using the format <code>integer {space} unit</code>, where valid units are:</p> <ul style="list-style-type: none"> <li>• S (seconds)</li> <li>• D (days)</li> <li>• W (weeks)</li> <li>• Y (years)</li> </ul> <p>This unit is currently limited to one. If no unit is specified, seconds are used.</p>

Bar Size	<p>Specifies the size of the bars that will be returned. The following bar sizes may be used, and are specified using the parametric value:</p> <table> <tr> <th>Bar Size String</th><th>Integer Value</th></tr> <tr> <td>1 second</td><td>1</td></tr> <tr> <td>5 seconds</td><td>2</td></tr> <tr> <td>15 seconds</td><td>3</td></tr> <tr> <td>30 seconds</td><td>4</td></tr> <tr> <td>1 minute</td><td>5</td></tr> <tr> <td>2 minutes</td><td>6</td></tr> <tr> <td>3 minutes</td><td>16</td></tr> <tr> <td>5 minutes</td><td>7</td></tr> <tr> <td>15 minutes</td><td>8</td></tr> <tr> <td>30 minutes</td><td>9</td></tr> <tr> <td>1 hour</td><td>10</td></tr> <tr> <td>1 day</td><td>11</td></tr> </table> <p>On the query return page, each "bar" is represented by a line in the spreadsheet. If you specify a duration of 300 seconds, and a bar size of "1" (one second) your return will include 300 lines, and the value in each line is equal to one second, or is a one-second bar. Note that you can use either the Integer value of the Bar Size String or the Integer Value to define the bar sizes.</p>	Bar Size String	Integer Value	1 second	1	5 seconds	2	15 seconds	3	30 seconds	4	1 minute	5	2 minutes	6	3 minutes	16	5 minutes	7	15 minutes	8	30 minutes	9	1 hour	10	1 day	11
Bar Size String	Integer Value																										
1 second	1																										
5 seconds	2																										
15 seconds	3																										
30 seconds	4																										
1 minute	5																										
2 minutes	6																										
3 minutes	16																										
5 minutes	7																										
15 minutes	8																										
30 minutes	9																										
1 hour	10																										
1 day	11																										



What to Show	<p>Determines the nature of the data extracted. Valid values include:</p> <ul style="list-style-type: none"> <li>• Trades</li> <li>• Midpoint</li> <li>• Bid</li> <li>• Ask</li> <li>• Bid/Ask</li> </ul> <p>All but the Bid/Ask data contain the <i>start time</i>, <i>open</i>, <i>high</i>, <i>low</i>, <i>close</i>, <i>volume</i> and <i>weighted average price</i> during the time slice queried.</p> <p>For the Bid/Ask query, the <i>open</i> and <i>close</i> values are the time-weighted average bid and the time-weighted average offer, respectively. These bars are identical to the TWS charts' candlestick bars.</p>
RTH Only	<p>Regular Trading Hours only. Valid values include:</p> <ul style="list-style-type: none"> <li>• 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.</li> <li>• 1 - only data within the regular trading hours for the product requested is returned, even if the time span falls partially or completely outside.</li> </ul>
Date Format Style	<p>Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 - dates that apply to bars are returned in the format <code>yyyymmdd {space} {space} hh:mm:dd</code> (the same format used when reporting executions).</li> <li>• 2 - the dates are returned as an integer specifying the number of seconds since 1/1/1970 GMT.</li> </ul>
Page Name	The name of the results page. This appears in the tab for the results page at the bottom of the worksheet.
Activate page	Enter TRUE to display the results page on top of the current window. Enter FALSE to display the results on a new page in the spreadsheet without appearing on top of the current window.

Note that the new page is added to the right of the existing tabs on the worksheet.

### Historical Data Page Toolbar Buttons

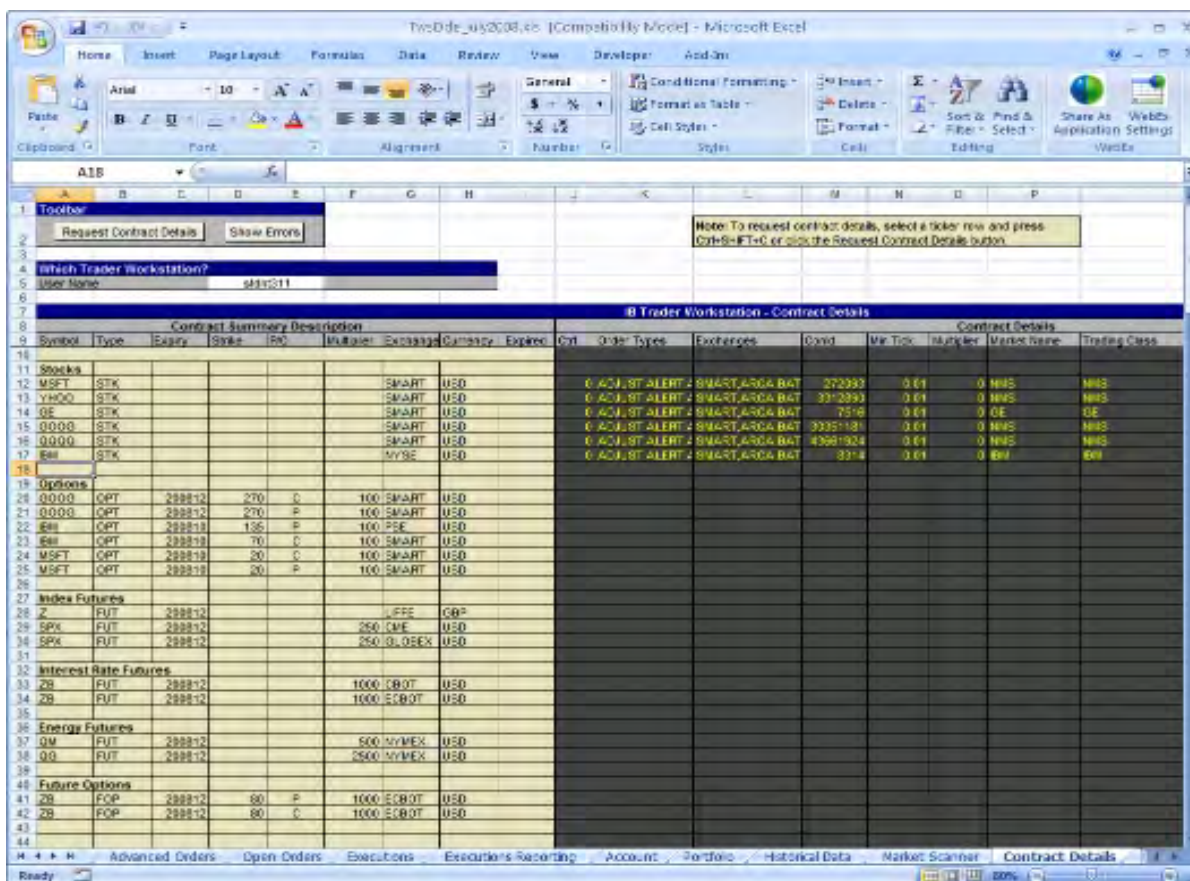
The toolbar on the Historical Data page includes the following buttons.

Button	Description
--------	-------------

Create Ticker	Opens the <a href="#">Ticker box</a> . Enter information to create a market data line.
Combo Legs	Opens the <a href="#">Combination Legs box</a> . Enter contract details to create legs of a combination order one by one.
Request Historical Data	Submits your historical data query to TWS and displays the results on a separate worksheet page.
Cancel Historical Data	Cancels the historical data request.

## Contract Details Page

Use the Contract Details page to request contract-specific information such as supported order types, valid exchanges, the contract ID, and so on.



## Requesting Contract Details

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

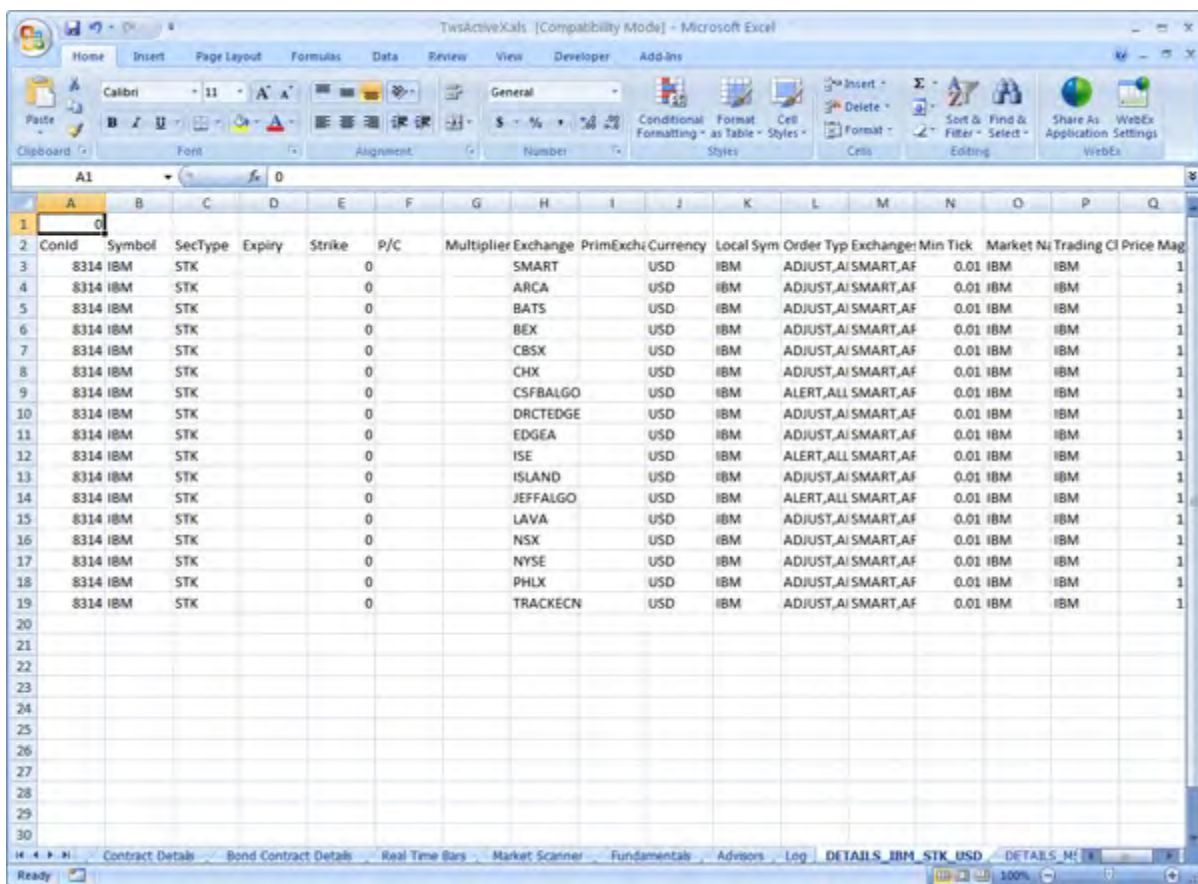
To request details for a contract

1. Click the **Contract Details** tab at the bottom of the spreadsheet to open the Contract Details page.
2. Select or enter the ticker symbol for which you want to request contract details.
3. To request contract details for an expired contract, type **TRUE** in the *Incl Expired* cell.
4. Select the row, then click **Request Contract Details** on the toolbar. The status of your request displays in the *Request Status* cell.

In the *Activate Page* cell, enter **TRUE** to display the results page on top of the current window. Enter **FALSE** to display the page on a separate tab in the spreadsheet without displaying on top of the current window.

The results are displayed on a new tabbed page in the spreadsheet, the name of which is specified in the *Page Name* cell.

The following figure shows a typical contract details page.



### Contract Details Page Toolbar Buttons

The toolbar on the Contract Details page includes the following button:

Button	Description
Request Contract Details	Returns information on the selected contract.

