

# Getting Started with the TWS ActiveX API Guide

Download the software to your PC and the demo version allows you to access your account via an internet browser, and is always available. The full version runs in memory and may run faster, but it requires a license key. To download to

Select Trader Workstation

Username and

Underlying	Exchange	Description	News Time	Time in Force	Bid Size	Bid	Ask	Ask Size	Last	Change	Volume
US Stocks											
ONLY	SMART	Stock (NMS)	18:59	1/14/08	8	25.42	25.42	8	25.42	-0.24	82.000K
AJD	SMART	Stock	20:54	1/14/08	3	104.50	104.64	2	104.62	+0.60	37.100K
SEPR	SMART	Stock (NMS)	17:45	1/13/08	1	99.32	99.36	3	99.32	+0.12	46.700K
IBM	SMART	Stock	07:02	1/14	1	111.42	111.42	4	111.42	-0.38	963.400K
BA	SMART	Stock	09:34	1/16	1	17.69	17.70	1	17.70	-0.16	1.270M
F	SMART	Stock			5	5.94	5.95	1	5.95	-0.03	2.100M
CAT	SMART	Stock			1	6.46	6.46	1	6.46	+0.05	456.000K
AAPL	SMART	Stock (NMS)			1	117.17	117.17	1	117.17	-2.87	4.300M
IBB	SMART	Stock (NMS)			1	117.17	117.17	1	117.17	+0.22	47.700K
MSFT	SMART	Stock			1	117.17	117.17	1	117.17	+0.22	6.430M
Options											
IBM	SMART	APN									30
IBM	SMART	APN									30
Futures											
ES	GLCC	Cr									138
FOREX											
EUR											
USD											



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

**Getting Started with the TWS ActiveX API**  
**September 2014**  
**Supports TWS API Release 9.71**

© 2014 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. TWS Javahelp version 013, March 25, 2008.

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

# Contents

<b>1</b>	<b>Introduction .....</b>	<b>7</b>
	How to Use this Book .....	8
	Organization .....	8
	Part 1: Introducing the TWS ActiveX API .....	8
	Part 2: Preparing to Use the TWS ActiveX API .....	9
	Part 3: Market Data .....	9
	Part 4: Orders and Executions .....	9
	Part 5: Additional Tasks .....	9
	Part 6: Where to Go from Here.....	9
	Footnotes and References .....	9
	Icons .....	10
	Document Conventions.....	11
<b>2</b>	<b>TWS and the ActiveX API.....</b>	<b>13</b>
	Chapter 1 - What is Trader Workstation?.....	14
	What Can You Do with TWS? .....	16
	A Quick Look at TWS .....	16
	The TWS Quote Monitor .....	16
	The Order Ticket .....	16
	Real-Time Account Monitoring .....	17
	Chapter 2 - Why Use the TWS ActiveX API?.....	18
	TWS and the API .....	18
	Available API Technologies .....	19
	An Example .....	19
<b>3</b>	<b>Preparing to Use the ActiveX API .....</b>	<b>21</b>
	Chapter 3 - Install an IDE .....	22
	Programming Languages, ActiveX and Microsoft Visual Studio 2008.....	22
	Chapter 4 - Download the API Software .....	24
	Chapter 5 - Connect to the ActiveX Sample Application .....	28
	Multiple Versions of the Sample Application .....	28

Connecting to the VB.NET Sample Application from Visual Studio 2008 .....	29
What's Next .....	31

## **4 Market Data..... 33**

Chapter 6 - Connecting to TWS.....	34
ActiveX Sample Application Basic Framework.....	34
dlgMainWnd.....	35
What Happens When I Click the Connect Button? .....	35
Connect Button Event Handler .....	37
Disconnecting from a Running Instance of TWS .....	38
Chapter 7: Requesting and Canceling Market Data .....	39
What Happens When I Click the Req Mkt Data Button?.....	40
Req Mkt Data Button Event Handler .....	40
States of the dlgOrder Object.....	41
The reqMktDataEx() Method .....	42
ActiveX Events that Return Market Data .....	44
Getting a Snapshot of Market Data.....	45
Canceling Market Data.....	45
The cancelMktData() Method.....	46
Chapter 8 - Requesting and Canceling Market Depth .....	47
What Happens When I Click the Req Mkt Depth Button?.....	48
Req Mkt Depth Button Event Handler .....	48
The reqMktDepthEx() Method.....	49
ActiveX Events that Return Market Depth .....	50
Canceling Market Depth.....	52
The cancelMktDepth() Method.....	52
Chapter 9 - Requesting and Canceling Historical Data .....	53
What Happens When I Click the Historical Data Button? .....	54
Historical Data Button Event Handler .....	54
The reqHistoricalDataEx() Method.....	55
ActiveX Events that Return Historical Data .....	56
Canceling Historical Data .....	57
The cancelHistoricalData() Method.....	58
Chapter 10 - Requesting and Canceling Real Time Bars.....	59
What Happens When I Click the Real Time Bars Button? .....	60
Real Time Bars Button Event Handler .....	60

The reqRealTimeBarsEx() Method .....	61
ActiveX Events that Return Real Time Bars .....	62
Canceling Real Time Bars.....	63
The cancelRealTimeBars() Method .....	63
Chapter 11 - Subscribing to and Canceling Market Scanner Subscriptions .....	64
What Happens When I Click the Market Scanner Button?.....	65
Market Scanner Button Event Handler .....	66
Requesting Scanner Parameters .....	67
Subscribing to a Market Scanner .....	67
The scannerDataEnd() Event.....	68
Cancel a Market Scanner Subscription .....	69
Chapter 12: Requesting Contract Data.....	70
What Happens When I Click the Req Contract Data Button? .....	71
Req Contract Data Button Event Handler .....	71
The reqContractDetailsEx() Method.....	72
The contractDetailsEx() Event .....	72
The contractDetailsEnd() Event .....	73

## **5 Orders and Executions..... 75**

Chapter 13: Placing and Canceling an Order .....	76
What Happens When I Place an Order? .....	77
Place Order Button Event Handler .....	77
The placeOrderEx() Method .....	78
The orderStatus() Event.....	79
Canceling an Order .....	80
The cancelOrder() Method .....	80
Modifying an Order .....	81
Requesting "What-If" Data before You Place an Order.....	81
Placing Combo Orders .....	82
Combo Legs Processing .....	84
Combo Legs Code Example.....	85
Placing Algo Orders .....	87
Algo Order Processing .....	89
Chapter 14: Exercising Options.....	90
What Happens When I Click the Exercise Options Button? .....	91
Exercise Options Button Event Handler.....	91

The exerciseOptionsEx() Method .....	92
Chapter 15: Extended Order Attributes .....	93
What Happens When I Click the Extended Button? .....	94
Chapter 16: Requesting Open Orders .....	95
Running Multiple API Sessions .....	95
The Difference between the Three Request Open Orders Buttons.....	96
What Happens When I Click the Req Open Orders Button?.....	96
Req Open Orders Button Event Handler .....	97
The openOrderEx() Event .....	97
The openOrderEnd() Event .....	98
What Happens When I Click the Req All Open Orders Button? .....	98
Req All Open Orders Button Event Handler.....	98
What Happens When I Click the Req Auto Open Orders Button? .....	99
Req Auto Open Orders Button Event Handler.....	99
The reqAutoOpenOrders() Method .....	99
Chapter 17: Requesting Executions .....	100
What Happens When I Click the Req Executions Button? .....	100
Req Executions Button Event Handler.....	101
The reqExecutionsEx() Method .....	101
The execDetails() Method .....	102
The execDetailEnd() Event.....	102
<b>6 Additional Tasks .....</b>	<b>103</b>
Chapter 18 - Requesting the Current Time .....	104
What Happens When I Click the Current Time Button? .....	104
Current Time Button Event Handler.....	104
The reqCurrentTime() Method .....	105
The currentTime() Event Handler.....	105
Chapter 19: Requesting the Next Order ID .....	106
The reqIds() Method .....	106
The nextValidId() Event .....	106
Chapter 20: Subscribing to News Bulletins .....	107
What Happens When I Click the Req News Bulletins Button? .....	107
Req News Bulletins Button Event Handler .....	108
The reqNewsBulletins() method.....	108
The updateNewsBulletin() Method .....	109

Canceling News Bulletins .....	109
Chapter 21: Viewing and Changing the Server Logging Level .....	110
What Happens When I Click the Log Configuration Button? .....	110
Log Configuration Button Event Handler .....	111
The setServerLogLevel() Method .....	111
<b>7 Where to Go from Here.....</b>	<b>113</b>
Chapter 22 - Linking to TWS using the TWS ActiveX API .....	114
Registering Third-Party ActiveX Controls .....	115
Chapter 23 - Additional Resources .....	116
Help with Visual Basic and VB.NET Programming .....	116
Help with the TWS ActiveX API .....	116
The API Reference Guide .....	116
The API Beta and API Production Release Notes.....	116
The TWS API Webinars.....	117
API Customer Forums .....	117
IB Customer Service .....	117
IB Features Poll.....	117
<b>Appendix A: Appendix A - Extended Order Attributes .....</b>	<b>119</b>
<b>Appendix B: Appendix B - Account Page Values.....</b>	<b>123</b>





# Introduction

You might be looking at this book for any number of reasons, including:

- You love IB's TWS, and are interested in seeing how using its API can enhance your trading.
- You use another online trading application that doesn't provide the functionality of TWS, and you want to find out more about TWS and its API capabilities.
- You never suspected that there was a link between the worlds of trading/financial management and computer programming, and the hint of that possibility has piqued your interest.