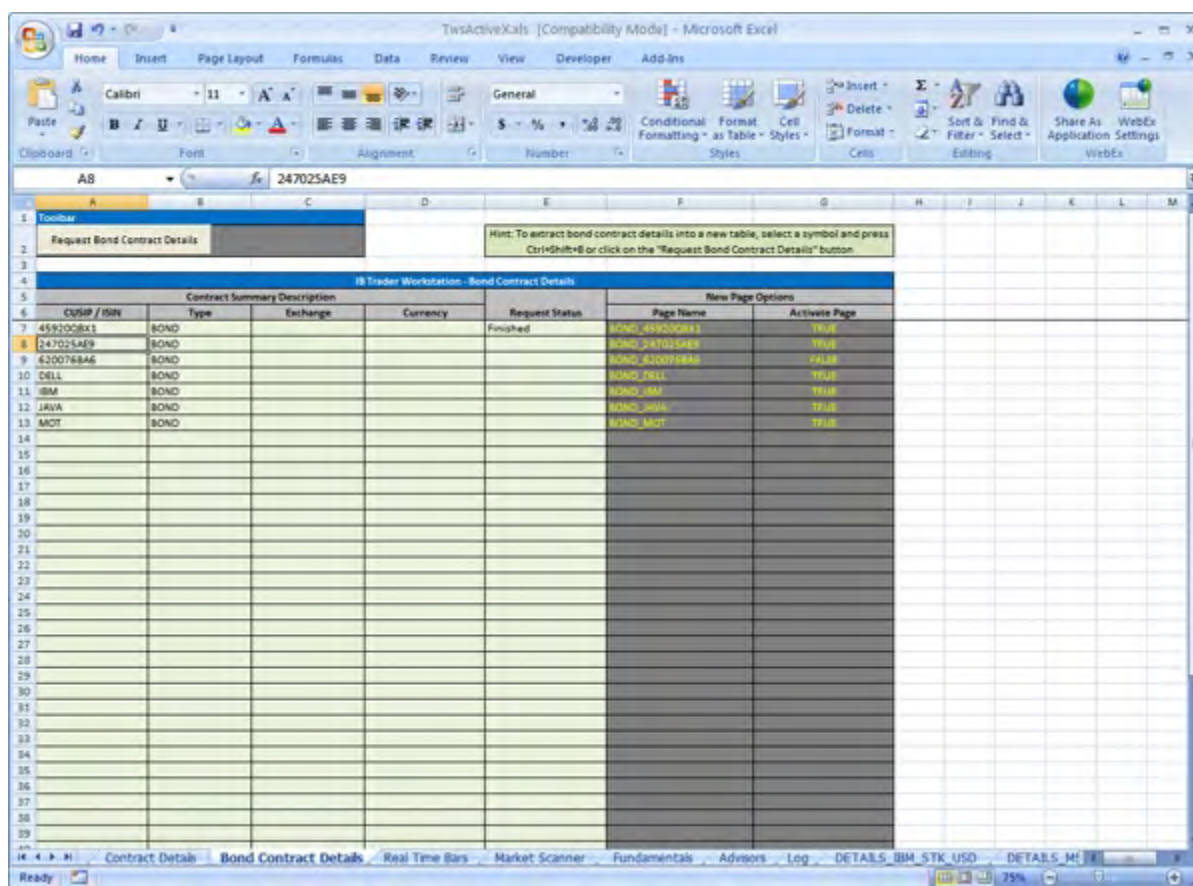


## Bond Contract Details Page

Use the Bond Contract Details page to request contract-specific information for bonds, including the coupon, ratings, bond type, maturity date, and so on.

**Note:** Beginning with TWS Version 921, some bond contract data will be suppressed and will not be available from the API. All bond contract data will continue to be available from Trader Workstation, but only the following bond contract data will be available from the API:

- Contract ID
- Minimum Tick
- CUSIP (if you have subscribed to the CUSIP service)
- Rating (if you have subscribed to ratings)



## Requesting Bond Contract Details

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To request details for a bond contract

1. Click the **Bond Contract Details** tab at the bottom of the spreadsheet to open the Bond Contract Details page.
2. Select or enter the ticker symbol for which you want to request bond contract details.

3. Select the row, then click **Request Bond Contract Details** on the toolbar. The status of your request displays in the *Request Status* cell.

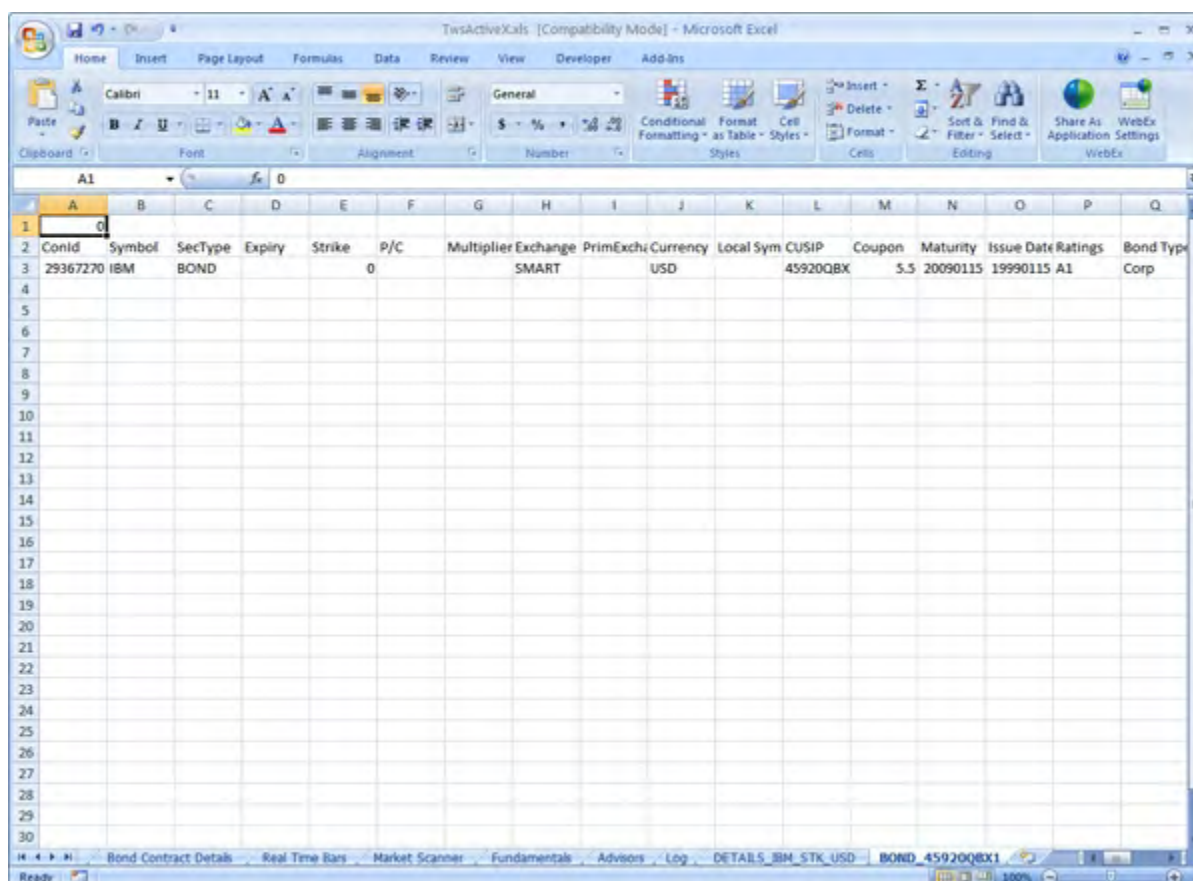
In the *Activate Page* cell, enter **TRUE** to display the results page on top of the current window. Enter **FALSE** to display the page on a separate tab in the spreadsheet without displaying on top of the current window.

The results are displayed on a new tabbed page in the spreadsheet, the name of which is specified in the *Page Name* cell.

**Note:** Beginning with TWS Version 921, some bond contract data will be suppressed and will not be available from the API. All bond contract data will continue to be available from Trader Workstation, but only the following bond contract data will be available from the API:

- Contract ID
- Minimum Tick
- CUSIP (if you have subscribed to the CUSIP service)
- Rating (if you have subscribed to ratings)

The following figure shows a typical bond contract details page.



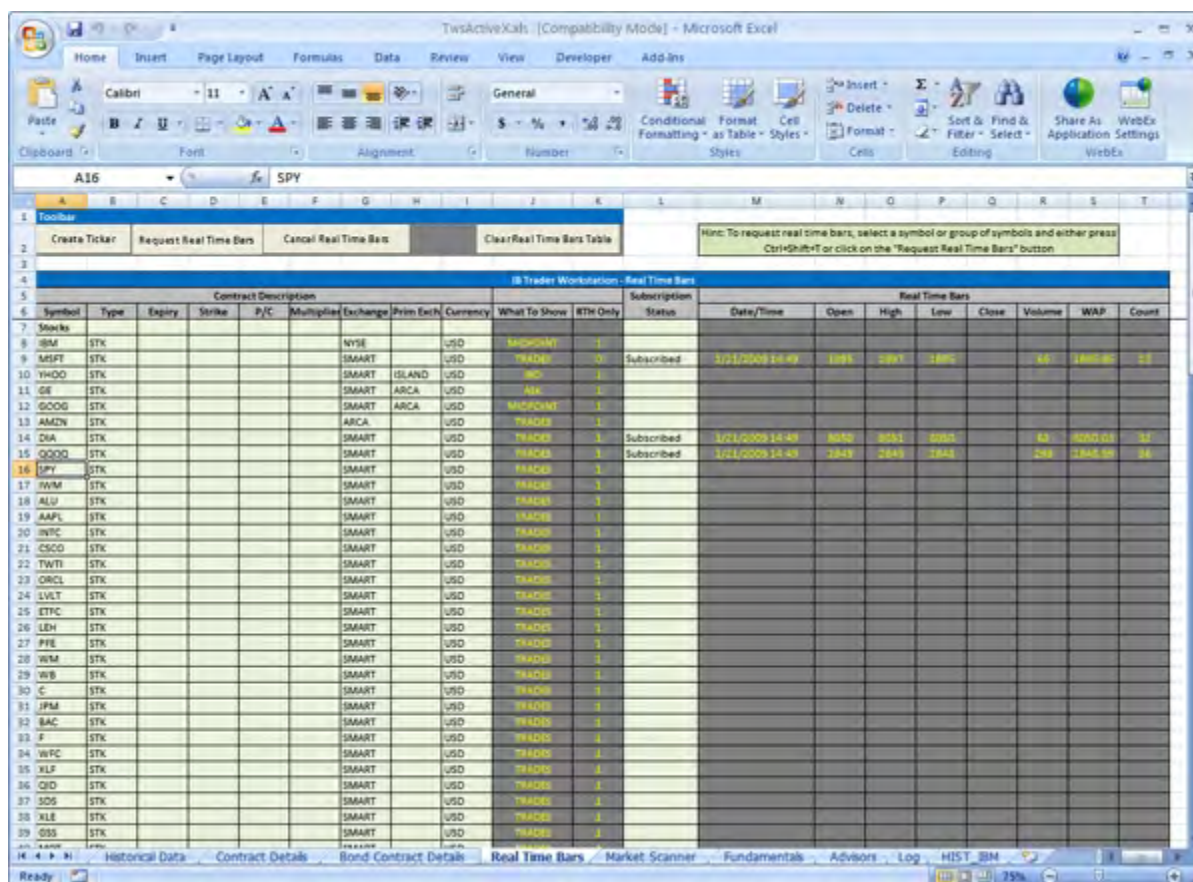
## Bond Contract Details Page Toolbar Buttons

The toolbar on the Bond Contract Details page includes the following button:

Button	Description
Request Bond Contract Details	Gets bond information data for the selected contract.

## Real Time Bars Page

Real time bars allow you to get a summary of real-time market data every five seconds, including the opening and closing price, and the high and the low within that five-second period (using TWS charting terminology, we call these five-second periods "bars"). You can also get data showing trades, midpoints, bids or asks. You request real time bars on the Real Time Bars page, which is shown below.



### To request real time bars

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

1. Click the **Real Time Bars** tab at the bottom of the spreadsheet to open the Real Time Bars page.
2. Select or enter the ticker symbol for which you want to request real time bars.
3. Enter the following information for the selected row:

- In the *What to Show* cell, enter TRADES, BID, ASK or MIDPOINT.
  - In the *RTH Only* cell, enter **0** to return all data available during the time span requested, including time intervals when the market in question was outside of regular trading hours. Enter **1** to return only data within the regular trading hours for the product requested is returned, even if the time span falls partially or completely outside.
4. Select the row, then click **Request Real Time Bars** on the toolbar. The status of your request displays in the *Subscription Status* cell.

Results are displayed in the *Real Time Bars* cells on the right side of the page.

### Real Time Bars Page Toolbar Buttons

The toolbar on the Real Time Bars page includes the following buttons.

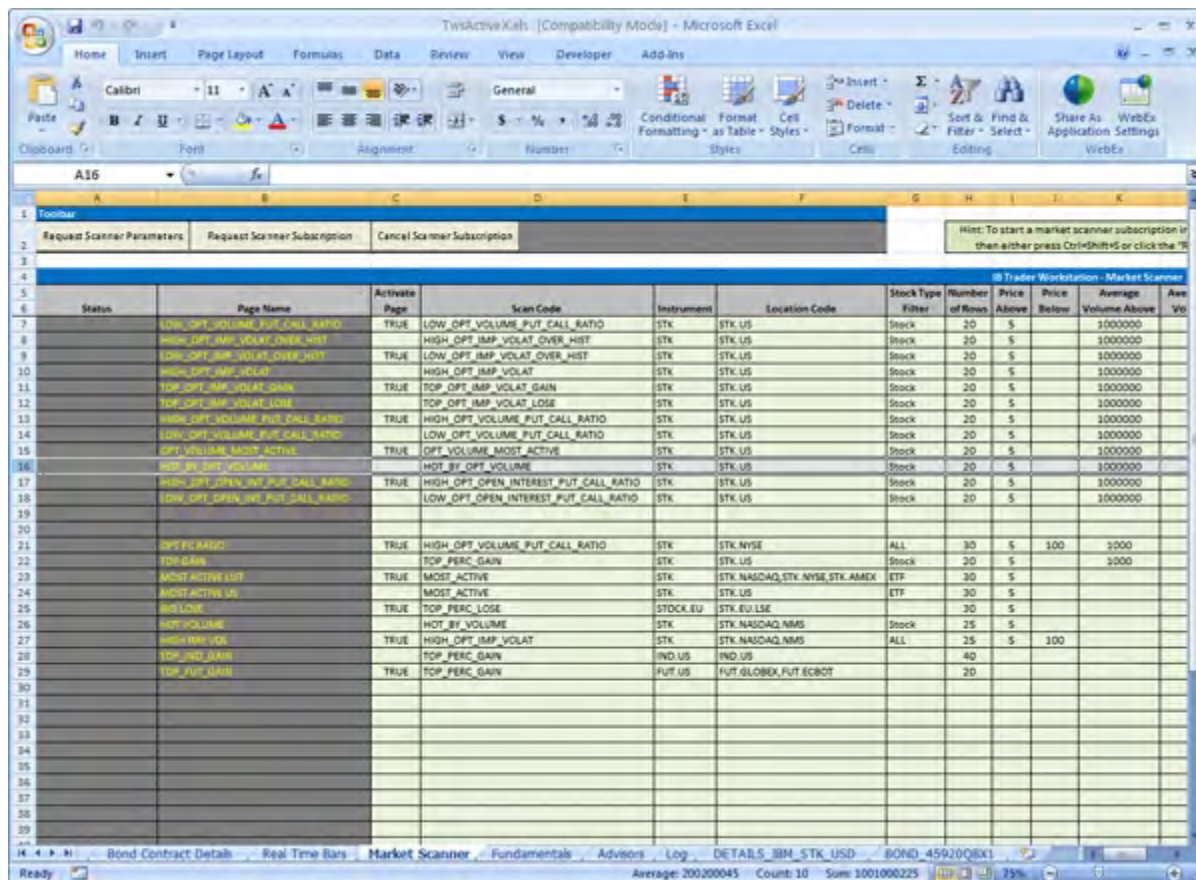
Button	Description
Create Ticker	Opens the Ticker box. Enter information to create a market data line.
Request Real Time Bars	Submits your real time bars request to TWS and displays the results in the dark gray cells the right side of the page.
Cancel Real Time Bars	Cancels the real time bars request.
Clear Real Time Bars Table	Clears all real time bar data from the page.

### Market Scanner Page

Use the Market Scanner page to subscribe to TWS market scanners. These scanners allow you to define criteria and set filters that return the top  $x$  number of underlyings which meet all scan criteria. The scan is continually updated in real time.

You can also display market scanner parameters from this page.





## Starting a Market Scanner Subscription

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

### To start a scanner subscription

1. Click the **Market Scanner** tab at the bottom of the spreadsheet.
2. Highlight an existing scanner row, or enter information for a different market scanner:
  - a. Type the name of the scan results page in the *Page Name* cell.
  - b. Type **TRUE** or **FALSE** in the *Activate Page* cell.

Setting this cell to TRUE forces the scan results page to pop to the front of your application every time it updates. To stop this behavior, set the value of this field to FALSE.

- c. Type values for the rest of the scan parameters in the lightly shaded section of the page. You can get all of the scan codes from the market scanner parameters.
3. Click **Request Scanner Subscription** on the toolbar. A new page for the scanner is created and is displayed after the subscription is processed.

## Market Scanner Parameters

You can display all of the market scanner parameters from the Market Scanner page. Scanner parameters are returned to the spreadsheet from TWS as an XML file.

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

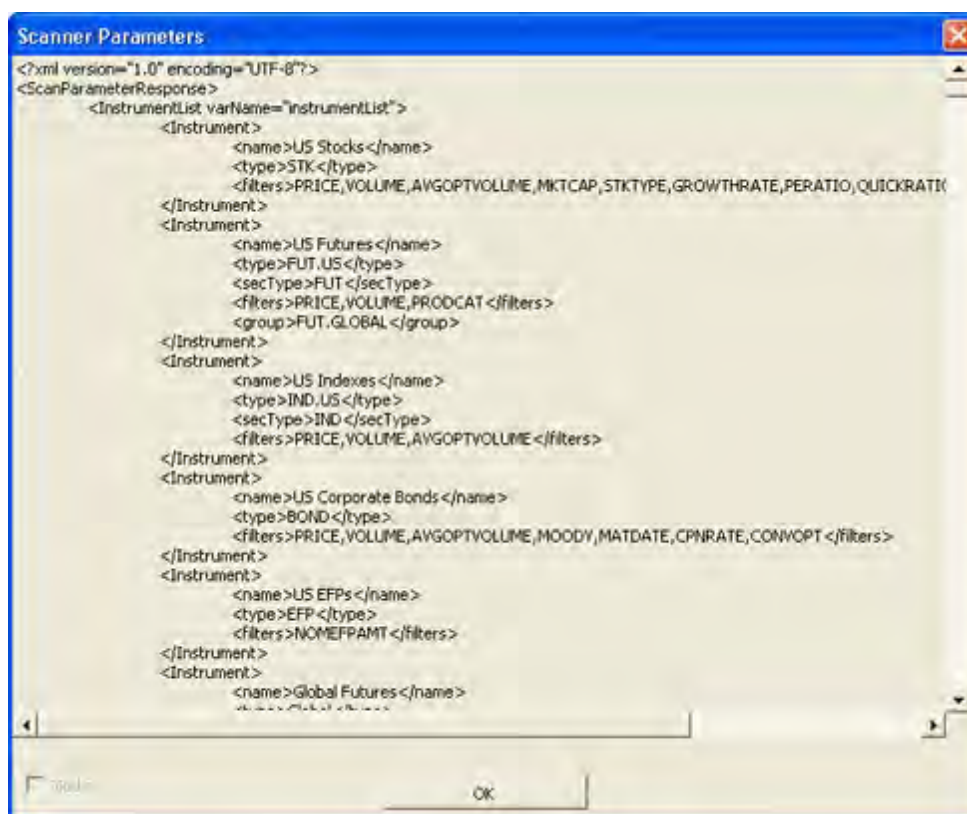
### To view scanner parameters

1. Click the **Market Scanner** tab at the bottom of the spreadsheet.
2. Click **Request Scanner Parameters** on the toolbar.

The entire scanner parameters XML file is displayed in a window.

3. To save the parameters in a convenient file on your computer, manually select part or all of the contents of the XML file in the Scanner Parameters window, then copy and paste it into a separate text document.
4. Click **OK** to close the Scanner Parameters window.

The Scanner Parameters window is shown on the next page.



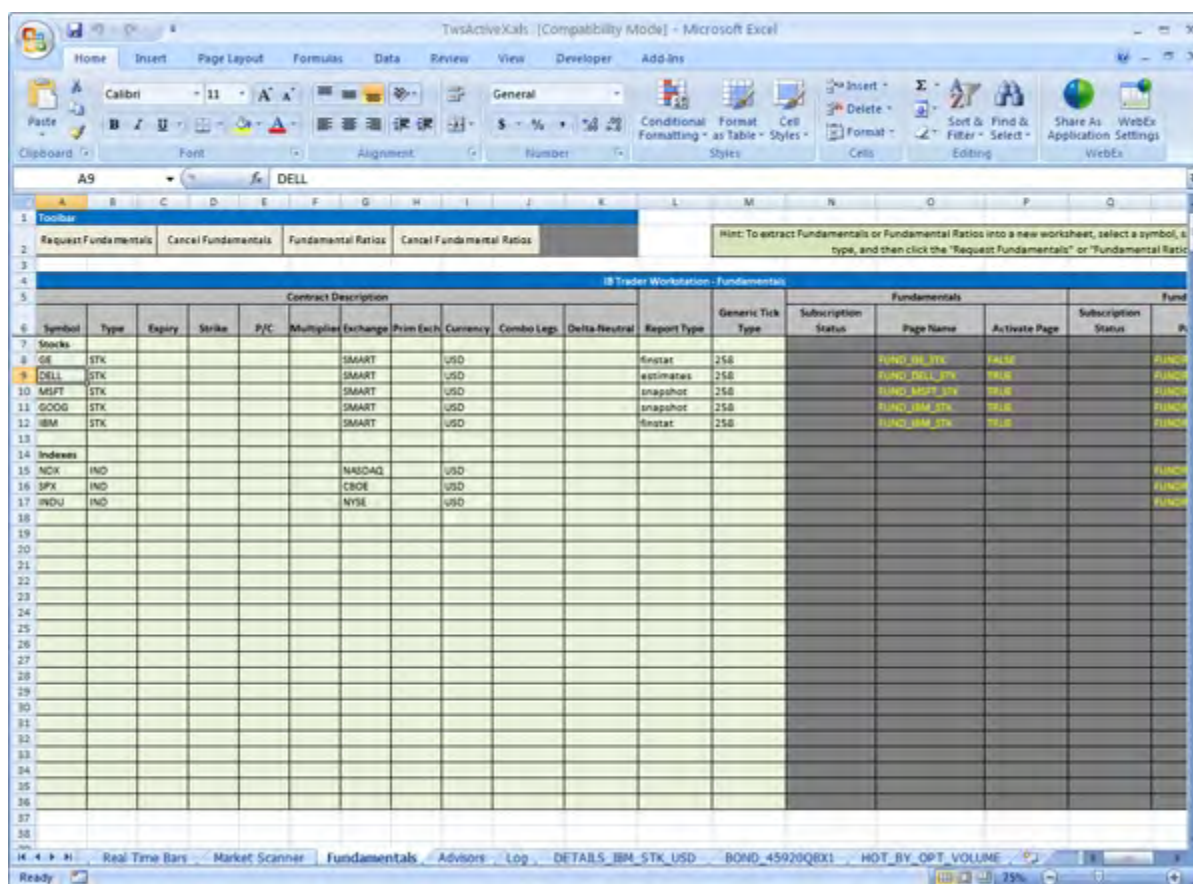
## Market Scanner Page Toolbar Buttons

The toolbar on the Market Scanner page includes the following buttons.

Button	Description
Request Scanner Parameters	Displays all scanner parameters in an XML file in a separate window.
Request Scanner Subscription	Creates and displays a new page for results of the selected market scanner.
Cancel Scanner Subscription	Cancels the market scanner.

## Fundamentals Page

Use the Fundamentals page to receive Reuters global fundamental data and fundamental ratios. There must be a paid subscription to Reuters Fundamental set up in Account Management before you can receive this data.



### To receive Reuters global fundamental data and fundamental ratios

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

1. Click the **Fundamentals** tab at the bottom of the spreadsheet to display the Fundamentals page.
2. Enter information about the contract for which you want fundamentals data or ratios into the *Contract Description* cells.
3. In the *Report Type* cell, enter the type of report you wish to view:
  - **finstat** - Financial Statement
  - **estimates** - Estimates
  - **snapshot** - Summary
4. In the *Generic Tick Type* cell, enter **258** as the tick type value. For details on generic tick type values, see [Generic Tick Types](#).
5. Enter the following information in the *Fundamentals* section of the page for fundamental data, or the *Fundamental Ratios* section for fundamental ratios:
  - a. Fundamental data and ratio results display on a new page in the spreadsheet. Type the name of the results page in the *Page Name* cell.
  - b. Type **TRUE** or **FALSE** in the *Activate Page* cell.

Setting this cell to TRUE forces the results page to pop to the front of your application every time it updates. To stop this behavior, set the value of this field to FALSE.
6. Do one of the following:
  - Click **Request Fundamentals** on the toolbar to view fundamentals data. A new page for the results is created and is displayed after the request is processed.
  - Click **Fundamentals Ratios** on the toolbar to view fundamentals ratios. A new page for the results is created and is displayed after the request is processed.

The status of your request appears in the *Subscription Status* cell.

## Fundamentals Page Toolbar Buttons

The toolbar on the Market Scanner page includes the following buttons.

Button	Description
Request Fundamentals	Requests fundamentals data, which displays on a new page in the spreadsheet based on the information you enter on the page.
Cancel Fundamentals	Cancels fundamentals data.
Fundamental Ratios	Requests fundamentals ratios, which displays on a new page in the spreadsheet based on the information you enter on the page.
Cancel Fundamental Ratios	Cancels fundamentals ratios.

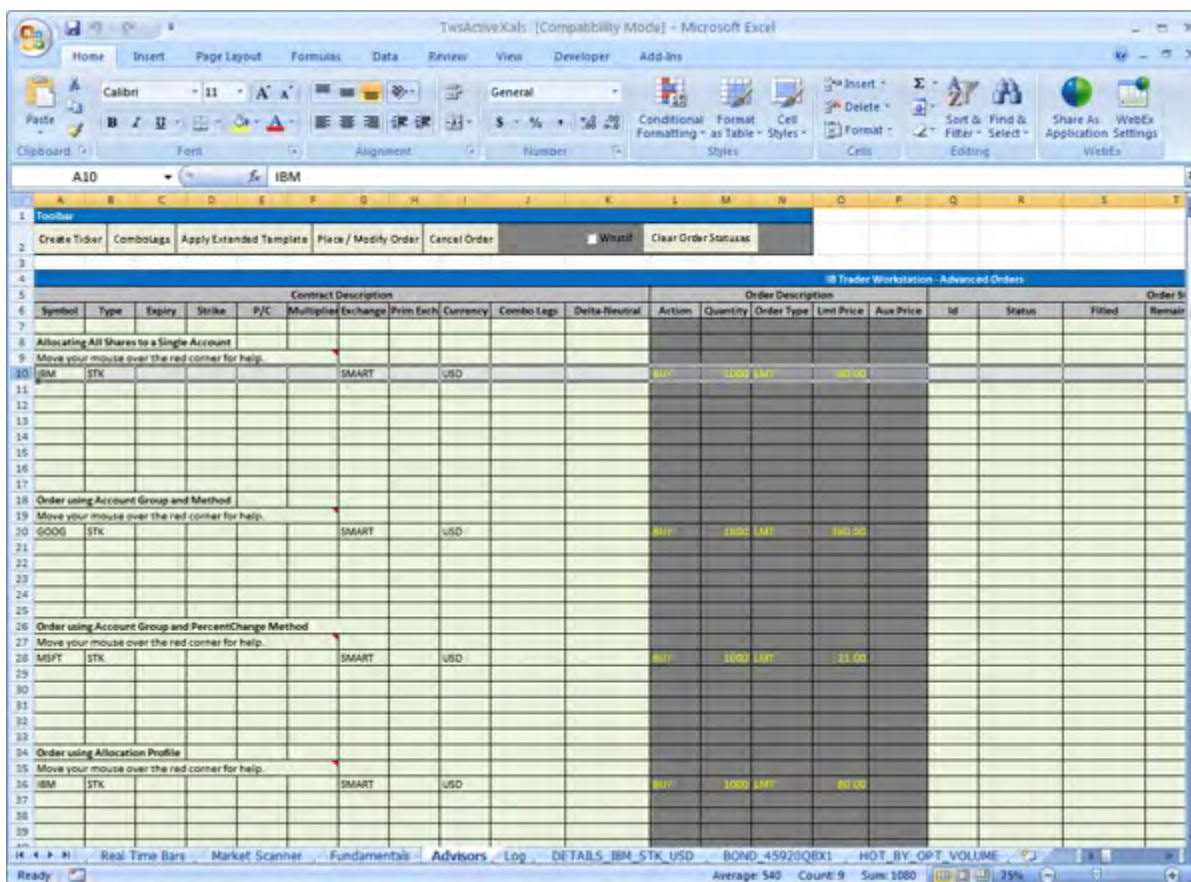
## Advisors Page

If you are a Financial Advisor and manage multiple accounts, use the Advisors page to create FA orders that:



- allocate shares to a single managed account
- use FA account groups and methods
- use allocation profiles

**Note:** You must set up your managed accounts, account groups, methods and allocation profiles in TWS before you can place FA orders in the ActiveX for Excel API sample spreadsheet.