Or more likely you have a reason of your own. Regardless of your original motivation, you now hold in your hands a unique and potentially priceless tome of information. Well, maybe that's a tiny bit of an exaggeration. However, the information in this book, which will teach you how to access and manage the robust functionality of IB's Trader Workstation through our TWS ActiveX for Visual Basic API, could open up a whole new world of possibilities and completely change the way you manage your trading environment. Keep reading to find out how easy it can be to build your own customized trading application.



*If you are a Financial Advisor who trades for and allocates shares among multiple client accounts and would like more information about using the ActiveX API, see the [Getting Started with the TWS ActiveX API for Advisors Guide](#).*

**Note:** This guide supports API releases no higher than 9.71.

# How to Use this Book

Before you get started, you should read this section to learn how this book is organized, and see which graphical conventions are used throughout.

Our main goal is to give active traders and investors the tools they need to successfully implement a custom trading application (i.e. a trading system that you can customize to meet your specific needs), and that doesn't have to be monitored every second of the day. If you're not a trader or investor you probably won't have much use for this book, but please, feel free to read on anyway!

We should also tell you that throughout this book we use the TWS ActiveX API sample application to demonstrate how we implemented the API. However, our sample application is not our primary focus. Our main objective is to introduce you to the methods, events and parameters in the ActiveX API that you will need to learn to build your own custom trading application. You can use the sample application as a starting point.



*Throughout this book, we use the acronym "TWS" in place of "Trader Workstation." So when you see "TWS" anywhere, you'll know we're talking about Trader Workstation.*



*Before you read any further, we need to tell you that this book focuses on the TWS side of the ActiveX API - we don't really help you to learn Visual Basic. If you aren't a fairly proficient Visual Basic programmer, or at least a very confident and bold beginner, this may be more than you want to take on. We suggest you start with a beginner's Visual Basic programming book, and come back to us when you're comfortable with the language.*

## Organization

We've divided this book into five major sections, each of which comprises a number of smaller subsections, and each of **those** have even smaller groupings of paragraphs and figures...well, you get the picture. Here's how we've broken things down:

### Part 1: Introducing the TWS ActiveX API

The chapters in this section help you answer those important questions you need to ask before you can proceed - questions such as "What can TWS do for me?" and "Why would I use an API?" and "If I WERE to use an API, what does the ActiveX API have to offer me?" and even "What other API choices do I have?"

If you already know you want to learn about the TWS API, just skip on ahead.

## **Part 2: Preparing to Use the TWS ActiveX API**

Part 2 walks you through the different things you'll need to do before your API application can effectively communicate with TWS. We'll help you download and install the API software, configure TWS, and get the sample application up and running. A lot of this information is very important when you first get started, but once it's done, well, it's done, and you most likely won't need much from this section once you've completed it.

## **Part 3: Market Data**

Part 3 gets you working with the ActiveX sample application to get market data. You'll learn how to request, receive and cancel market data, market depth, historical data, real time bars, run market scanners and get contract data. We'll tell you exactly what methods you need to use to send info to TWS, and just what TWS will send you back. We've already documented the method parameters, descriptions and valid values in the *API Reference Guide*, but we have provided a lot of those details here for your convenience.

## **Part 4: Orders and Executions**

Part 4 takes you through the order-related tasks in the ActiveX API sample application. You'll learn how the API handles the process of placing and canceling an order, viewing open orders, and viewing executions. Here too we provide the methods, events and parameters used for these trading tasks.

## **Part 5: Additional Tasks**

Part 5 continues your path through the ActiveX API sample application by describing the methods, events and parameters used for the rest of the buttons in the sample application, including how to get the current system time and how to request and cancel news subscriptions.

## **Part 6: Where to Go from Here**

After filling your head with boatfuls of API knowledge, we wouldn't dream of sending you off empty-handed! Part 7 includes some additional information about linking to TWS using our ActiveX for Visual Basic API, then tells you how to keep abreast of new API releases (which of course means new features you can incorporate into your trading plan), how to navigate the Interactive Brokers website to find support and information, and what resources we recommend to help you answer questions outside the realm of IB support, questions such as "Why isn't my Visual Studio working?"

## **Footnotes and References**

<sup>1</sup>Any symbols displayed are for illustrative purposes only and are not intended to portray a recommendation.

## Icons



### **TWS-Related**



### **ActiveX Tip**

When you see this guy, you know that there is something that relates specifically to TWS: a new feature to watch for, or maybe something you're familiar with in TWS and are looking for in the API.

These Visual Basic tips are things we noted and think you might find useful. They don't necessarily relate only to TWS. We don't include too many of these, but when you see it you should check it out - it will probably save you some time.



### **Important!**



### **Take a Peek!**

You may want to take a peek, but it isn't the end of the world if you don't.



### **Go Outside!**

This icon denotes references outside of this book that we think may help you with the current topic, including links to the internet or IB site, or a book title.

## Document Conventions

Here's a list of document conventions used in the text throughout this book.

Convention	Description	Examples
<b>Bold</b>	Indicates: <ul style="list-style-type: none"><li>• menus</li><li>• screens</li><li>• windows</li><li>• dialogs</li><li>• buttons</li><li>• tabs</li><li>• keys you press</li><li>• names of classes and methods</li></ul>	When you click the <b>Req Mkt Data</b> button...  Press <b>Ctrl+C</b> to copy...
<i>Italics</i>	Indicates: <ul style="list-style-type: none"><li>• commands in a menu</li><li>• objects on the screen, such as text fields, check boxes, and drop-down lists</li></ul>	To access the users' guide, under the <b>Software</b> menu, select <i>Trader Workstation</i> , then click <i>Users' Guide</i> .
<b>Code samples</b>	Code samples appear gray boxes throughout the book.	



# TWS and the ActiveX API

The best place to start is by getting an idea of what Trader Workstation (TWS), is all about. In this section, first we'll describe TWS and some of its major features. Then we'll explain how the API can be used to enhance and customize your trading environment. Finally, we'll give you a summary of some of the things the ActiveX API can do for you!

Here's what you'll find in this section:

- [Chapter 1 - What is Trader Workstation?](#)
- [Chapter 2 - Why Use the ActiveX API?](#)

## Chapter 1 - What is Trader Workstation?

Interactive Brokers' Trader Workstation, or TWS, is an online trading platform that lets you trade and manage orders for all types of financial products (including stocks, bonds, options, futures and Forex) on markets all over the world - all from your choice of two workspaces:


- The Advanced Order Management workspace, which is a single spreadsheet-like screen.

Contract	Last	Change	Change %	Bid Size	Bid	Ask	Ask Size	Position	Avg Price	P&L	Submitter
IBM	193.01	-0.14	-0.07%	2	193.01	193.74	1				
YHOO	29.65	+0.03	0.10%	31	29.50	29.65	9				
AAPL	447.31	-2.81	-0.62%	1	447.02	447.50	1				
GOOG	887.76	0.00	0.00%	1	885.00	887.25	1				
FB	42.74	+0.23	0.54%	22	42.73	42.75	24				
IBM Jan17'14 2...	C4.36										

- Mosaic, a single, comprehensive and intuitive workspace which provides easy access to Trader Workstation's trading, order management and portfolio functionality.





 To get a little bit of a feel for TWS, go to the IB website and try the TWS demo application. Its functionality is slightly limited and it only supports a small number of symbols, but you'll definitely get the idea. Once you have an approved, funded account you'll also be able to use PaperTrader, our simulated trading tool, with paper-money funding in the amount of \$1,000,000, which you can replenish at any time through TWS Account Management.

## What Can You Do with TWS?

So, what can you do with TWS? For starters, you can:

- Send and manage orders for all sorts of products (all from the same screen!);
- Monitor the market through Level II, NYSE Deep Book and IB's Market Depth;
- Keep a close eye on all aspects of your account and executions;
- Use Technical, Fundamental and Price/Risk analytics tools to spot trends and analyze market movement;
- Completely customize your trading environment through your choice of modules, features, tools, fonts and colors, and user-designed workspaces.

Basically, almost anything you can think of TWS can do - or will be able to do soon. We are continually adding new features, and use the latest technology to make things faster, easier and more efficient. As a matter of fact, it was this faith in technology's ability to improve a trader's success in the markets (held by IB's founder and CEO Thomas Peterffy) that launched this successful endeavor in the first place. Since the introduction of TWS in 1995, IB has nurtured this relationship between technology and trading almost to the point of obsession!

## A Quick Look at TWS

This section gives you a brief overview of the most important parts of TWS.

### The TWS Quote Monitor

First is the basic TWS Quote Monitor. It's laid out like a spreadsheet with rows and columns. To add tickers to a page, you just click in the Underlying column, type in an underlying symbol and press Enter, and walk through the steps to select a product type and define the contract. Voila! You now have a live market data line on your trading window. It might be for a stock, option, futures or bond contract. You can add as many of these as you want, and you can create another window, or trading page, and put some more on that page. You can have any and all product types on a single page, maybe sorted by exchange, or you can have a page for stocks, a page for options, etc. Once you get some market data lines on a trading page, you're ready to send an order.