### Allocating Shares to a Single Account

You can use the **Advisors** page to set up an order and allocate all shares in the order to a single account.

**Note:** Ensure that TWS is running, and that you have connected the spreadsheet to TWS.

#### To allocate shares to a single account:

1. Create an account group in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.

4. Click the **Extended Order Attributes** tab. Enter the account code in the *Value* cell for the *Account (Institutional only)* extended order attribute.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the *Account* order attribute value to the order. The *Account* value is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
7. Click the **Place/Modify Order** button.
8. When you are done allocating shares to the account, delete the *Account* value from the **Extended Order Attributes** page. If you do not delete this value, it will be applied to all subsequent orders placed from the ActiveX for Excel spreadsheet.

Optionally, you can receive margin and commission information that would result from the order if you placed it by selecting the **WhatIf** check box on the toolbar. In this case, your order is not actually placed. Deselect the check box to place your order without seeing the margin and commission information ahead of time.

#### Placing an Order using an FA Account Group and Method

You can also use the **Advisors** page to set up an order using an FA account group and FA method.

#### To place an order using an FA account group and FA method:

1. Create the FA account group(s) and FA method(s) in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.
4. Click the **Extended Order Attributes** tab. Enter values for the following extended order attributes:
  - FA Group - Enter the name of the account group.
  - FA Method - Enter the name of the allocation method to use for this order.
  - FA Percentage - Enter the percentage used by the PctChange allocation method to use for this order. This attribute applies only to FA groups that use this method.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the extended order attribute values to the order. The values for *FA Group*, *FA Method* and *FA Percentage* are applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
7. Click the **Place/Modify Order** button.
8. When you are done allocating shares to the account, delete the values you entered on the **Extended Order Attributes** page. If you do not delete these values, they will be applied to all subsequent orders placed from the ActiveX for Excel spreadsheet.

Optionally, you can receive margin and commission information that would result from the order if you placed it by selecting the **WhatIf** check box on the toolbar. In this case, your order is not actually placed. Deselect the check box to place your order without seeing the margin and commission information ahead of time.

### Placing an Order using an Allocation Profile

You can also use the **Advisors** page to set up an order using an FA allocation profile.

#### To place an order using an FA allocation profile:

1. Create the FA allocation profile in TWS.
2. Click the **Advisors** tab at the bottom of the spreadsheet.
3. Enter the contract information in the *Contract Description* cells, then enter the order information in the *Order Description* cells.
4. Click the **Extended Order Attributes** tab. Enter the name of the allocation profile in the *Value* field for the *FA Profile* extended order attribute.
5. Click the **Advisors** tab.
6. Highlight the order row, then click the **Apply Extended** button to apply the extended order attribute value to the order. The value for *FA Profile* is applied to the selected order and displayed in the *Extended Order Attributes* section of the page.
7. Click the **Place/Modify Order** button.
8. When you are done allocating shares to the account, delete the *FA Profile* value you entered on the **Extended Order Attributes** page. If you do not delete this value, it will be applied to all subsequent orders placed from the ActiveX for Excel spreadsheet.

Optionally, you can receive margin and commission information that would result from the order if you placed it by selecting the **WhatIf** check box on the toolbar. In this case, your order is not actually placed. Deselect the check box to place your order without seeing the margin and commission information ahead of time.

### Advisors Page Toolbar Buttons

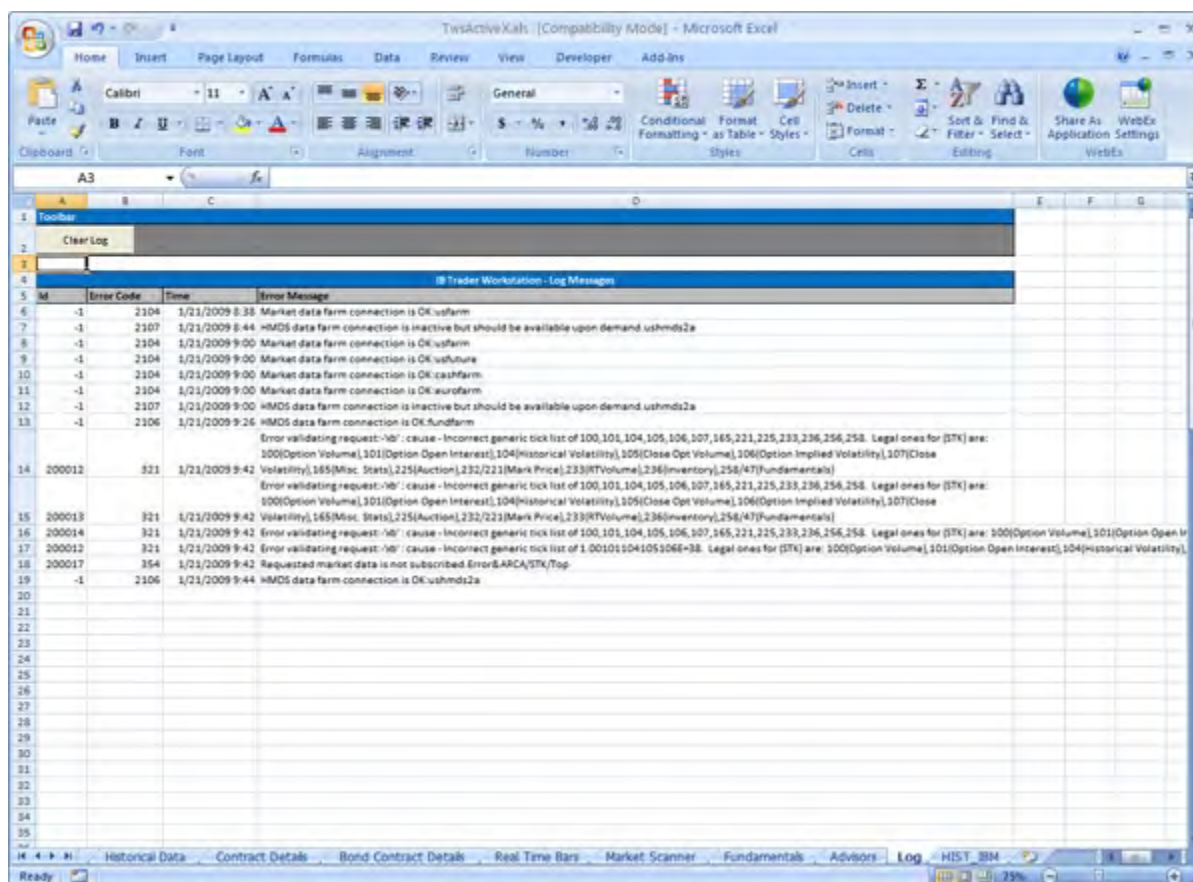
The toolbar on the Basic Orders page includes the following buttons:

Button	Description
Create Ticker	Opens the Ticker box. Enter information to create a market data line.
Combo Legs	Opens the Combination Legs box. Enter contract details to create legs of a combination order one by one.
Apply Extended Template	Applies the current values on the Extended Order Attributes page to the highlighted order row.
Place/Modify Orders	After you have completed the Order Description fields, and defined any extended attributes, click to create an order for the selected contract.
Cancel Order	This button cancels the order(s) you have highlighted.
Clear Order Statuses	Clears all information from the <i>Order Status</i> cells.

## Log Page

The Log page displays all error messages received while logged into TWS and using the ActiveX for Excel spreadsheet. You can clear all the information on the page by clicking **Clear Log** on the toolbar.

Here is an example of a typical Log page:



## POSIX

This chapter describes the POSIX API.

The POSIX API is based on our [C++](#) API code. The C++ code was refactored so it could be built on any POSIX-compliant platform. Use this new POSIX API to build a TWS API on Linux, and on Windows in non-MFC applications.

**Note:** Although the pre-existing public interface has been preserved, you must recompile your client applications.

We also include a POSIX test client. The API installation directory includes these directories for the POSIX API: PosixSocketClient and TestPosixSocketClient. The POSIX test client uses the same methods as the C++ Socket client, plus it exposes several extra methods that clients must call when data is available on a socket for read/write. Refer to TestPosixSocketClient as an example. Please note that this test client is greatly simplified. For real POSIX API applications, you will have to use a select system of some kind to manage several sockets and/or asynchronous events.

To run the POSIX test client on a Windows machine, see [Running the POSIX Client on a Windows Machine](#).

## Running the POSIX Client on a Windows Machine

### To run the POSIX client on a Windows machine

1. Run **vcvars32.bat** at the command prompt. In the example below, **vcvars32.bat** is located in *C:\Program Files\Microsoft Visual Studio 8\VC\bin\*.

```
C:\>cd c:\Program Files\Microsoft Visual Studio 8\VC\bin
C:\Program Files\Microsoft Visual Studio 8\VC\bin>vcvars32.bat
```

2. If you ran **vcvars32.bat** successfully, the command prompt should look like this:

```
C:\WINDOWS\system32\cmd.exe
C:\>cd c:\Program Files\Microsoft Visual Studio 8\VC\bin
C:\Program Files\Microsoft Visual Studio 8\VC\bin>vcvars32.bat
C:\Program Files\Microsoft Visual Studio 8\VC\bin>"C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\vcvars32.bat"
Setting environment for using Microsoft Visual Studio 2005 x86 tools.
C:\Program Files\Microsoft Visual Studio 8\VC\bin>_
```

3. Navigate to *C:\...\TestPosixSocketClient* in the same command prompt window. In the example below, *TestPosixSocketClient* is located in the *C:\IB\_API\_963*.

```
C:\IB_API_963\TestPosixSocketClient>
```

4. Type **nmake -f Makefile.win** at the command prompt.

```
C:\IB_API_963\TestPosixSocketClient> nmake -f Makefile.win
```

5. Now run the POSIX sample application by running **PosixSocketClientTest.exe** in the same *C:\...\TestPosixSocketClient* directory.

```
C:\IB_API_963\TestPosixSocketClient>PosixSocketClientTest.exe
```

# Reference

This chapter includes the following TWS API reference information:

- [API Message Codes](#)
- [Historical Data Limitations](#)
- [Tick Types](#)
- [Generic Tick Types](#)
- [Order Types and IBAlgos](#)
- [Extended Order Attributes](#)
- [Order Status for Partial Fills](#)
- [Available Market Scanners](#)
- [Supported Time Zones](#)
- [Smart Combo Routing](#)
- [API Logging](#)
- [Requests for Quotes \(RFQs\)](#)
- [Support for Mini Options](#)
- [Requesting Real-Time Index Premium Data](#)
- [Troubleshooting FAQs](#)



## 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 <i>Percentage</i> 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:



Code	Description
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:
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.

Code	Description
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	<p>Max number (3) of market depth requests has been reached.</p> <p><b>Note:</b> 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.</p>
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.
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.
336	<p>The time or time zone is invalid.</p> <p>The correct format is hh:mm:ss xxx  where xxx is an optionally specified time-zone. E.g.: 15:59:00 EST</p> <p>Note that there is a space between the time and the time zone.</p> <p>If no time zone is specified, local time is assumed.</p>
337	<p>The date, time, or time-zone entered is invalid. The correct format is  yyyyymmdd hh:mm:ss xxx  where yyyyymmdd and xxx are optional. E.g.: 20031126 15:59:00 EST</p> <p>Note that there is a space between the date and time, and between the  time and time-zone.</p> <p>If no date is specified, current date is assumed.</p> <p>If no time-zone is specified, local time-zone is assumed.</p>
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	<p>Invalid delta. Valid values are from 1 to 100.</p> <p>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.</p>
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.
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.
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.
40	Dividend schedule query failed.

Code	Description
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.
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:
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
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 don't start next login 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
1100	Connectivity between IB and TWS has been lost.
1101	Connectivity between IB and TWS has been restored-data lost.*
1102	Connectivity between IB and TWS has been restored-data 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
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.