### The Order Ticket

What? An order ticket? Sure, we have an order ticket if that's what you really want. But we thought you might find it easier to simply click on the bid or ask price and have us create a complete order line instantly, right in front of your eyes! Look it over, and if it's what you want click a button to transmit the order. You can easily change any of the order parameters right on the order line. Then just click the green Transmit guy to transmit your order! It's fast and it's easy, and you can even customize this minimal two-click procedure (by creating hotkeys and setting order defaults for example) so that you're creating and transmitting orders with just ONE click of the mouse.

## **Real-Time Account Monitoring**

TWS also provides a host of real-time account and execution reporting tools. You can go to the Account Window at any time to see your account balance, total available funds, net liquidation and equity with loan value and more. You can also monitor this data directly from your trading window using the Trader Dashboard, a monitoring tool you can configure to display the last price for any contracts and account-related information directly on your trading window.

So - TWS is an all-inclusive, awesome powerful trading tool. You may be wondering, "Where does an API fit in with this?" Read on to discover the answer to that question.



*For more information on TWS, see the TWS Users' Guide on our web site.*

## Chapter 2 - Why Use the TWS ActiveX API?

OK! Now that you are familiar with TWS and what it can do, we can move on to the amazing API. If you actually read the last chapter, you might be thinking to yourself "Why would I want to use an API when TWS seems to do everything." Or you could be thinking "Hmmm, I wonder if TWS can... fill in the blank?" OK, if you're asking the first question, I'll explain why you might need the API, and if you're asking the second, it's actually the API that can fill in the blank.

TWS has the capability to do tons of different things, but it does them in a certain way and displays results in a certain way. It's likely that our development team, as fantastic as they are, hasn't yet exhausted the number of features and way of implementing them that all of you collectively can devise. So it's very likely that you, with your unique way of thinking, will be or have been inspired by the power of TWS to say something like "Holy moly, I can't believe I can really do all of this with TWS! Now if I could only just (fill in the blank), my life would be complete!"

That's where the API comes in. Now, you can fill in the blank! It's going to take a little work to get there, but once you see how cool it is to be able to access functionality from one application to another, you'll be hooked.

### **TWS and the API**

In addition to allowing you pretty much free reign to create new things and piece together existing things in new ways, the API is also a great way to automate your tasks. You use the API to harness the power behind TWS - in different ways.

Here's an analogy that might help you understand the relationship between TWS and the API. Start by imagining TWS as a book (since TWS is constantly being enhanced, our analogy imagines a static snapshot of TWS at a specific point in time). It's the reference book you were looking for, filled with interesting and useful information, a book with a beginning, middle and end, which follows a certain train of logic. You could skip certain chapters, read Chapter 10 first and Chapter 2 last, but it's still a book. Now imagine, in comparison, that the API is the word processing program in which the book was created with the text of the book right there. This allows you access to everything in the book, and most importantly, it lets you continually change and update material, and automate any tasks that you'd have to perform manually using just a book, like finding an index reference or going to a specific page from the table of contents.

The API works in conjunction with TWS and with the processing functions that run behind TWS, including IB's SmartRouting, high-speed order transmission and execution, support for over 40 orders types, etc. TWS accesses this functionality in a certain way, and you can design your API to take advantage of it in other ways.

## Available API Technologies

IB provides a suite of custom APIs in multiple programming languages, all to the same end. These include Java, C++, Active X for Visual Basic and .NET, ActiveX for Excel, DDE for Excel (Visual Basic for Applications, or VBA), CSharp and POSIX. This book focuses specifically on just one, the ActiveX version. Why would you use ActiveX over the other API technologies? The main reason might be that you are an ActiveX expert. If you don't know ActiveX or any other programming language, you should take a look at the Excel/DDE API, which has a much smaller learning curve. But if you know ActiveX, this platform offers more flexibility than the DDE for Excel (the DDE is only supported in Windows), and provides very high performance.



*For more information about our APIs, see the Trading Technology > API Solutions page on our web site.*

## An Example

It's always easier to understand something when you have a real life example to contemplate. What follows is a simple situation in which the API could be used to create a custom result.

TWS provides an optional field that shows you your position-specific P&L for the day as either a percentage or an absolute value. Suppose you want to modify your position based on your P&L value? At this writing, the only way to do this would be to watch the market data line to see if the P&L changed, and then manually create and transmit an order, but only if you happened to catch the value at the right point. Hmmmmm, I don't think so! Now, enter the API! You can instruct the API to automatically trigger an order with specific parameters (such as limit price and quantity) when the P&L hits a certain point. Now that's power! Another nice benefit of the API is that it gives you the ability to use the data in TWS in different ways. We know that TWS provides an extensive Account Information window that's chock-full of everything you'll ever want to know about your account status. The thing is, it's only displayed in a TWS window, like the one on the next page.



Account

File

Portfolio

Currencies

Configure

Help

Lovely though it is, what if you wanted to do something else with this information? What if you want it reflected in some kind of banking spreadsheet where you log information for all accounts that you own, including your checking account, Interactive Brokers' account, 401K, ROIs, etc? Again - enter the API!

You can instruct the API to get any specific account information and put it wherever it belongs in a spreadsheet. The information is linked to TWS, so it's easy to keep the information updated by simply linking to a running version of TWS. With a little experimenting, and some help from the *API Reference Guide* and the *TWS Users' Guide*, you'll be slinging data like a short-order API chef in no time!

There are a few other things you must do before you can work with the TWS ActiveX API. The next chapter gets you geared up and ready to go.

# Preparing to Use the ActiveX API

Although the API provides great flexibility in implementing your automated trading ideas, all of its functionality runs through TWS. This means that you must have a TWS account with IB, and that you must have your TWS running in order for the API to work. This section takes you through the minor prep work you will need to complete, step by step.

Here's what you'll find in this section:

- [Chapter 3 - Install an IDE](#)
- [Chapter 4 - Download the API Software](#)
- [Chapter 5 - Connect to the ActiveX API Sample Application](#)



*We want to tell you again that this book focuses on the TWS side of the ActiveX API - we don't really help you to learn Visual Basic. Unless you are a fairly proficient VB programmer, or at least a very confident and bold beginner, this may be more than you want to take on. We suggest you start with a beginner's Visual Basic programming book, and come back to us when you're comfortable with the language.*

## Chapter 3 - Install an IDE

OK, well we've already said that you need to know Visual Basic before you can successfully implement your own TWS ActiveX for Visual Basic API application, and there's a good chance you already have the tools you'll need downloaded and installed. But in case you don't, we'll quickly walk you through what you need, which is simply an integrated development environment (IDE) that supports Microsoft Visual Basic and the Microsoft .NET framework.



*In this book we use Microsoft Visual Studio 2008 as the IDE of choice. We'll try to keep the Visual Studio-specific instructions to a minimum, but if you're using another IDE you'll have to interpret those instructions to fit your development environment. If you're using Visual Studio 2008 and aren't totally familiar with it, we recommend browsing through the How Do I section of the online help, which you can access from Visual Studio's Help menu.*

### Programming Languages, ActiveX and Microsoft Visual Studio 2008



*Read this section if you are new to Microsoft Visual Studio 2008 or simply need a refresher course in Visual Studio 2008's supported languages.*

Microsoft Visual Studio 2008 includes Microsoft's .NET framework and supports the following .NET programming languages:

- Visual Basic (also called VB.NET)
- Visual C#
- Visual C++

Our TWS ActiveX API is written in Visual Basic. Another way to say this is that we have created our TWS ActiveX component using Visual Basic code. What does this have to do with you? It simply means that you can use Visual Studio 2008 to work with our API Visual Basic code to create your own ActiveX-based trading application.



*Programming languages and associated technologies can have many names, depending on the person doing the naming. VB.NET is just another name for Microsoft's latest version of the Visual Basic language. Our ActiveX API was written in Visual Basic 6, but our API software comes with two different versions of the sample application: one prebuilt Visual Basic version, and one VB.NET version that hasn't been built. If you're confused about this, don't worry. Chapter 5 gives you more information about the two different versions of the ActiveX sample application. For purposes of this book, we use the terms "ActiveX sample application" and "Visual Basic sample application" interchangeably.*



Anyway, we're not giving you too much here, but we are assuming you have enough savvy to find this stuff, download it, and install it. This is a tough line for us to walk, because we're really focusing on getting started with the TWS ActiveX API, not on getting started with ActiveX or Visual Basic. If you're having trouble at this point, you should probably start with the TWS DDE for Excel API to get your feet wet!

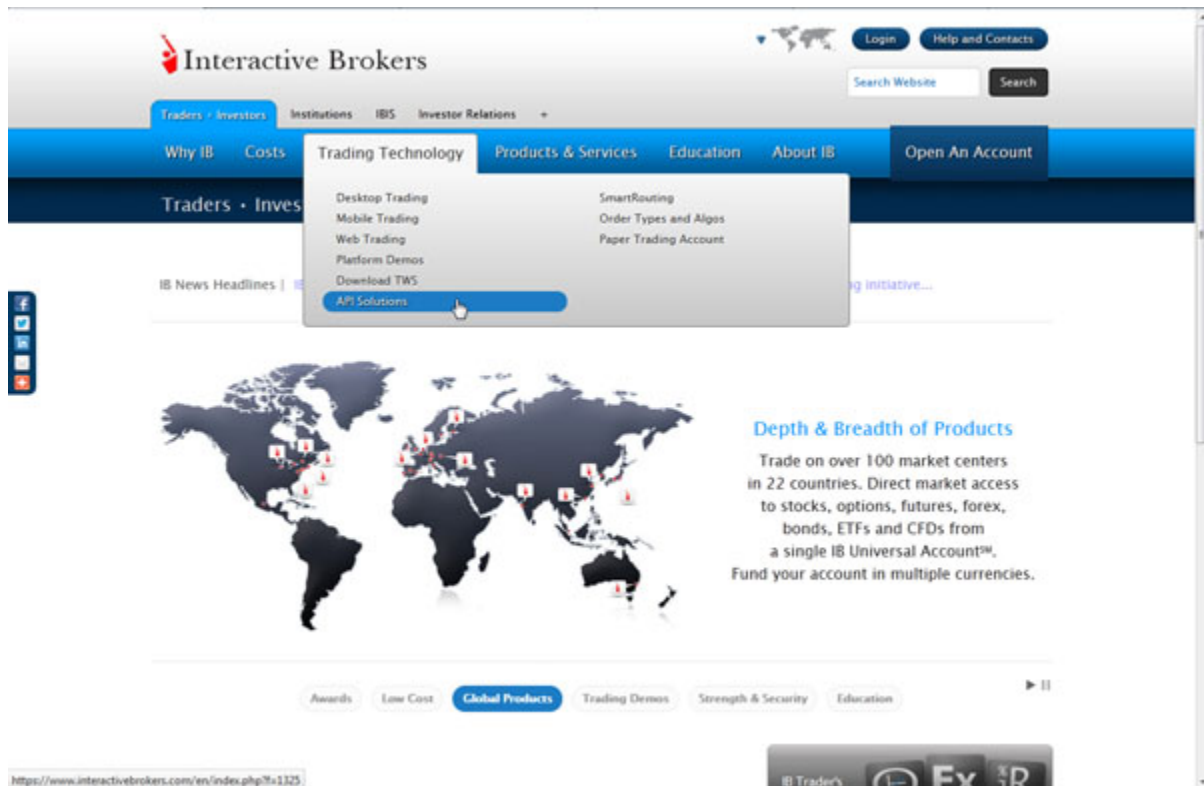
Once you have these pieces downloaded and installed, you can go to the IB website and download the TWS API software.

## Chapter 4 - Download the API Software

Next, you need to download the API software from the IB website.

### Step 1: Download the API software.

This step takes you out to the IB website at <https://individuals.interactivebrokers.com/en/index.php?f=1325>. The menus are along the top of the homepage. Hold your mouse pointer over the Trading Technology menu, then click *API Solutions*.



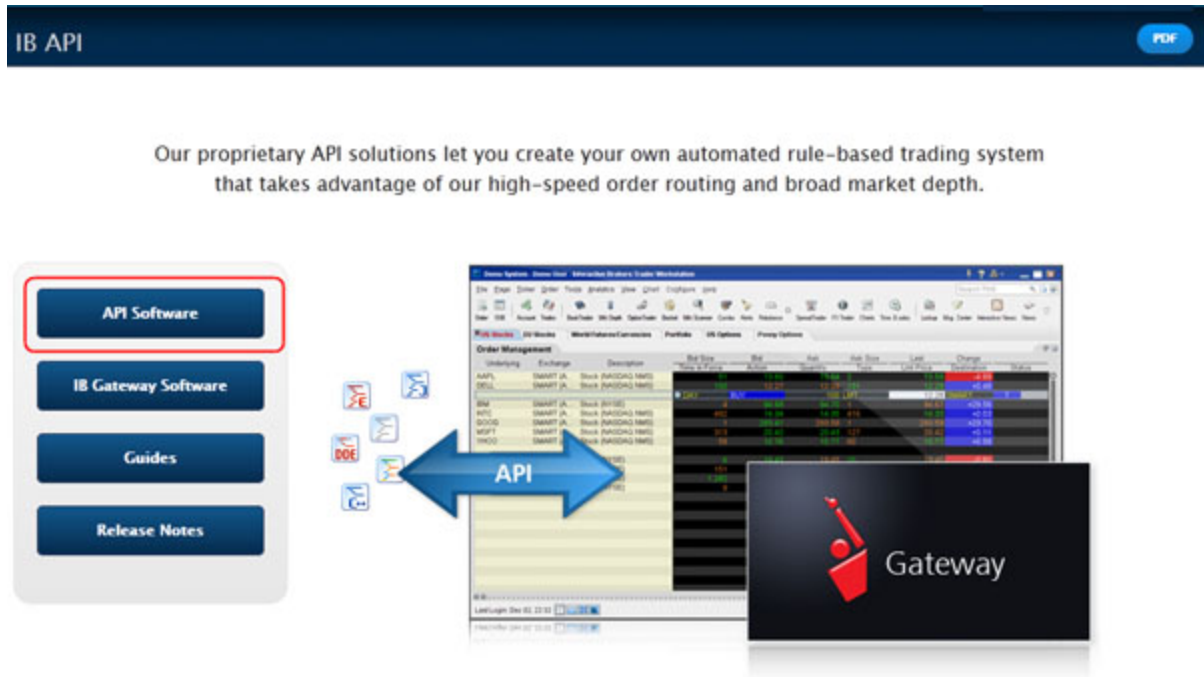
On the API Solutions page, click the **more info** button next to IB API.



#### IB API **more info**

Build your own trading applications in Excel (using DDE or ActiveX), C++, Posix C++, Java, and Visual Basic for ActiveX using IB's Application Programming Interface (API). The IB API connects through Trader Workstation (TWS) or the IB Gateway, and does not require additional technical overhead such as a dedicated FIX server.

On the next page that appears, click the **API Software** button.



#### IB API Software

Program traders may build their own add-on applications in Excel (using DDE or ActiveX), C++, Posix C++, Java, and Visual Basic for ActiveX with our proprietary IR Application Program Interface (API) which requires connectivity via either the TWS or the IR Gateway. We encourage API users to test their API

Click the **I Agree** button on the license agreement page to open the API software download page.

This displays the IB API page which shows a table with buttons that initiate the API software download process for Windows, MAC or Unix platforms. When available, there will also be a Windows Beta version of the software. Find the OS you need, then click the button to download the API installation program.

Interactive Brokers API Software

Download Contribute

Windows	Mac / Unix
<b>IB API for Windows</b> Version: API 9.69 Release Date: July 1 2012	<b>IB API for Mac/Unix</b> Version: API 9.69 Release Date: July 1 2012
<b>IB API Beta for Windows</b> Version: API beta 9.70 Release Date: Sep 9 2013	<b>IB API Beta for Mac / Unix</b> Version: API beta 9.70 Release Date: Sep 9 2013
<b>IB API Previous for Windows</b>	<a href="#">Click for Mac Instructions</a> <a href="#">Click for Unix Instructions</a>
Includes the C++ Socket, Java Socket, DDE, Active X APIs, and sample code for each.	Includes the Java Socket API, Posix C++ Socket API and sample code for each.
Support: <a href="#">API Reference Guide</a> or <a href="#">IB Discussion Forum</a>	

**Note:**  
As a reminder, the use of the IB API as a means of disseminating information, including market data or any other licensed or copyrighted information, to third parties or non-registered IB customers is strictly prohibited without prior written approval of Interactive Brokers.



*For this book, we assume that you are using Windows. If you're using a different operating system (Mac, Unix), be sure to adjust the instructions accordingly!*

In the Windows column, click the **IB API for Windows** button. This opens a File Download box, where you can decide whether to save the installation file, or open it. We recommend you choose **Save** and then select a place where you can easily find it, like your desktop (you choose the path in the Save in field at the top of the Save As box that opens up). Once you've selected a good place to put it, click the **Save** button. It takes seconds to download the executable file. Note that the API installation file is named for the API version; for example, *TWS API Install 9.69.01.msi*.



*We'll usually be stressing just the opposite, but at this point, you need to make sure TWS is **not** running. If it is, you won't be able to install the API software.*

## Step 2: Install the API software.

Next, go to the place where you saved the file (for example, your desktop or some other location on your computer), and double-click the API software installation file icon. This starts the installation wizard, a simple process that displays a series of dialogs with questions that you must answer.



Once you have completed the installation wizard, the sample application installs, and you're ready to open the ActiveX sample application, connect to TWS, and get started using the ActiveX API!



*ActiveX controls must be registered before you can use them. When you install our API software, the TWS ActiveX control, `Tws.ocx`, is automatically registered for you. So if you install the Beta API software, the beta version of the TWS ActiveX control is registered and the production version of the ActiveX control is no longer registered. This is important to remember because if you try to run the production version of the ActiveX sample application after having installed the Beta API software, you will get errors unless you re-register the production version of the TWS ActiveX control!*

## Chapter 5 - Connect to the ActiveX Sample Application

OK, you've got all the pieces in place. Now that we're done with the prep work, it's time to get down to the fun stuff.

Although the API provides great flexibility in implementing your automated trading ideas, all of its functionality runs through TWS. This means that you must have a TWS account with IB, and you must have TWS running in order for the API to work. This section describes how to enable TWS to connect to the ActiveX API. Note that if you don't have an account with IB, you can use the Demo TWS system to check things out.. If you DO have an account, we recommend opening a linked PaperTrader test account, which simulates the TWS trading environment, and gives you \$1,000,000 in phantom cash to play with.

### Multiple Versions of the Sample Application

We mentioned this before in passing, but let's talk about it again here. That's right, we've included more than one version of the ActiveX sample application with our API software:

- **Visual Basic:** This sample application was programmed in Visual Basic.
- **VB.NET:** This sample application looks and works exactly like the Visual Basic version, except that it is compatible with Microsoft's VB.NET version of Visual Basic. This sample application has not been built, so you have to open it in MS Visual Studio and build and run it from there.