## Historical Data Limitations

Historical data requests are subject to the following limitations:

- Historical data requests that use a bar size of 30 seconds or less can only go back six months.
- Each request is restricted to duration and bar size values that return no more than 2000 bars (2000 bars per request).
- Historical data requests can go back one full calendar year or more, depending on the number of concurrent real-time 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.

**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.

## Valid Duration and Bar Size Settings for 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

## 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
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()

Tick Value	Description	Event/Function/Method
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()
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()



Tick Value	Description	Event/Function/Method
46	SHORTABLE	tickString()
47	FUNDAMENTAL_RATIOS	tickString()
48	RT_VOLUME	tickGeneric()
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:

0 = Not halted  
1 = Halted.

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

0 = Not halted  
1 = General halt (trading halt is imposed for purely regulatory reasons) with/without volatility halt.  
2 = Volatility only halt (trading halt is imposed by the exchange to protect against extreme volatility).

3. Applies to bond contracts only.

## 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	<a href="#">RTVolume</a> - 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	<a href="#">Fundamental Ratios</a>	47
411	Realtime Historical Volatility	58
456	<a href="#">IBDividends</a>	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.



Underlying	Exch	Description	Shortable Key	Bid TIF	Bid Size Action	Ask Quantity
IBM	SMAR...	Stock (NYSE)	GREEN	116.90	1	116.
AGG	SMAR...	Stock (ARCA)	RED	98.98	2	99.
GOOG	SMAR...	Stock (NASDAQ...)	GREEN	439.99	1	440.
MS	SMAR...	Stock (NYSE)	RED	27.47	12	27.
GS	SMAR...	Stock (NYSE)	RED	129.71	5	129.
YHOO	SMAR...	Stock (NASDAQ...)	GREEN	18.98	47	18.
SPY	SMAR...	Stock (AMEX)	RED	118.91	97	118.
CSCO	SMAR...	Stock (NASDAQ...)	GREEN	22.92	271	22.

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).

### TAG Values for FUNDAMENTAL\_RATIOS tickType

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

TAG	Description
NPRICE	<b>Closing Price</b> 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	<b>High Price</b> This price is the highest price the stock traded at in the last 12 months. This could be an intra-day high.
NLOW	<b>Low Price</b> This price is the lowest price the stock traded at in the last 12 months. This could be an intra-day low.
PDATE	<b>Pricing date</b> The pricing date is the date at which the issue was last priced.
VOL10DAVG	<b>Volume</b> This is the daily average of the cumulative trading volume for the last ten days.
MKTCAP	<b>Market capitalization</b> This value is calculated by multiplying the current Price by the current number of shares outstanding.
TTMEPSXCLX	<b>EPS excluding extraordinary items</b> 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	<b>EPS Normalized</b> 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.

TAG	Description
TTMREVPS	<p><b>Revenue/share</b></p> <p>This value is the trailing twelve month Total Revenue divided by the Average Diluted Shares Outstanding for the trailing twelve months.</p> <p><b>Note:</b> 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).</p>
QBVPS	<p><b>Book value (Common Equity) per share</b></p> <p>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.</p>
QTANBVPS	<p><b>Book value (tangible) per share</b></p> <p>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.</p>
QCSHPS	<p><b>Cash per share</b></p> <p>This is the Total Cash plus Short Term Investments divided by the Shares Outstanding at the end of the most recent interim period.</p> <p><b>Note:</b> This does NOT include cash equivalents that may be reported under long term assets.</p>
TTMCFSHR	<p><b>Cash Flow per share</b></p> <p>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.</p>
TTMDIVSHR	<p><b>Dividends per share</b></p> <p>This is the sum of the Cash Dividends per share paid to common stockholders during the last trailing twelve month period.</p>
IAD	<p><b>Dividend rate</b></p> <p>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.</p>
PEEXCLXOR	<p><b>P/E excluding extraordinary items</b></p> <p>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.</p>

TAG	Description
APENORM	<b>P/E Normalized</b> This is the Current Price divided by the latest annual Normalized Earnings Per Share value.
TMPR2REV	<b>Price to sales</b> 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	<b>Price to Tangible Book</b> 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.
TTMPCFPS	<b>Price to Cash Flow per share</b> 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	<b>Price to Book</b> This is the Current Price divided by the latest interim period Book Value Per Share.
QCURRATIO	<b>Current ratio</b> 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.
QQUICKRATI	<b>Quick ratio</b> 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.

TAG	Description
QLTD2EQ	<b>LT debt/equity</b> This ratio is the Total Long Term Debt for the most recent interim period divided by Total Shareholder Equity for the same period.
QTOTD2EQ	<b>Total debt/total equity</b> 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	<b>Payout ratio</b> 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	<b>Revenue</b> 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	<b>EBITD</b> 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	<b>Earnings before taxes</b> 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	<b>Net Income available to common</b> 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	<b>Earnings before taxes Normalized</b> This is the Income Before Tax number excluding the impact of all unusual/one-time/special charges items for the most recent annual period.

TAG	Description
ANIACNORM	<b>Net Income Available to Common, Normalized</b> 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.
TTMGROSMGN	<b>Gross Margin</b> 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	<b>Net Profit Margin %</b> 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	<b>Operating margin</b> 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	<b>Pretax margin</b> 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	<b>Return on average assets</b> 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	<b>Return on average equity</b> 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.
TTMROIPT	<b>Return on investment</b> 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.



TAG	Description
REVCHNGYR	<b>Revenue Change %</b> 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	<b>Revenue Change %</b> 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	<b>Revenue growth rate</b> The Five Year Revenue Growth Rate is the annual compounded growth rate of Revenues over the last 5 years.
EPSCHNGYR	<b>EPS Change %</b> 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 is Not Meaningful (NM).
TTMEPSCHG	<b>EPS Change %</b> 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	<b>EPS growth rate</b> 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	<b>Growth rate % - dividend</b> 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 or false).

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

## Order Types and IBAlgos

This section includes the following topics:

- [Supported Order Types](#)
- [IBAlgo Parameters](#)
- [CSFB Algo Parameters](#)

### Supported Order Types

IB's API technologies support the order types listed below.

API orders only mimic the behavior of Trader Workstation (TWS). Test each order type, ensuring that you can successfully submit each one in TWS, before you submit the same order using the API.

Order Type	Abbreviation
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

Order Type	Abbreviation
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
Immediate-or-Cancel	IOC
Advanced Trading	
One-Cancels-All	OCA
Spreads	
Volatility	VOL
Algorithmic Trading (Algos)	
Arrival Price	
Balance Impact and Risk	
Minimize Impact	

Order Type	Abbreviation
Percent of volume	
Scale	
TWAP	
VWAP - Best Effort	
Accumulate/Distribute	
IBDARK	

## IBAlgo Parameters

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	allowPastEndTime	noTakeLiq	riskAversion	forceCompletion
<b>Algo Strategy</b>									
<b>For US Stocks</b>									
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		
<b>For US Options</b>									
Balance Impact and Risk	X							X	X
Minimize Impact	X								

**Arrival Price (ArrivalPx)**

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”

**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);
```

**For More Information...**

- [Arrival Price Algo](#)

**Dark Ice (DarkIce)**

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"

**For More Information...**

- [Dark Ice Algo](#)

**Percentage of Volume (PctVol)**

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"

**For More Information...**

- [Percentage of Volume Algo](#)



**TWAP (Twap)**

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”

**For More Information...**

- [TWAP Algo](#)

**VWAP (Vwap)**

Parameter	Description	Syntax
maxPctVol	Maximum percentage	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”
getDone	Get Done	“0” or “1”
noTradeAhead	No trade ahead	“0” or “1”
useOddLots	Use odd lots	“0” or “1”

**For More Information...**

- [VWAP Algo](#)

**Balance Impact and Risk (BalanceImpactRisk)**

Parameters	Description	Syntax
maxPctVol	Maximum percentage	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);

```

#### For More Information...

- [Balance Impact and Risk Algo](#)

#### Minimize Impact (MinImpact)

Parameter	Description	Syntax
maxPctVol	Maximum percentage	range: “0.01” – “0.5”

#### For More Information...

- [Minimize Impact](#)

### Accumulate/Distribute (AD)

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);
```

#### For More Information...

- [Accumulate Distribute Order Type](#)
- [TWS Accumulate Distribute](#)

#### 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.

	<b>CSFB Algo Corresponding Parameters</b>						
	Abbreviation	StartTime	EndTime	MinPctVolume	MaxPctVolume	DisplaySize	ExecutionStyle
<b>Algo Strategy</b>							
CrossFinder	CROS	X	X		X		X
Float	FLT	X				X	X
Guerilla	GRRL	X			X		X
Work It IW	INIW	X					
Work It	INLN	X	X	X	X		X
Path Finder	PTHF	X	X				
Reserve	RSRV	X				X	
Strike	SNPR	X				X	
10B 18	TENB	X	X		X		
Tex	TEX	X	X		X		X
TWAP	TWAP	X	X		X		
VWAP	VWAP	X	X		X		

### Crossfinder (CROS)

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
endTime	End time:	"15:00:00 EST"
MaxPctVolume	Maximum percentage volume	range: "0" – "99"
ExecutionStyle	Execution style	"Normal", "Patient", "Aggressive"

### Crossfinder (CROS) Java Code Sample

```
void onCrossFinderAlgo() {

    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "13:30:00 EST"));
    m_algoParams.add(new TagValue("EndTime", "14:30:00 EST"));
    m_algoParams.add(new TagValue("MaxPctVolume", "25")); // Max % Volume
    m_algoParams.add(new TagValue("ExecutionStyle", "Normal"));
    //possible values for ExecutionStyle: Normal, Patient, Aggressive
}
```

```

Order order = new Order();
order.m_action = "BUY";
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "CROS";
order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);

}

```

#### Float (FLT)

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
DisplaySize	Display size	"50" (integer)
ExecutionStyle	Execution style	"Normal", "Patient", "Aggressive"

#### Float (FLT) Java Code Sample

```

void onFloatCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "13:30:00 EST"));
    m_algoParams.add(new TagValue("DisplaySize", "10")); //iceberg
    m_algoParams.add(new TagValue("ExecutionStyle", "Normal"));
    //possible values for ExecutionStyle: Normal, Patient, Aggressive
}

```

```

Order order = new Order();
order.m_action = "BUY";
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "FLT";
order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);

}

```

### Guerilla (GRRL)

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
MaxPctVolume	Maximum percentage volume	range: "0" – "99"
ExecutionStyle	Execution style	"Normal", "Patient", "Aggressive"

### Guerilla (GRRL) Java Code Sample

```

void onGuerillaCsfbAlgo() {

    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "13:30:00 EST"));
    m_algoParams.add(new TagValue("MaxPctVolume", "25")); // Max % Volume
    m_algoParams.add(new TagValue("ExecutionStyle", "Patient"));
    //possible values for ExecutionStyle: Normal, Patient, Aggressive
}

```

```

Order order = new Order();
order.m_action = "BUY";
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "GRRL";
order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);
}

```

#### Work It IW (INIW)

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"

#### Work It IW (INIW) Java Code Sample

```

void onWorkItIwCsfbAlgo() {

Contract con = new Contract();
con.m_symbol = "IBM";
con.m_secType = "STK";
con.m_exchange = "CSFBALGO";
con.m_currency = "USD";

Vector<TagValue> m_algoParams = new Vector<TagValue>();
m_algoParams.add(new TagValue("StartTime", "13:30:00 EST"));

Order order = new Order();
order.m_action = "BUY";
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "INIW";

```



```

order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);

}

```

**Work It (INLN)**

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
endTime	End time:	"15:00:00 EST"
MinPctVolume	Minimum percentage volume	range: "0" – "99"
MaxPctVolume	Maximum percentage volume	range: "0" – "99"
ExecutionStyle	Execution style	"Normal", "Patient", "Aggressive"

**Work It (INLN) Java Code Sample**

```

void onWorkItCsfbAlgo() {

Contract con = new Contract();
con.m_symbol = "IBM";
con.m_secType = "STK";
con.m_exchange = "CSFBALGO";
con.m_currency = "USD";

Vector<TagValue> m_algoParams = new Vector<TagValue>();
m_algoParams.add(new TagValue("StartTime", "13:45:00 EST"));
m_algoParams.add(new TagValue("EndTime", "15:30:00 EST"));
m_algoParams.add(new TagValue("MinPctVolume", "15")); // Min % Volume
m_algoParams.add(new TagValue("MaxPctVolume", "25")); // Max % Volume
m_algoParams.add(new TagValue("ExecutionStyle", "Patient"));
//possible values for ExecutionStyle: Normal, Patient, Aggressive

Order order = new Order();
order.m_action = "BUY";

```

```
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "INLN";
order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);
}
```

**Pathfinder (PTHF)**

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
endTime	End time:	"15:00:00 EST"

**Pathfinder (PTHF) Java Code Sample**

```
void onPathFinderCsfbAlgo() {

    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "13:45:00 EST"));
    m_algoParams.add(new TagValue("EndTime", "15:30:00 EST"));

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "PTHF";
```

```

order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);
}

```

### Reserve (RSRV)

Parameter	Description	Syntax
startTime	Start time	"9:00:00 EST"
DisplaySize	Display size	"50" (integer)

### Reserve (RSRV) Java Code Sample

```

void onReserveCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
    m_algoParams.add(new TagValue("DisplaySize", "50")); //iceberg