For this book, it doesn't matter which sample application you run, although we recommend running the VB.NET version. The API methods, events and parameters are the same, and the sample application itself looks and works the same way.

## Connecting to the VB.NET Sample Application from Visual Studio 2008

If you prefer, you can run the VB.NET sample application from within Microsoft Visual Studio 2008. Here's how:

### Step 1: Log into TWS.

OK, log into TWS by clicking **Login > Trader Workstation Latest** or **Trader Workstation** (which is the TWS version released prior to the latest version), or run the Demo available by clicking the **Demo** button on the Trading Technology > Desktop Trading page on our website.

### Step 2: Enable TWS to support the ActiveX API.

This step is the also the same for both versions of the sample application; again, we're repeating it for your convenience.

Click the **Edit** menu, and then click *Global Configuration*. In the Configuration window, click *API* in the left pane, then click *Settings*, which reveals several options on the right side of the window. Check the *Enable ActiveX and Socket Clients* check box and click **OK**.

### Step 3: Run the VB.NET Sample Application.

Here's how to do this:

- 1 Navigate to the samples\TestActiveX\_VB.NET folder in your API installation folder, then rename the \*.vbproj file.

We recommend renaming the original \*.vbproj file because Visual Studio will regenerate the project files to be compatible with both 32- and 64-bit operating systems.

- 2 Open Visual Studio 2008, then select *Open > Project/Solution* from the **File** menu.
- 3 Browse to your TWS API installation folder, then select the \*.vbproj file that you renamed in the samples\TestActiveX\_VB.NET folder.
- 4 The Visual Studio Conversion Wizard opens. Click **Next** to run the Wizard and convert the project to the Visual Studio 2008 format.

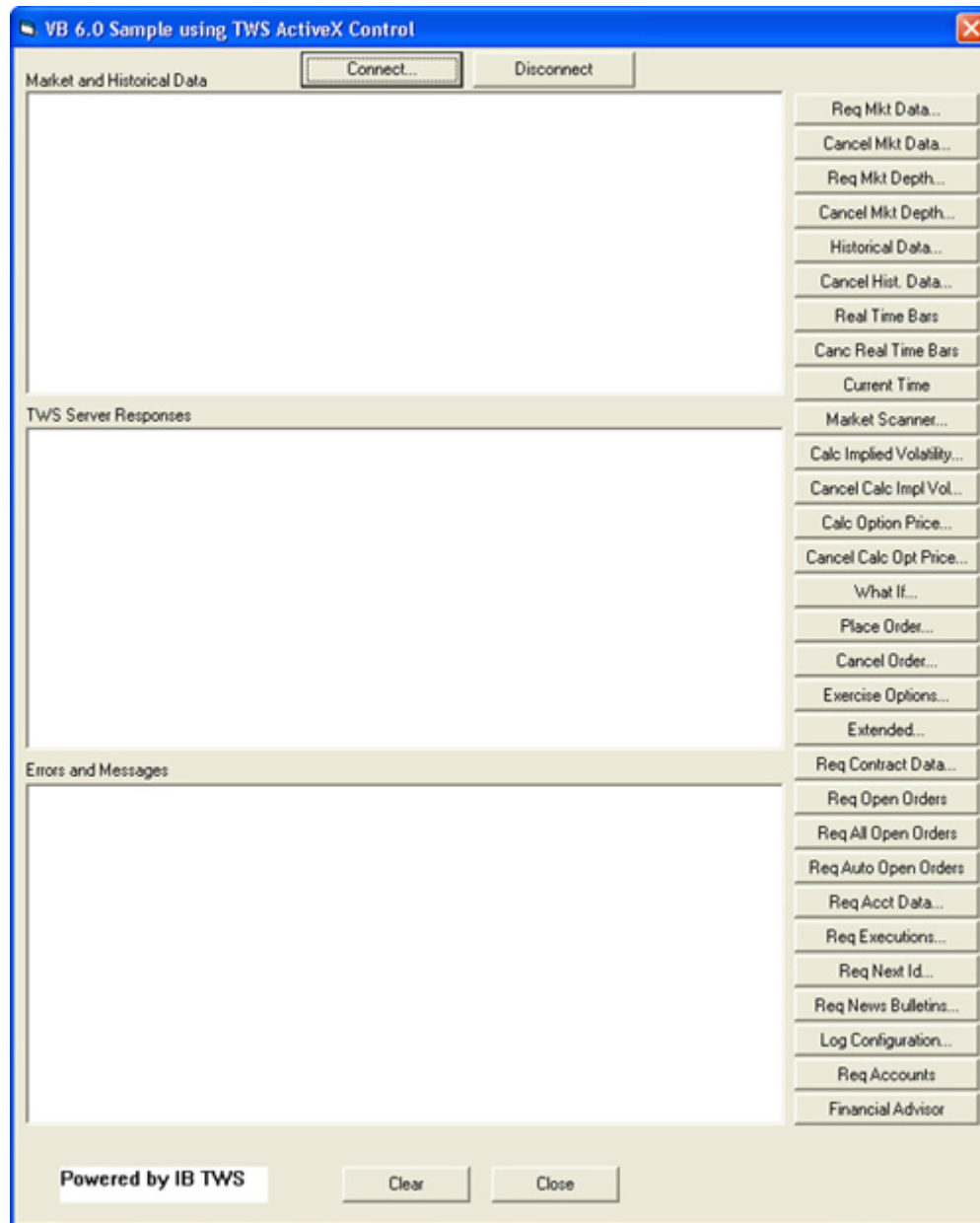
The initial Conversion Wizard screen is shown on the next page.



- 5 When Conversion Wizard is complete, press **Ctrl+F5**.



And here's the sample application:



## What's Next

Part 3 focuses on performing market data-related trading tasks defined by the action buttons in the sample client. We'll take a quick, general look at what's going on behind the GUI. Then we'll walk through the basics of requesting market data using the TWS ActiveX API.



# Market Data

You've completed the prep work, and you have the ActiveX sample application up and running. This section of the book starts with a description of the basic framework of the sample application, then reviews the TWS ActiveX API methods associated with each trading task.

This section describes how to connect the sample application to TWS and how to perform market data-related tasks such as requesting and canceling market data, historical data and real time bars, as well as how to subscribe to market scanners and get contract data. We'll show you the methods, events and parameters behind these trading tasks.

Here's what you'll find in this section:

- [Chapter 6 - Connecting to TWS](#)
- [Chapter 7 - Requesting and Canceling Market Data](#)
- [Chapter 8 - Requesting and Canceling Market Depth](#)
- [Chapter 9 - Requesting and Canceling Historical Data](#)
- [Chapter 10 - Requesting and Canceling Real Time Bars](#)
- [Chapter 11 - Subscribing to and Canceling Market Scanner Subscriptions](#)
- [Chapter 12 - Requesting Contract Data](#)

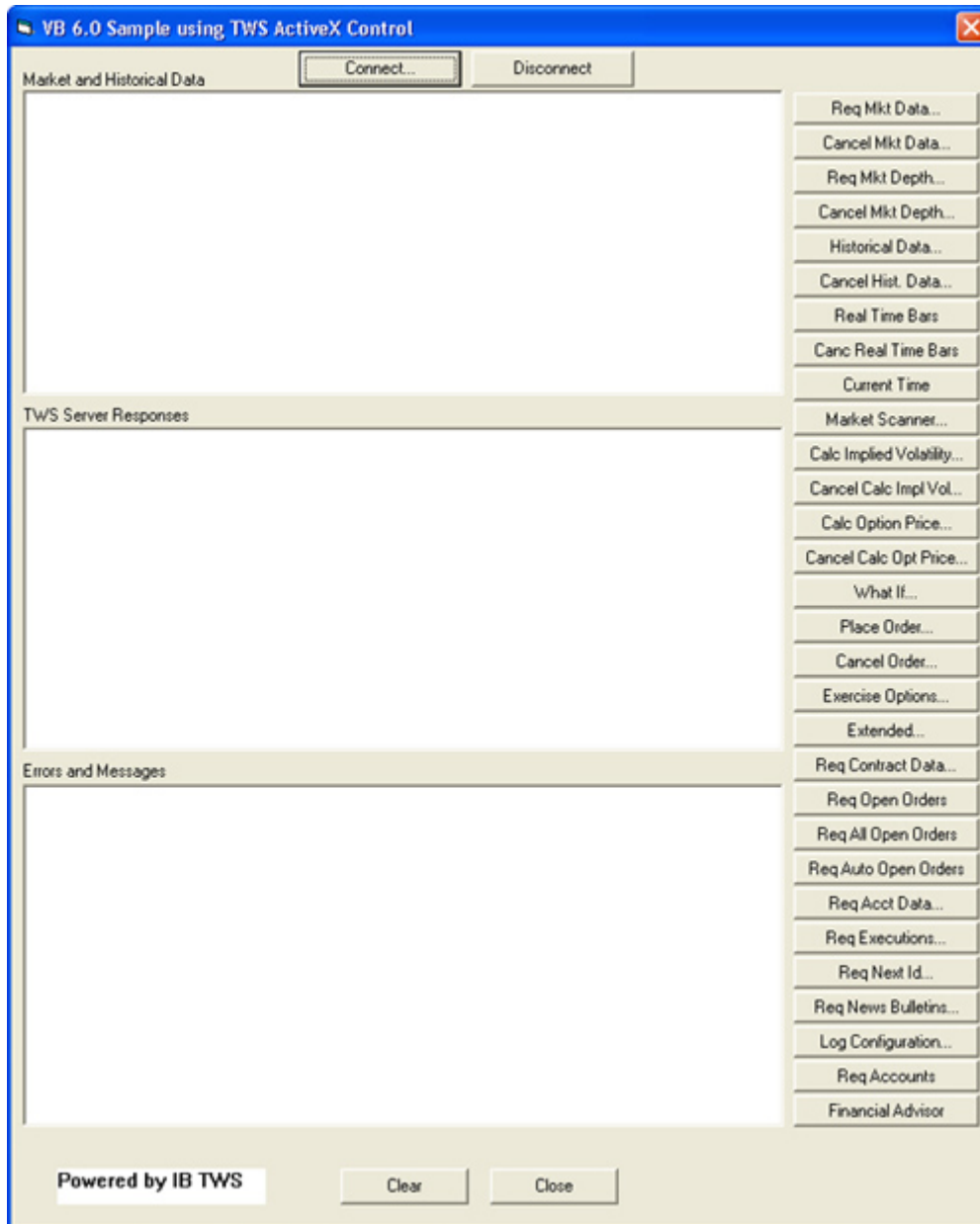
Using the ActiveX sample application is a good way to practice locating and using the reference information in the [API Reference Guide](#). With the sample program, you can compare the data in the sample message with the method parameters in the *API Reference Guide*.