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "RSRV";
    order.m_algoParams = m_algoParams;
    order.m_transmit = false;
    m_client.placeOrder(globalOrderId++, con, order);
}

```

**Strike (SNPR)**

Parameter	Description	Syntax
startTime	Start time	“9:00:00 EST”
DisplaySize	Display size	“50” (integer)

**Strike (SNPR) Java Code Sample**

```

void onStrikeCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
    m_algoParams.add(new TagValue("DisplaySize", "50")); //iceberg

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "SNPR";
    order.m_algoParams = m_algoParams;
    order.m_transmit = false;
    m_client.placeOrder(globalOrderId++, con, order);
}

```

**10B 18 (TENB) Java Code Sample**

Parameter	Description	Syntax
startTime	Start time	“9:00:00 EST”
endTime	End time:	“15:00:00 EST”
MaxPctVolume	Maximum percentage volume	range: “0” – “99”

**10B 18 (TENB) Java Code Sample**

```

void on10BCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
    m_algoParams.add(new TagValue("EndTime", "15:40:00 EST"));
    m_algoParams.add(new TagValue("MaxPctVolume", "35")); // Max % Volume

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "TENB";
    order.m_algoParams = m_algoParams;
    order.m_transmit = false;
    m_client.placeOrder(globalOrderId++, con, order);
}

```

**Tex (TEX)**

Parameter	Description	Syntax
startTime	Start time	“9:00:00 EST”
endTime	End time:	“15:00:00 EST”
MaxPctVolume	Maximum percentage volume	range: “0” – “99”
ExecutionStyle	Execution style	"Normal", "Patient", "Aggressive"

**Tex (TEX) Java Code Sample**

```

void onTexCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
    m_algoParams.add(new TagValue("EndTime", "15:40:00 EST"));
    m_algoParams.add(new TagValue("MaxPctVolume", "47")); // Max % Volume
    m_algoParams.add(new TagValue("ExecutionStyle", "Normal"));
    //possible values for ExecutionStyle: Normal, Patient, Aggressive

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "TEX";
    order.m_algoParams = m_algoParams;
    order.m_transmit = false;
    m_client.placeOrder(globalOrderId++, con, order);
}

```

**TWAP (TWAP)**

Parameter	Description	Syntax
startTime	Start time	“9:00:00 EST”
endTime	End time:	“15:00:00 EST”
MaxPctVolume	Maximum percentage volume	range: “0” – “99”

**TWAP (TWAP) Java Code Sample**

```

void onTwapCsfbAlgo() {
    Contract con = new Contract();
    con.m_symbol = "IBM";
    con.m_secType = "STK";
    con.m_exchange = "CSFBALGO";
    con.m_currency = "USD";

    Vector<TagValue> m_algoParams = new Vector<TagValue>();
    m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
    m_algoParams.add(new TagValue("EndTime", "15:40:00 EST"));
    m_algoParams.add(new TagValue("MaxPctVolume", "48")); // Max % Volume

    Order order = new Order();
    order.m_action = "BUY";
    order.m_totalQuantity = 100;
    order.m_orderType = "LMT";
    order.m_lmtPrice = 200.0;
    order.m_algoStrategy = "TWAP";
    order.m_algoParams = m_algoParams;
    order.m_transmit = false;
    m_client.placeOrder(globalOrderId++, con, order);
}

```

**VWAP (VWAP)**

Parameter	Description	Syntax
startTime	Start time	“9:00:00 EST”
endTime	End time:	“15:00:00 EST”
MaxPctVolume	Maximum percentage volume	range: “0” – “99”

**VWAP (VWAP) Java Code Sample**

```

void onVwapCsfbAlgo() {

```

```
Contract con = new Contract();
con.m_symbol = "IBM";
con.m_secType = "STK";
con.m_exchange = "CSFBALGO";
con.m_currency = "USD";

Vector<TagValue> m_algoParams = new Vector<TagValue>();
m_algoParams.add(new TagValue("StartTime", "14:40:00 EST"));
m_algoParams.add(new TagValue("EndTime", "15:40:00 EST"));
m_algoParams.add(new TagValue("MaxPctVolume", "49")); // Max % Volume

Order order = new Order();
order.m_action = "BUY";
order.m_totalQuantity = 100;
order.m_orderType = "LMT";
order.m_lmtPrice = 200.0;
order.m_algoStrategy = "VWAP";
order.m_algoParams = m_algoParams;
order.m_transmit = false;
m_client.placeOrder(globalOrderId++, con, order);
}
```



## Extended Order Attributes

The extended order attributes below can be used in all placeOrder functions and Open\_Order events.

Attribute	Possible Values
string m_tif	Day, GTC, IOC, GTD
string m_ocaGroup	Identifies a member of a one-cancels-all group.
string m_account	Institutional only.
string m_openClose	Institutional only.
int m_origin	Institutional only.
string m_orderRef	Customer defined order ID tag.
boolean m_transmit	Specifies whether the order will be transmitted by TWS. If set to false, order is created by not transmitted.
int m_parentId	The order ID of the parent, used for bracket, auto stop and trailing stop orders.
boolean m_blockOrder	If set to true, specifies that the order is a block order.
boolean m_sweep-ToFill	If set to true, specifies that the order is a Sweep-to-fill order.
int m_displaySize	The publicly disclosed order size to be used when placing iceberg orders.
int m_triggerMethod	Specifies how Simulated Stop, Stop-Limit and Trailing Stop orders are triggered. Valid values are: <ul style="list-style-type: none"> <li>• O - 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.</li> <li>• 1 - use "double bid/ask" method, where stop orders are triggered based on two consecutive bid or ask prices.</li> <li>• 2 - "last" method, where stop orders are triggered based on the last price.</li> <li>• 3 - "double-last" method, where stop orders are triggered based on last two prices.</li> </ul>
boolean m_ignoreRth	If set to true, allows triggering of orders outside of regular trading hours.
boolean m_hidden	If set to true, the order will not be visible when viewing the market depth. The only applies to orders routed to INet.

Attribute	Possible Values
string m_good-AfterTime	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.
string m_goodTillDate	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.
string m_faGroup	The advisor group to which the trade will be allocated. Use an empty string if not applicable.
string m_faProfile	The advisor allocation profile to which the trade will be allocated. Use an empty string if not applicable.
string m_faMethod	The advisor allocation method with which the trade will be allocated. Use an empty string if not applicable.
string m_faPercentage	The advisor percentage concerning the trade's allocation. Use an empty string if not applicable.
string m_primaryExch	To clarify any ambiguity for Smart-routed contracts, include the primary exchange, along with the Smart designation, for the destination.
int m_shortSaleSlot	For institutional customers only. <ul style="list-style-type: none"> <li>• 0 - unapplicable (i.e. retail customer or not sshort leg)</li> <li>• 1 - clearing broker</li> <li>• 2 - third party. If this value is used, you must enter a designated location.</li> </ul>
string m_designatedLocation	Only valid when shortSaleSlot value = 2. Otherwise leave blank or orders will be rejected.
long ocaType	Cancel on Fill with Block = 1 Reduce on Fill with Block = 2 Reduce on Fill without Block = 3
int rthOnly	Regular trading hours only. yes=1, no=0

Attribute	Possible Values
String rule80A	Individual = 'I' Agency = 'A', AgentOtherMember = 'W' IndividualPTIA = 'J' AgencyPTIA = 'U' AgentOtherMemberPTIA = 'M' IndividualPT = 'K' AgencyPT = 'Y' AgentOtherMemberPT = 'N'
String settlingFirm	Institutional only
String clearingAccount	The true beneficiary of the order. This value must be sent on FUT/FOP orders for reporting the exchange.
String clearingIntent	IB, Away, or PTA
int allOrNone	yes=1, no=0
long minQty	Identifies a minimum quantity order type.
double percentOffset	The percent offset for relative orders.
int eTradeOnly	Trade with electronic quotes. yes=1, no=0
int firmQuoteOnly	Trade with firm quotes. yes=1, no=0
double nbboPriceCap	Maximum SMART order distance from the NBBO.
long auctionStrategy	match = 1 improvement = 2 transparent = 3 For BOX exchange only.
double startingPrice	Starting price. For BOX exchange only.
double stockRefPrice	The stock reference price. For BOX exchange only.
double delta	For BOX exchange only.
double stock-RangeLower	The lower value of the acceptable stock range. For BOX exchange only.
double stock-RangeUpper	The upper value of the acceptable stock range. For BOX exchange only.

Attribute	Possible Values
double m_volatility	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.
int m_volatilityType	1 = Daily; 2 = Annual
m_continuousUpdate	0 = false; 1 = True
int m_referencePriceType	1 = Average; 2 = BidorAsk
String m_deltaNeutralOrderType	Enter an accepted order type such as: MKT, LMT, REL etc.
double m_deltaNeutralAuxPrice	Enter the Aux Price for Hedge Delta order types that require one.
int m_scale-NumComponents	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.
int m_scale-ComponentSize	For Scale orders: Defines the number of units per component, backing into the number of components into which the parent order is split.
double m_scalePriceIncrement	For Scale orders: Defines the price increment per scale component.
double m_basisPoints	EFP orders
int basisPointsType	EFP orders

## 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:

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

## Available Market Scanners

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

Market Scanner (Scan Code)	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 Historical (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 current 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.

Market Scanner (Scan Code)	Description
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)	<p>Contracts with the highest trading volume today, based on units used by TWS (lots for US stocks; contract for derivatives and non-US stocks).</p> <p>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.</p>
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)	<p>Contracts where:</p> <ul style="list-style-type: none"> <li>today's Volume/avgDailyVolume is highest.</li> <li>avgDailyVolume is a 30-day exponential moving average of the contract's daily volume.</li> </ul>

Market Scanner (Scan Code)	Description
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: <ul style="list-style-type: none"> <li>• <math>(\text{lastTradePrice} - \text{prevClose}) / \text{avgDailyChange}</math> is highest in absolute value (positive or negative).</li> <li>• The avgDailyChange is defined as an exponential moving average of the contract's <math>(\text{dailyClose} - \text{dailyOpen})</math></li> </ul>
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.



Market Scanner (Scan Code)	Description
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.

Market Scanner (Scan Code)	Description
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

## 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

## 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 **smart-ComboRoutingParams**, 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.

## 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:

**[Client-  
tID: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.

## 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)

## Support for Mini Options

The APIs support mini-options in requests to TWS and data returned by TWS. Click one of the following links for more information.

- [Support for Mini Options - Active X, Java and C++ APIs](#)
- [Support for Mini Options - DDE for Excel API](#)

### 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*.

## Support for Mini Options - DDE for Excel

You can identify mini options in the DDE for Excel API by providing the multiplier or trading class in both requests to TWS and responses from TWS. The DDE for Excel API spreadsheet (*TwsDde.xls*) released with API Version 9.69 was updated to include the Trading Class column on most pages. You use this field to request mini options data from TWS.

### Requirements

Mini-option support in the DDE for Excel API requires the following:

- The updated Excel spreadsheet requires *ddedll.dll* Version 16 and TWS Version 9.69 or higher.
- Previous versions of the Excel spreadsheet will not work with *ddedll.dll* Version 16 and TWS Version 9.69 or higher.
- However, TWS Version 9.69 or higher will work with previous versions of the Excel spreadsheet if you are using a previous version of the *ddedll.dll* file.

## DDE Syntax Examples

DDE syntax has been updated to support mini options data. The following examples show which requests for contract data have been updated for mini options support. To check the DDE syntax in the Excel spreadsheet, look at the Ctrl cells on the spreadsheet page for each data request.

### Requests That Send Contract Data to TWS

- Request market data:

topic=tik request=req (Request Market Data - Tickers page)

```
=Salexd406|tik!'id0?req?AAPL_OPT_20130614_530_C_SMART_USD_~_AAPL7/'
```

```
=Salexd406|tik!'id1?req?AAPL_BAG_SMART_USD_CMBLGS_2_126721266_1_BUY_SMART_0_1236033_1_SELL_SMART_0_CMBLGS_~/'
```

- Calculate implied volatility:

topic=calcimplvol request=get (Calculate Implied Volatility - Tickers page)

```
=Salexd406|calcimplvol!'id4?get?5_690_AAPL_OPT_20130614_530_C_SMART_USD_AAPL7/'
```

- Calculate options price:

topic=calcoptionprice request=get (Calculate Option Price - Tickers page)

```
=Salexd406|calcoptionprice!'id5?get?0.23_690_AAPL_OPT_20130614_530_C_SMART_USD_AAPL7/'
```

- Request generic ticks:

topic=gentick request=get (Request Generic Ticks - Tickers page)

```
=Salexd406|gentick!'id6?req?318?AAPL_OPT_20130614_530_C_SMART_USD_~_AAPL7/'
```

```
=Salexd406|gentick!'id5?req?318?AAPL_BAG_SMART_USD_CMBLGS_2_126721266_1_BUY_SMART_0_1236033_1_SELL_SMART_0_CMBLGS_~/'
```

- Place order:

topic=ord request=place (Place / Modify Order - Basic Orders, Conditional Orders, Advanced Orders, Advisors pages)

```
=Salexd406|ord!'id424433486?place?AAPL_OPT_20130614_530_C_SMART_USD~_AAPL7/BUY_1_LMT_0.05~_DAY~_~_O_0~_1~_0_0_0_0~_0_0~...'

```

```
=Salexd406|ord!'id424433487?place?AAPL_BAG_SMART_USD_CMBLGS_2_126721266_1_BUY_SMART_0_1236033_1_SELL_SMART_0_CMBLGS~/BUY_1_LMT_0.05~_DAY~_~_O_0~_1~_0_0_0_0~_0_0~...'

```

- Request historical data:

topic=hist request=req (Request Historical Data - Historical Data page)