## Chapter 6 - Connecting to TWS

This chapter describes the basic framework of the ActiveX sample application and what happens when you connect and disconnect to a running instance of TWS.

### ActiveX Sample Application Basic Framework

Let's take a look at the basic framework of the ActiveX sample application and the ActiveX API. Here's the ActiveX sample application when you first run it:



## dlgMainWnd

The sample application pictured on the previous page is defined in the code as **dlgMainWnd**, which is a standard Windows form. If you look at the code for the form, in addition to Windows-generated code, you will see declarations, a few public functions, event handlers for button and TWS events, and XML Utilities already defined for you.



*dlgMainWnd is unique to the sample application; it is not part of the TWS ActiveX API and therefore is not documented in the API Reference Guide.*

Every button in the sample application has a corresponding button event in the code that defines what happens when you click that button. The event handlers for the buttons on the main sample application window are all located in the code for **dlgMainWnd**.

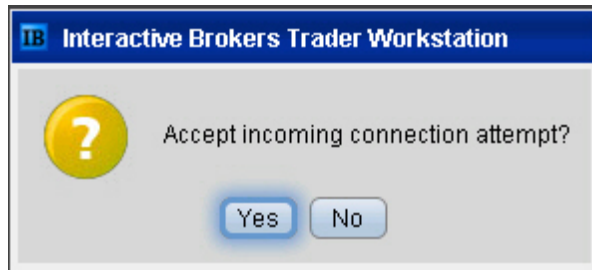
## What Happens When I Click the Connect Button?



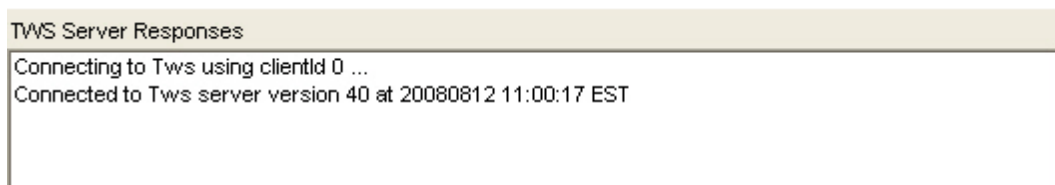
The very first thing you do with the ActiveX sample application is to connect it to a running instance of TWS. You click the **Connect** button to do this. This displays the Connect dialog, shown below.

- IP Address - This is the IP address of the system where Trader Workstation is installed. If TWS and the API are installed on the same computer, you can either leave this field blank, or enter **127.0.0.1** to indicate local host.
- Port - **7496** is the default port number, but you can modify this if you need to.
- Client ID - This is used to determine each API connection. Each connection must have a unique Client ID.

Then you can enter the *IP Address*, *Port* and *Client Id* values in the input fields of the dialog. When you click the **OK** button, the Connect dialog closes and a message indicating that you are connecting to TWS is displayed in the *TWS Server Responses* text panel on the sample application window. A confirmation dialog appears in TWS; click **Yes** to confirm that you want to connect. Note that this dialog does not appear if the IP Address of your connection is local host.



Finally, a message indicating that you have successfully connected to TWS appears in the *TWS Server Responses* text panel on the sample application window.



That's what happens on the user side of things. Now let's see what happens behind the scenes.

## Connect Button Event Handler

When you click the **Connect** button, the event handler called **cmdConnect\_Click** and defined in `dlgMainWnd` and runs. Here is what the code for the event handler looks like:

### Connect Button Event Handler

```
Private Sub cmdConnect_Click(ByVal eventSender As System.Object, ByVal eventArgs As
System.EventArgs) Handles cmdConnect.Click
    ' assume this is a non Financial Advisor account. If it is the managedAccounts()
    ' event will be fired.
    m_faAccount = False

    m_dlgConnect.ShowDialog()
    If m_dlgConnect.ok Then
        With m_dlgConnect
            Call m_utils.addListItem(Utls.List_Types.SERVER_RESPONSES, _
                "Connecting to Tws using clientId " & .clientId & " ...")
            Call Tws1.connect(.hostIP, .port, .clientId)
            If (Tws1.serverVersion() > 0) Then
                Dim msg As String
                msg = "Connected to Tws server version " & Tws1.serverVersion()
                & _
                " at " & Tws1.TwsConnectionTime()
                Call m_utils.addListItem(Utls.List_Types.SERVER_RESPONSES, msg)
            End If
        End With
    End If
End Sub
```

**cmdConnect\_Click** does the following:

- Sets `m_faAccount` equal to `False`, which identifies this as a non-Financial Advisor (FA) account. If this is an FA account, then the `managedAccounts()` method would be called. But you don't have to worry about this right now.
- Displays the Connect dialog.
- Calls `m_utils.addItem(Utils.List_Types.SERVER_RESPONSES)` to display the message "Connecting to Tws using client Id" and inserts the client Id you entered in the Connect dialog. `m_utils.addItem` is defined in the `Utils` object.
- Calls the ActiveX **connect()** method and passes the input values for the *IP Address* (`.hostIP`), *Port* (`.port`) and *Client ID* (`.clientId`) fields to TWS as parameters of the **connect()** method.
- If the connection to TWS is successfully established, calls `m_utils.addItem(Utils.List_Types.SERVER_RESPONSES)` again, this time to display the message "Connected to Tws server version" and inserts the TWS server version.

## Disconnecting from a Running Instance of TWS



To disconnect from a running instance of TWS, click the **Disconnect** button in the ActiveX sample application. When you do this, **cmdDisconnect\_Click**, the event handler for the **Disconnect** button, calls the **disconnect()** ActiveX method, which disconnects the sample application from TWS. It's that simple!

Here's what **cmdDisconnect\_Click** looks like:

### Disconnect Button Event Handler

```
Private Sub cmdDisconnect_Click(ByVal eventSender As System.Object, ByVal
eventArgs As System.EventArgs) Handles cmdDisconnect.Click
    Call Tws1.disconnect()
End Sub
```

When the sample application disconnects from TWS, an additional event handler in `dlgMainWnd` runs. When this event handler runs, the **connectionClosed()** ActiveX event provides notification that the TWS-API connection has been broken.

### connectionClosed() Event Handler

```
Private Sub Tws1_connectionClosed(ByVal eventSender As System.Object, _
ByVal eventArgs As System.EventArgs) Handles Tws1.connectionClosed
    Call m_utils.addItem(Utils.List_Types.ERRORS, "Connection to Tws
has been closed")
    ' move into view

    lstErrors.TopIndex = lstErrors.Items.Count - 1
End Sub
```

OK, got all of that? Great! Now let's move on, and see what happens when you use the market data buttons.



## Chapter 7: Requesting and Canceling Market Data

This chapter describes how the ActiveX sample application requests and cancels market data. You click the **Req Mkt Data** button to display the Request Market Data dialog, then enter information in the appropriate fields and click **OK**.

The following image shows the Request Market Data dialog and the fields you need to fill in to get market data.