```
=Salexd406|hist!'id4?req?AAPL_OPT_20130517_490_C_SMART_USD~_AAPL7/2-0130512singleSpace10singleColon00singleColon00singleSpaceGMT_1singleSpaceW_11_MIDPOINT_0_1_TRUE'

```

```
=Salexd406|hist!'id5?req?AAPL_OPT_20130517_490_C_SMART_USD~_AAPL7/2-0130512singleSpace10singleColon00singleColon00singleSpaceGMT_1singleSpaceW_11_MIDPOINT_0_1'

```

```
=Salexd406|hist!'id6?req?AAPL_BAG_SMART_USD_CMBLGS_2_126721266_1_BUY_SMART_0_1236033_1_SELL_SMART_0_CMBLGS~/20130512singleSpace10singleColon00singleColon00singleSpaceGMT_1singleSpaceW_11_MIDPOINT_0_1'

```

- Request contract details:

topic=contract request=req (Re-request Contract Details - Contract Details page)

```
=Salexd406|contract!'id1?req?AAPL_OPT_20130517_490_C_SMART_USD~_AAPL7/'

```

- Request market depth:

topic=mktDepth request=req (Request Market Depth - Market Depth page)

```
=Salexd406|mktDepth!'id0?req?AAPL_OPT_20130517_490_C_SMART_USD_AAPL7/?0'=

```

#### Requests That Receive Contract Data from TWS

- topic=opens request=req (open orders - Open Orders page)

Trading Class is expected after Multiplier and before Exchange

- topic=execs request=req (executions - Executions page)

Trading Class is expected after P/C (Right) and before Exchange

- topic=ports request=req (portfolio updates - Portfolio page)

Trading Class is expected after Right and before Currency

- topic=scan request=req (market scanner data)

Trading Class is expected after Right and before Exchange

#### For more information

- [DDE Syntax for Excel](#)

## 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).

## Troubleshooting FAQs

Here are some common API issues/questions and their solutions/answers.

### **I received the error "The DDEDLL.dll file required for Excel integration is either missing or out of date." - (DDE for Excel API)**

This error appears when you try to launch Trader Workstation (TWS) on a 64-bit Windows computer using a 32-bit Java executable. To solve this issue, launch TWS using the 64-bit Java executable.

#### **To launch the standalone TWS using the 64-bit Java executable**

1. On your Windows desktop, right-click the TWS shortcut and select *Properties* from the popup menu.
2. In the Properties dialog, click the **Shortcut** tab, then click at the very beginning of the Target field.
3. Typically the text in the Target field displays a long path beginning with "*C:\Windows\system32\javaw.exe ...*". Change the part of the path that says *system32* to *sysWOW64*. The *sysWOW64* folder contains the 64-bit Java executable (javaw.exe). Click **OK** when you're done.
4. If the Target path already contains "*sysWOW64*," change to *system32*, then click **Apply**. Then change *system32* back to *sysWOW64* and click **Apply**, then **OK**.
5. Launch TWS.

The browser-based TWS can be launched using 64-bit Java executable with a 64-bit browser like Internet Explorer (Firefox and Google Chrome do not have 64-bit version when this version of the API Reference Guide was published). In some cases, when both Java executables are installed (32-bit and 64-bit), the 64-bit browser may still use the 32-bit Java. In this case it is necessary to uninstall the Java 32-bit executable and possibly even re-install the Java 64-bit executable.

### **Can I get historical data without a market data subscription?**

No. However, some market data subscriptions are free and enabled by default and you can retrieve historical data for the market data represented by these free subscriptions.

### **Can I retrieve through API orders created in TWS?**

Yes.

### **Can I modify orders in the API that were created in TWS?**

Yes in the Active X, Java and C++ APIs (*not* in the DDE for Excel API). To modify the orders, you must use the orderID, not the permID. The orderID will be negative if the order is created in TWS.

### **Why do I get I get all zeroes in a market data cell when using the DDE for Excel API?**

This happens when the Excel DDE sheet has not been connected properly to TWS. To solve, restart both Excel and TWS.

### **Are conditional orders possible in the API?**

Conditional orders sent from API, in the sense that they are held and monitored on the servers of Interactive Brokers, are not available. Traders can monitor the condition on their machines and send the orders for execution when the condition is satisfied.

TWS *does* offer conditional orders that are held and monitored on IB's servers, albeit the range of conditions is limited to a few variables like price and volume.

### **How many API clients can connect to Trader Workstation/IB Gateway simultaneously?**

A single instance of Trader Workstation (or IB Gateway) can support a maximum of 8 API clients at the same time.



# Index

## A

about the APIs 24

account details in ActiveX for Excel 346

account information in Excel 68

account information, viewing in ActiveX for Excel 347

Account page

    in ActiveX for Excel 346

    using in Excel 68

Account page in Excel 67

    account values 69

Account page toolbar 69

Account page, using in ActiveX for Excel 347

account values in Excel 69

accountSummary() 143, 208, 276

accountSummaryEnd() 145, 210, 279

Active X 105

Active X events 139

ActiveX

    linking to TWS 106

    placing a combination order 168

    registering third-party controls 107

ActiveX API

    on 64-bit systems 108

ActiveX COM objects 149-151, 153-155, 163-165

ActiveX events 127-132, 134, 137-139, 141-143, 146-148

ActiveX factory methods 124-125

ActiveX for Excel

    Account page 346

    Advanced Orders page 336

    Advisors page 366

    allocating shares to a single account 367

    Basic Orders page 328

    Bond Contract Details page 359

    bracket orders in 338

    Bulletins page 326

    connecting to TWS 321

    Contract Details page 357

    disconnecting from TWS 322

    download API components 318

    Executions page 350

    Extended Order Attributes page 342

    FA orders using account group and method 368

    FA orders using allocation profile 369

    Fundamentals page 365

    General page 321

    getting started 318

    Historical Data page 352

    Log page 370

- Market Depth page 327
- Market Scanner page 362
- Open Orders page 344
- opening sample spreadsheet 319
- placing orders in 329
- Real Time Bars page 361
- relative orders in 341
- requesting current time 322
- scale orders in 341
- setting server log level in 322
- Tickers page 323
- trailing stop limit orders in 340
- ActiveX for Excel historical data
  - expired contracts 353
- ActiveX for Excel on 64-bit Windows 318
- ActiveX for Excel sample spreadsheet
  - using 320
- ActiveX methods 110-112, 115, 117, 122-124
- ActiveX properties 167
- ActiveX sample program 109
- Advanced Orders
  - in ActiveX for Excel 336
- Advanced Orders page
  - in Excel 58
- Advanced Orders page toolbar 63
- advisors 307
  - financial reporting for 311
- Advisors
  - change or update allocation information 315
  - Java code samples for 314
  - place order for a single managed account 314
  - place order for an account group 315
  - place order for an allocation profile 314
- Advisors page 92-93
  - in ActiveX for Excel 366
- Advisors page in Excel 91
- Advisors page toolbar 94
- advisors, Excel DDE support for 92-93, 309
- algos
  - CSFB 410
- allocating shares to a single account in Excel 92
- allocation methods for account groups 312
- allocation profiles in Excel 93
- API
  - about 24
  - recommendations 28
- API components, downloading 318
- API components, using 24
- API logging 435
- API message codes 374
- API overview 23
- API request/server response message identifiers 436
- API settings in TWS 32, 38



API software

    downloading 37

    uninstalling 34

API, for financial advisor accounts 307

apply extended order attributes 49

Apply Extended Template button 343

Arrival Price Java code sample 405

available market scanners 428

available market scanners in Excel 82

AvailableEquity Method 312

## **B**

bar size settings for historical data 388

Basic Orders page

    combination orders 330

    in ActiveX for Excel 328

    in Excel 44

    modifying orders 330

    placing orders in ActiveX for Excel 329

    toolbar buttons 332

Basic Orders page toolbar in Excel 47

basket orders in Excel 45

basket orders, in ActiveX for Excel 329

bond contract details 88

Bond Contract Details page

    in ActiveX for Excel 359

Bond Contract Details page in Excel 88

Bond Contract Details page toolbar 89

bond contract details, requesting in ActiveX for Excel 359

bondContractDetails() 139, 205, 273

bracket orders

    in ActiveX for Excel 338

    in Excel 59

Bulletins page

    in ActiveX for Excel 326

    toolbar buttons 327

## **C**

C++ 171, 194

    Class EClientSocket methods 179

    Class EWrapper functions 195

    combinations orders 230

    linking to TWS 172

    prebuilt sample application 178

    running the TestSocketClient from Visual Studio 2008 178

    using the sample program 178

C++ SocketClient properties 214

calcOptionPriceAndGreeks 113

calculateImpliedVolatility 112, 182, 250

calculateOptionPrice 182, 250

calendar spread in Excel 46

calendar spread order in Excel 330

calendar spread order, in C++ 230

cancelAccountSummary() 120, 190, 258

cancelCalculateImpliedVolatility 112, 182, 250

cancelCalculateOptionPrice 113, 183, 250

cancelFundamentalData 126, 194

cancelFundamentalData() 263

cancelHistoricalData() 122, 192, 261

cancelMktData() 112, 182, 249

cancelMktDepth() 116, 186, 254

cancelNewsBulletins() 117, 187, 255

cancelOrder() 114, 183, 251

cancelPositions() 120, 190, 258

cancelRealTimeBars() 124, 193, 262

cancelScannerSubscription() 123, 192, 259

checkMessages() 181

Class EClientSocket methods 179-187, 190, 192-193

Class EWrapper functions 195-200, 202-207, 211-213

code for DDE for Excel API 95

code modules in Excel 96

com.ib.client Java package 245

combination order, in ActiveX 168

combination order, in ActiveX for Excel 330

combination orders in Excel 46

combination orders, in C++ 230

combination orders, in Java 300

ComboLeg 219, 288

commissionReport 206

CommissionReport 229, 285

commissionReport() 141, 274

common issues and solutions 442

conditional Orders page 55

Conditional Orders page 333

in Excel 54

conditional orders in Excel, examples of 56, 335

Conditional Orders page

toolbar buttons 336

Conditional Orders page toolbar 58

Conditional Orders page, modifying orders 57, 336

conditional orders, in Excel 55, 334

configure TWS 25, 38

connect() 111

connecting to TWS

using ActiveX for Excel 321

connectionClosed() 128, 196, 265

Contract 216, 285

contract details 87

Contract Details page

in ActiveX for Excel 357

Contract Details page in Excel 86

Contract Details page toolbar 87

contract details, requesting in ActiveX for Excel 357

contract parameters

samples in Java 303

ContractDetails 217, 286

contractDetails() 139, 205, 273

contractDetailsEx() 139

createComboLegList() 124  
createContract() 124  
createExecutionFilter() 124  
createOrder() 124  
createScannerSubscription() 125  
createTagValueList 125  
createUnderComp() 125  
creating a ticker in ActiveX for Excel 324  
creating a ticker in Excel spreadsheet 41  
CSFB algo parameters 410  
current time  
    requesting in ActiveX for Excel 322  
currentTime() 128, 196, 264

## D

DDE defined 35  
DDE for Excel 35  
    downloading 37  
    getting started with 36  
    macros 96  
    modules 96  
    named ranges in 97  
    open the spreadsheet 39  
    syntax 98  
    viewing the code 95  
DDE for Excel for Advisors 91  
DDE for Excel reference 95  
DDE for Excel spreadsheet pages 40

DDE links  
    removing 43  
DDE syntax 98  
delta-neutral RFQs 437  
determine a futures contract in Java 304  
determine a stock in Java 304  
determine an option contract in Java 303  
disconnect() 111  
disconnecting from TWS  
    using ActiveX for Excel 322  
downloading API components 318  
downloading API software 37  
Dynamic Data Exchange 35

## E

Eclient Socket methods 252  
EClient Socket methods 247-249, 251-255, 258-259, 261-263  
EClientSocket functions 179  
EClientSocket() 180, 248  
Eclipse  
    running Java test client in 242  
eConnect() 180, 248  
eDisconnect() 180, 248  
Enable DDEclients setting in TWS 38  
EqualQuantity Method 312  
errMsg() 128

- error messages
  - viewing in ActiveX for Excel 370
- error() 196, 265
- EWrapper methods 264
- Excel
  - Advisors page 91
  - Bond Contract Details 88
  - Contract Details page 86
  - Historical Data page 74
  - Market Depth page 89
  - Market Scanner page 79
  - market scanner parameters 81
  - starting market scanner 80
  - viewing your portfolio in 73
- Excel Advanced Order page 63
- Excel API 35
  - getting market data 42
  - supported order types 47
- Excel DDE 317
  - pages 320
  - supported order types 332
- Excel DDE sample spreadsheet, installing 318
- Excel DDE, extended order attributes 342
- Excel DDE, Extended Order attributes page 342
- Excel DDE, financial advisor support 309
- Excel DDE, supported order types 332
- Excel modules 96
- Excel sample spreadsheet 40
  - opening 39
- Excel spreadsheet
  - Advanced Orders page 58
  - Basic Orders page 44
  - Conditional Orders page 54