**Request Market Data**

Id: 9

**Contract Description**

Contract Id: 0  
Symbol: QQQQ  
Type: STK  
Expiry:   
Strike: 0  
Right:   
Multiplier:   
Exchange: SMART  
Primary Exchange:   
Currency: USD  
Local Symbol:   
Include Expired: 0  
Sec Id Type:   
Sec Id:

**Order Description**

Action: BUY  
Quantity: 10  
Order Type: LMT  
Lmt/Opt Price / Volatility: 40  
Aux/Under Price: 0  
Good After Time:   
Good Till Date:   
FA Alloc:   
Combo Legs:   
Delta Neutral:   
Algo Params:

**Market Data**

Generic Tick Tags: 100,101,104,10  
☐ Snapshot

**Market Depth**

Max Market Depth Rows: 20

**Exercise Options**

Action (1 or 2): 1  
Quantity: 1  
Override (0 or 1): 0

**Historical Data**

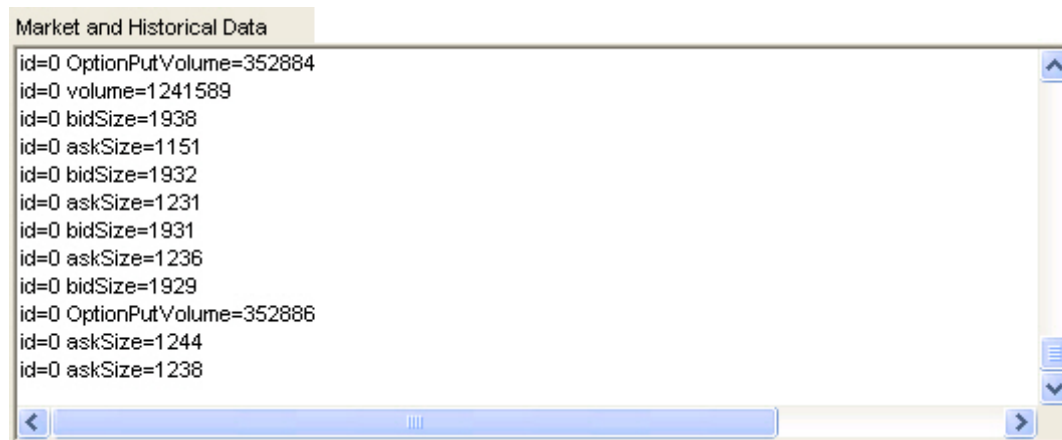
End Date/Time: 20110315 09:53:15  
Query Duration: 1 M  
Bar Size Setting: 1 day  
What to Show: TRADES  
Regular Trading Hours (1 or 0): 1  
Date Format Style (1 or 2): 1

Ok Cancel

## What Happens When I Click the Req Mkt Data Button?

Req Mkt Data...

Once you connect to TWS using the ActiveX sample application, you get market data by clicking the **Req Mkt Data** button, then entering an underlying and some other information in the Request Market Data dialog, such as symbol, type, and exchange, and clicking **OK**. The market data you request is displayed in the *Market and Historical Data* text panel, as shown below.



*The Symbol, Security Type, Exchange and Currency values are required for all instrument types. If your security type is STK, those four values are all you need. But if you're looking for the latest price on a Jan08 27.5 call, you need to give the method a bit more than that. The moral: be sure you include values in the appropriate fields based on what return values you want to get.*

That's what happens from a user's point of view. But what's really happening?

### Req Mkt Data Button Event Handler

When you click the **Req Mkt Data** button, the event handler **cmdReqMktData\_Click**, defined in **dlgMainWnd**, runs. Here is what the code for the event handler looks like:

#### Req Mkt Data Button Event Handler

```
Private Sub cmdReqMktData_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdReqMktData.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_MKT_DATA_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.reqMktDataEx(m_dlgOrder.orderId, m_contractInfo, _
            m_dlgOrder.genericTickTags, m_dlgOrder.snapshotMktData)
    End If
End Sub
```



*The code samples in this book may not look exactly like the code when you view it Visual Studio. Don't worry, the code is exactly the same. We've simply added a few extra line breaks so that the code samples fit the size of the pages in the book.*

**cmdReqMktData\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.REQ_MKT_DATA_DLG`. This initializes the Request Market Data dialog. (The different states of the `dlgOrder` object are described in the next section, below.)
- Displays the Request Market Data dialog.
- Calls the ActiveX method **reqMktDataEx()** when the **OK** button is clicked.

### States of the `dlgOrder` Object

Before we continue with market data requests using the ActiveX API sample application, let's stop and take a look at the Request Market Data dialog. This is the dialog that is pictured at the beginning of this chapter.

As you work through the different trading tasks described in this book, you will notice that the same dialog box is used for a variety of functions. This is the `dlgOrder` object, and it is used in slightly different forms to:

- request and cancel market data;
- place and cancel orders;
- request and cancel market depth;
- request contract details;
- request and cancel historical data;
- request and cancel real time bars
- exercise options.

Each "version" of the `dlgOrder` object has its own unique name and has only certain fields and buttons available. These different versions are defined in the code for the `dlgOrder` object, which identifies which sets of fields and buttons are available or unavailable for each version, and what text appears in the dialog's title bar.

So, for example, when you request market data, this dialog is called the Request Market Data dialog ("Request Market Data" appears in the title bar of the dialog), and only those fields required for market data requests are accessible. The fields required for the other trading tasks listed above are grayed out and unavailable.

The **cmdReqMktData\_Click** event handler initializes the correct version of the dialog on these lines of code:

```
m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_MKT_DATA_DLG), _
m_contractInfo, m_orderInfo, m_underComp, Me)
```

## The reqMktDataEx() Method

Now let's get back to requesting market data.

When you click **OK** in the sample application to submit a market data request, the **reqMktDataEx()** method sends your market data request to TWS and, if all the entries are valid, the requested data is returned by way of the **tickPrice()**, **tickSize()**, **tickGeneric()**, **tickOptionComputation()**, **tickString()** and **tickEFP()** events. What's really happening though is that the **reqMktDataEx()** method triggers a series of "tick" events, which return the market data from TWS. We'll look at these tick events a little later.

For now, let's find out which parameters are used for requesting market data. The **reqMktDataEx()** method looks like this:

```
Sub reqMktDataEx(ByVal tickerId As Integer, ByVal contract As
TWSLib.IContract, ByVal genericTicklist As String, ByVal snapshot As
Integer)
```

Parameter	Description
<b>tickerId</b>	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.
<b>contract</b>	This structure contains attributes used to describe the contract.
<b>genericTicklist</b>	A comma delimited list of generic tick types.
<b>snapshot</b>	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.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

As you can see from the table above, this method has four parameters, the first three of which correspond to the fields in the Request Market Data dialog that you fill in.

Now let's take another look at the Request Market Data dialog and see how and where it relates to the **reqMktDataEx()** method.

The circled sections in the picture above (*Ticker Description* and *Market Data*) correspond to the *contract* and *genericTickList* parameters in the **reqMktDataEx()** method. This means that the values you entered in the dialog are passed to TWS by the parameters in the **reqMktDataEx()** method. The *tickerId* parameter corresponds to the *Ticker ID* field in the dialog. The *snapshot* parameter is used to get a snapshot of market data; we'll describe this in more detail a little later in this chapter.

The *contract* object (*ICContract TWS COM object*) contains the properties that correspond to the fields in the *Ticker Description* section of the Request Market Data dialog. For a complete list of the properties in the *contract* structure, see the [API Reference Guide](#). You can ignore the other fields in the dialog right now because they represent parameters from different methods. Don't worry, we'll be revisiting them very soon!



*IContract is a TWS COM object that is created by the factory method **createContract()**. You **MUST** use the **createContract()** factory method to create the IContract object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.*

## ActiveX Events that Return Market Data

As we mentioned before, requested market data is returned to the sample application by way of the **tickPrice()**, **tickSize()**, **tickGeneric()**, **tickOptionComputation()**, **tickString()** and **tickEFP()** events, each of which returns a different part of the market data. This means that there is an event handler in dlgMainWnd for each “tick” event, and these events are triggered by the **reqMktDataEx()** method. Here are the “tick” events:

### tickPrice()

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

### tickSize()

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

### tickOptionComputation()

```
Sub tickOptionComputation(ByVal id As Integer, ByVal TickType As Integer, ByVal impliedVol As Double, ByVal delta As Double, ByVal modelPrice As Double, ByVal pvDividend As Double)
```

### tickGeneric()

```
Sub tickGeneric(ByVal tickerId As Integer, ByVal tickType As Integer, ByVal value As Double)
```

### tickString()

```
Sub tickString(ByVal Id As Integer, ByVal tickType As Integer, ByVal value As String)
```

### tickEFP()

```
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)
```



*For more details about these events and their parameters, see the [ActiveX Events](#) section of the API Reference Guide. For details about tick types and tick values, see the [Tick Types](#), [Generic Tick Types](#), and [Tick Values](#) topics in the API Reference Guide, available on our website.*

## Getting a Snapshot of Market Data

Another way to get market data from TWS to the ActiveX sample application is to get a snapshot of market data. A market data snapshot gives you all the market data in which you are interested for a contract for a single moment in time. What this means is that instead of watching the requested market data continuously scroll by in the *Market and Historical Data* text panel of the ActiveX sample application, you get a single "snapshot" of the data. This frees you from having to keep up with the scrolling data and having to cancel the market data request when you are finished.

To get snapshot market data, simply click the **Req Mkt Data** button, then fill in the appropriate fields in the Request Market Data dialog, and finally check the *Snapshot* check box and click **OK**.

*Snapshot* is a parameter of the **reqMktDataEx()** method.

## Canceling Market Data



When you click the **Cancel Mkt Data** button, the Cancel Market Data dialog appears. This is another version of the same dialog saw when we requested market data; the only difference is that when you cancel market data, all the fields in the dialog are grayed out except *Ticker Id* and *Contract Id*. Simply click **OK** to submit your cancellation of the market data.

So what happens in the code when you do this?

When you click the **Cancel Mkt Data** button, the **cmdCancelMktData\_Click** button event handler in `dlgMainWnd` runs.

### Cancel Mkt Data Button Event Handler

```
Private Sub cmdCancelMktData_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdCancelMktData.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.CANCEL_MKT_DATA_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.cancelMktData(m_dlgOrder.orderId)
    End If
End Sub
```

The **cmdCancelMktData** button event handler does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.CANCEL_MKT_DATA_DLG`. This initializes the Cancel Market Data dialog.
- Displays the Cancels Market Data dialog.
- Calls the ActiveX method **cancelMktData()** when the **OK** button is clicked.