  - Executions page 65
  - Extended Order Attributes page 48
  - Open Orders page 63
  - placing orders 45
  - removing all links 43
  - setting log detail level 43
  - setting processing rate 42
  - setting refresh rate 42
  - Tickers page 41
- execDetails() 205, 273
- execDetailsEnd() 141, 206, 274
- execDetailsEx() 141
- Execution 214, 283
- Execution page toolbar 66
- execution reporting, for financial advisors 311
- execution reporting, in ActiveX for Excel 350
- execution reports
  - running in Excel 67
- ExecutionFilter 215, 284
- executions
  - viewing in Excel 66

## Executions page

- in ActiveX for Excel 350

- Executions page in Excel 65

- Executions Reporting page in Excel 66

- executions, viewing in ActiveX for Excel 351

- exerciseOptions() 185, 252

- exerciseOptionsEx() 115

- exercising options

- in ActiveX for Excel 350

- expired contracts

- historical data in Excel 75

- extended order attributes 423

- applying to individual or groups of orders 343

- applying to orders 49

- extended order attributes in Excel 50

- Extended Order Attributes page

- in ActiveX for Excel 342

- in Excel 48

- extended order attributes, manually programming
  - in ActiveX for Excel 49, 343

## F

- FA account groups in Excel 93

- FA information

- in ActiveX for Excel 348

- FA managed account codes in Excel 68

- FA managed accounts, in ActiveX for Excel 347

- Portfolio page 349

- FA orders 93

- allocating shares to a single account 367

- allocation profiles 369

- using account group and method 368

- FA orders in ActiveX for Excel 366

- FA page in Excel 91

- filtering executions in Excel 66

- financial advisors 307

- allocation methods for account groups 312

- execution reporting for 311

- orders and account configuration 308

- financial advisors, Excel DDE support for 309

- financial advisors, support by other API technologies 310

- fundamental data

- in ActiveX for Excel 365

- report types 366

- fundamental data() 282

- fundamental ratios

- in ActiveX for Excel 365

- FUNDAMENTAL\_RATIOS tickType 394

- fundamentalData() 148, 213

- Fundamentals page

- in ActiveX for Excel 365

## G

- General page

- toolbar buttons 323

generic tick types 392

getting started

    DDE for Excel 36

    with ActiveX for Excel

        sample spreadsheet 318

## H

historical data

    duration and bar size settings 388

    viewing in ActiveX for Excel 353

historical data in Excel 75

    query specification fields 77

historical data limitations 387

Historical Data page

    in ActiveX for Excel 352

    query specification fields 354

Historical Data page in Excel 74

Historical Data page toolbar 76

historicalData() 146, 211, 280

## I

IB Gateway

    running the API through 26

IBAlgo parameters 404

IBAlgos 402, 404

IBDividends tick type example 399

IComboLeg 154

IComboLegList 155

ICommissionReport 151

IContract 151

IContractDetails 153

IExecution 149

IExecutionFilter 150

if-filled order, in Excel 56, 335

Index Premium data 441

installing Excel DDE sample spreadsheet 318

instrument codes for market scanners 432

IOrder 155

IOrderComboLeg 163

IOrderState 163

IScannerSubscription 164

isConnected() 180, 248

ITagValue 165

ITagValueList 165

IUnderComp 165

## J

Java 233

    combination orders 300

    linking to TWS 234

    sample program 238

Java API overview 245

Java code samples 303-304

    for FAs 314-315

Java EClient Socket methods 247

Java EWrapper methods 264-269, 271-276, 280-282

java sample program 244

    classes 244

Java SocketClient properties 283-286, 288-289,  
    297, 299

Java Test Client 244

    overview 244

    running 238

Java Test Client and Eclipse 242

## L

limitations

    of historical data requests 387

linking to TWS, using ActiveX 106

location codes for market scanners 432

log detail in Excel 43

Log page

    in ActiveX for Excel 370

logging 435

## M

macros in Excel 96

managedAccounts() 143, 207, 276

market data in Excel 42

market depth

    requesting in ActiveX for Excel 328

Market Depth page

    in ActiveX for Excel 327

    toolbar buttons in ActiveX for Excel 328

    using ActiveX for Excel 328

    using in Excel 90

Market Depth page in Excel 89

Market Depth page toolbar 91

Market Scanner page

    in ActiveX for Excel 362

Market Scanner page in Excel 79

Market Scanner page toolbar 82

market scanner parameters

    in ActiveX for Excel 364

market scanner parameters in Excel 81

market scanner subscription

    starting in Excel 80

market scanner subscription, starting in ActiveX for  
    Excel 363

market scanners 428

    available in Excel 82

    instruments and locations codes for 432

marketDataType() 132, 200, 269

message codes 374

mini options - DDE for Excel 439

mini options - socket clients 438

modifying orders in ActiveX for Excel 330

modifying orders in the DDE for Excel API 45

modifying orders, on Conditional Orders page 57, 336

## N

Name Manager in Excel 97

named ranges in Excel 97

NetLiq Method 312

nextValidId() 134, 202, 271

## O

open orders

removing in Excel 64

viewing in Excel 64

Open Orders page

ActiveX for Excel 344

in Excel 63

Open Orders page toolbar 65

open orders, viewing in ActiveX for Excel 345

opening the ActiveX sample program 109

openOrder() 202, 271

openOrderEx() 134

options

exercising in ActiveX for Excel 350

Order 220, 289

order IDs 31

order in Excel

modifying 45

order status event 427

order status for partial fills 427

order types 402

order types in Excel DDE 332

order types, in Excel 332

OrderComboLeg 289

orders 29

in Excel 59-63

placing in ActiveX for Excel 329

orders and account configuration, for financial advisors 308

orders in Excel API 45

orders in Java

for a single managed account 314

for an account group 315

for an allocation profile 314

orders, modifying in ActiveX for Excel 330

OrderState 227, 297

orderStatus() 132, 200, 269

overview 23

Java API 245

## P

pages 320

pages in Excel spreadsheet 40

partial fills and order status 427

PctChange Method 312

permId() 134

placeOrder() 183, 251

placeOrderEx() 113

placing orders

basket 329

combination order in ActiveX for Excel 330

conditional orders in Excel 334

placing orders in ActiveX for Excel 329

placing orders in Excel 45



portfolio data in FA managed accounts, in ActiveX for Excel 349

Portfolio page

in FA managed accounts, in ActiveX for Excel 349

Portfolio page in Excel 73

Portfolio page toolbar 74

portfolio, viewing in FA managed accounts, in ActiveX for Excel 349

position() 145, 210, 279

positionEnd() 146, 210, 279

POSIX 371

running client on Windows machine 372

premium data 441

price-change order, in Excel 57, 335

processing rate in Excel 42

## Q

query specification fields for historical data in Excel 77

query specification fields, on Historical Data page in ActiveX for Excel 354

## R

real time bars

in ActiveX for Excel 361

realtimeBar() 147, 212, 281

receiveFA() 143, 207, 276

recommendations for using API 28

reference 373

refresh rate in Excel 42

refresh rate, for market depth in Excel 328

refresh rate, on ActiveX for Excel Tickers page 325

registering third-party ActiveX controls 107

relative orders

in Excel 63

relative orders, in ActiveX for Excel 341

removing DDE links 43

replaceFA() 187, 255

reqAccountSummary 118, 188, 256

reqAccountUpdates() 117, 185, 253

reqAllOpenOrders 252

reqAllOpenOrders() 114, 184

reqAutoOpenOrders() 114, 184, 252

reqContractDetails() 186, 254

reqContractDetailsEx() 116

reqCurrentTime() 111, 180, 249

reqExecutions() 185, 253

reqExecutionsEx() 116

reqFundamentalData 125, 194

reqFundamentalData() 262

reqHistoricalData() 190, 259

reqHistoricalDataEx() 120

reqIds() 115, 184

reqIDs() 252

reqManagedAccts() 117, 187, 255

reqMarketDataType() 113, 183, 251

reqMktData() 181, 249

reqMktDataEx() 112  
reqMktDepth() 186, 254  
reqMktDepthEx() 116  
reqNewsBulletins() 117, 186, 254  
reqOpenOrders() 114, 184, 251  
reqPositions() 120, 190, 258  
reqRealTimeBars() 193, 261  
reqRealTimeBarsEx() 123  
reqScannerParameters() 122, 192, 258  
reqScannerSubscription() 192, 259  
reqScannerSubscriptionEx() 122  
Request for Quote 437  
request market depth in Excel 90  
requestFA() 117, 187, 255  
requesting bond contract details in ActiveX for Excel 359  
requesting bond contract details in Excel 88  
requesting contract details in ActiveX for Excel 357  
requesting contract details in Excel 87  
requesting market data, in ActiveX for Excel 325  
requesting market depth  
    in ActiveX for Excel 328  
Reuters global fundamentals  
    in ActiveX for Excel 365  
RFQs 437  
RTVolume 400  
running execution reports in Excel 67

running the API through IB Gateway 26

## S

sample program

    ActiveX 109

    C++ 178

    Java 238

sample spreadsheet

    Conditional Orders page 333

    opening 319

    pages in 320

sample spreadsheet, installing 318

scale orders

    in ActiveX for Excel 341

    in Excel 62

scannerData() 211, 280

scannerDataEnd() 147, 212, 281

scannerDataEx() 146

scannerParameters() 146, 211, 280

ScannerSubscription 227, 297

server log level

    setting in ActiveX for Excel 322

serverVersion() 181, 249

setLogLevel() 181

setServerLogLevel() 111, 248

SHORTABLE tick 393

smart combo routing 434

smartCombotRoutingParams 434

Socket Client Properties, in Java 285

SocketClient Properties 214-217, 219-220, 227, 229

SocketClient properties, in Java API 283

software

    downloading 37

spreadsheet pages 40

starting market scanner in ActiveX for Excel 363

supported order types 402

supported order types in Excel 47

supported time zones 433

## T

tables 373

TAG values for FUNDAMENTAL\_RATIOS 394

TestJavaClient 238

third-party controls, for ActiveX 107

tick types 389

tickEFP() 131, 199, 268

ticker

    creating in Excel 41

ticker, creating in ActiveX for Excel 324

Tickers page

    in ActiveX for Excel 323

    in Excel 41

    requesting market data in ActiveX for Excel 325

    setting the refresh rate in ActiveX for Excel 325

    toolbar buttons in ActiveX for Excel 325

    using 41

Tickers page toolbar 43

Tickers page, using ActiveX for Excel 324

tickGeneric() 130, 198, 267

tickOptionComputation() 129, 197, 267

tickPrice() 128, 265

tickPrice()Class EWrapper Functions 196

tickSize() 129, 197, 266

tickSnapshotEnd() 131, 199, 269

tickString() 130, 198, 268

time zones 433

toolbar

    Historical Data page 76

toolbar buttons

    Advanced Order page 63

    on ActiveX for Excel Advanced Orders page 342

    on ActiveX for Excel Advisors page 369

    on ActiveX for Excel Basic Orders page 332

    on ActiveX for Excel Bond Contract Details page 360

    on ActiveX for Excel Contract Details page 358

    on ActiveX for Excel Executions page 352

    on ActiveX for Excel Fundamentals page 366

    on ActiveX for Excel Historical Data page 356

    on ActiveX for Excel Market Depth page 328

    on ActiveX for Excel Market Scanner page 364

- on ActiveX for Excel Open Orders page 346
- on ActiveX for Excel Portfolio page 350
- on ActiveX for Excel Real Time Bars  
page 362
- on Conditional Orders page 336
- on FA managed accounts, in ActiveX for Excel  
Account page 348
- toolbar buttons, on ActiveX for Excel Bulletins  
page 327
- toolbar buttons, on ActiveX for Excel General  
page 323
- toolbar buttons, on ActiveX for Excel Tickers  
page 325
- toolbars
  - Advisors page 94
  - Basic Orders page 47
  - Bond Contract Details 89
  - Conditional Orders page 58
  - Contract Details 87
  - Execution page 66
  - Market Depth page 91
  - Market Scanner page 82
  - Open Orders page 65
  - Portfolio page 74
  - Tickers page 43
- trailing stop limit orders
  - Excel 61
- trailing stop limit orders, in ActiveX for  
Excel 340
- troubleshooting 442

- TWS
  - linking from Java 234
- TWS API settings 32
- TWS log file 435
- TWS precautionary settings 29
- TWS, configuring for API 25
- TWS, linking from C++ 172
- TWS, linking using ActiveX 106
- TwConnectionTime() 181, 249
- TwSocketClient.dll 172
  - linking to 172

## U

- updateMktDepthL2() 275
- UnderComp 229, 299
- uninstalling the API software 34
- updateAccountTime() 138, 204, 273
- updateAccountValue() 134, 203, 271
- updateMktDepth() 141, 206, 274
- updateMktDepthL2() 142, 207
- updateNewsBulletin() 138, 204, 275
- updatePortfolio() 203, 272
- updatePortfolioEx() 137
- using Account page in ActiveX for Excel 347
- using Account page in Excel 68
- using API components 24
- using the ActiveX for Excel sample spreadsheet 320
- using the ActiveX for Excel Tickers page 324

using the Market Depth page in ActiveX for  
Excel 328

using the Tickers page 41

util module in Excel 96

## V

VB\_API\_sample.vbp

opening 109

viewing code in Excel 95

viewing executions in Excel 66

viewing executions, in ActiveX for Excel 351

viewing historical data in ActiveX for Excel 353

viewing historical data in Excel 75

viewing open orders in ActiveX for Excel 345

viewing portfolio data in FA managed accounts,  
in ActiveX for Excel 349

viewing your portfolio in Excel 73

Visual Basic editor 95

Visual Basic sample program, for ActiveX 109

VOL orders

in ActiveX for Excel 338

volatility orders

in Excel 60

## W

winError() 196