**The cancelMktData() Method**

This method has a single parameter, *id*, which is the same ID that was specified in the reqMktDataEx() call for market data. The cancelMktData() method is shown below.

**cancelMktData()**

```
Sub cancelMktData(ByVal id As Integer)
```

Next we'll see how the ActiveX API handles requests for market depth.



## Chapter 8 - Requesting and Canceling Market Depth

This chapter discusses the methods for requesting and canceling market depth in the ActiveX sample application. We'll show you the methods and parameters behind the sample application and how they call the methods in the TWS ActiveX API.

For requesting market depth, you need to use the highlighted fields in the Request Market Depth dialog as shown here:

**Request Market Depth**

Id: 9

**Contract Description**

Contract Id: 0  
Symbol: DELL  
Type: STK  
Expiry:   
Strike: 0  
Right:   
Multiplier:   
Exchange: ISLAND  
Primary Exchange: ISLAND  
Currency: USD  
Local Symbol:   
Include Expired: 0  
Sec Id Type:   
Sec Id:

**Order Description**

Action: BUY  
Quantity: 10  
Order Type: LMT  
Lmt/Opt Price / Volatility: 40  
Aux/Under Price: 0  
Good After Time:   
Good Till Date:   
FA Alloc  
Combo Legs  
Delta Neutral  
Algo Params

**Market Data**

Generic Tick Tags: 100,101,104,10  
☐ Snapshot

**Historical Data**

End Date/Time: 20110315 09:56:15  
Query Duration: 1 M  
Bar Size Setting: 1 day  
What to Show: TRADES  
Regular Trading Hours (1 or 0): 1  
Date Format Style (1 or 2): 1

**Exercise Options**

Action (1 or 2): 1  
Quantity: 1  
Override (0 or 1): 0

**Market Depth**

Max Market Depth Rows: 20

Ok Cancel

## What Happens When I Click the Req Mkt Depth Button?

Req Mkt Depth...

Once you connect to TWS using the ActiveX sample application, you can request market depth by clicking the **Req Mkt Depth** button, then entering information in the *Ticker Description* fields in the Request Market Depth dialog and clicking **OK**. The market depth you request is displayed in the Market Depth dialog, as shown below.

Market Depth for:					
Bid					Ask
MM	Price	Size	cumSize	avgPrice	
	14.49	18	18	14.49	
	14.48	66	84	14.482143	
	14.47	26	110	14.479273	
	14.46	48	158	14.473418	
	14.45	37	195	14.468974	
MM	Price	Size	cumSize	avgPrice	
	14.5	38	38	14.5	
	14.51	34	72	14.504722	
	14.52	39	111	14.51009	
	14.53	34	145	14.514759	
	14.54	33	178	14.519439	
Close					

That's what happens from a user's point of view. Let's see what's going on behind the scenes.

### Req Mkt Depth Button Event Handler

When you click the **Req Mkt Depth** button, the event handler **cmdReqMktDepth\_Click**, defined in **dlgMainWnd**, runs. Here is what the code for the event handler looks like:

#### Req Mkt Depth Button Event Handler

```
Private Sub cmdReqMktDepth_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdReqMktDepth.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_MKT_DEPTH_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then

        m_dlgMktDepth.init()
        Call Tws1.reqMktDepthEx(m_dlgOrder.orderId, _
            m_contractInfo, m_dlgOrder.numRows)
        m_dlgMktDepth.ShowDialog()

        ' unsubscribe to mkt depth when the dialog is closed
        Call Tws1.cancelMktDepth(m_dlgOrder.orderId)

    End If

End Sub
```

**cmdReqMktDepth\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.REQ_MKT_DEPTH_DLG`. This initializes the Request Market Depth dialog. (Remember that there are several different “versions” of the `dlgOrder` object.)
- Displays the Request Market Depth dialog.
- Calls the ActiveX method **reqMktDepthEx()** when the **OK** button is clicked.

### The **reqMktDepthEx()** Method

When you click **OK** in the sample application to submit a market depth request, the **reqMktDepthEx()** method sends your request to TWS and, if all the entries are valid, the requested data is returned by way of the **updateMktDepth()** and **updateMktDepthL2** events. So the **reqMktDepthEx()** method triggers these two events in order to return market depth to the sample application.

Now let's see which parameters are used to request market depth. The **reqMktDepthEx()** method looks like this:

```
Sub reqMktDepthEx(ByVal tickerId As String, ByVal contract As
TWSLib.IContract, ByVal numRows As Integer)
```

Parameter	Description
<b>tickerId</b>	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.
<b>contract</b>	This object contains attributes used to describe the contract.
<b>numRows</b>	Specifies the number of market depth rows to return.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

The *contract* object (*IContract TWS COM object*) contains the properties that correspond to the fields in the *Ticker Description* section of the Request Market Data dialog. For a complete list of the properties in the *contract* structure, see the [API Reference Guide](#).



*IContract is a TWS COM object that is created by the factory method **createContract()**. You MUST use the **createContract()** factory method to create the IContract object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.*

## ActiveX Events that Return Market Depth

There are two ActiveX events that return market depth: **updateMktDepth()** and **updateMktDepthL2()**. The difference between them is that **updateMktDepthL2()** returns LII market depth. Both events are triggered by the **reqMktDepthEx()** method in `dlgMainWnd`.

The **updateMktDepth()** event, its parameters and the event handler in `dlgMainWnd` and are shown below. You can see the data that is returned by looking at the the event's parameters in the tables.

### updateMktDepth()

```
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
<b>id</b>	The ticker ID that was specified previously in the call to <code>reqMktDepth()</code>
<b>position</b>	Specifies the row ID of this market depth entry.
<b>operation</b>	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>
<b>side</b>	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>
<b>price</b>	The order price.
<b>size</b>	The order size.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

### updateMktDepth() Event Handler

```
Private Sub Tws1_updateMktDepth(ByVal eventSender As System.Object, _
ByVal eventArgs As AxTWSLib._DTwsEvents_updateMktDepthEvent) Handles _
Tws1.updateMktDepth
    m_dlgMktDepth.updateMktDepth(eventArgs.id, eventArgs.position, " ", _
eventArgs.operation, eventArgs.side, eventArgs.price, eventArgs.size)
End Sub
```

The **updateMktDepthL2()** event, its parameters and the event handler in `dlgMainWnd` are shown below. This method only applies to customers who have subscribed to LII market data (NYSE's Open Book and NASDAQ's Total View market data subscriptions). Again, the parameters returned by this event correspond to the market depth data being returned from TWS.

### updateMktDepthL2

```
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
<b>id</b>	The ticker ID that was specified previously in the call to <code>reqMktDepth()</code>
<b>position</b>	Specifies the row id of this market depth entry.
<b>marketMaker</b>	Specifies the exchange hosting this order.
<b>operation</b>	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>
<b>side</b>	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>
<b>price</b>	The order price.
<b>size</b>	The order size.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

### updateMktDepthL2 Event Handler

```
Private Sub Tws1_updateMktDepthL2(ByVal eventSender As System.Object, _
ByVal eventArgs As AxTWSLib._DTwsEvents_updateMktDepthL2Event) Handles _
Tws1.updateMktDepthL2
    m_dlgMktDepth.updateMktDepth(eventArgs.id, eventArgs.position, _
eventArgs.marketMaker, eventArgs.operation, eventArgs.side, _
eventArgs.price, eventArgs.size)
End Sub
```

## Canceling Market Depth



When you click the **Cancel Mkt Depth** button, the Cancel Market Depth dialog appears. Yes, this is yet another version of the same dialog saw when we requested market data and market depth; the only difference is that when you cancel market depth, all the fields in the dialog are grayed out except *Ticker Id* and *Contract Id*. Simply click **OK** to submit cancel market depth.

Let's see what happens in the code when you do this.

When you click the **Cancel Mkt Depth** button, the **cmdCancelMktDepth\_Click** button event handler in **dlgMainWnd** runs.

### Cancel Mkt Depth Button Event Handler

```
Private Sub cmdCancelMktDepth_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdCancelMktDepth.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.CANCEL_MKT_DEPTH_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.cancelMktDepth(m_dlgOrder.orderId)
    End If
End Sub
```

The **cmdCancelMktDepth\_Click** button event handler does the following:

- Sets the state of the **dlgOrder** object by setting **dlgOrder.Dlg\_Type.CANCEL\_MKT\_DEPTH\_DLG**. This initializes the Cancel Market Depth dialog.
- Displays the Cancels Market Depth dialog.
- Calls the ActiveX method **cancelMktDepth()** when the **OK** button is clicked.

### The **cancelMktDepth()** Method

This method has a single parameter, *id*, which is the same ID that was specified in the **reqMktDepthEx()** call for market depth. The **cancelMktDepth()** method is shown below.

#### **cancelMktDepth()**

```
Sub cancelMktDepth(ByVal id As Integer)
```

Next we'll look at how the sample application handles another kind of data request, the historical data request.

## Chapter 9 - Requesting and Canceling Historical Data

This chapter describes how to request and cancel historical data in the sample application, and the API methods and events behind the process. For requesting historical data, you need to use the highlighted fields in the Request Historical Data dialog shown here:

**Request Historical Data**

Id: 10

**Contract Description**

Contract Id: 0  
 Symbol: IBM  
 Type: STK  
 Expiry:   
 Strike: 0  
 Right:   
 Multiplier:   
 Exchange: NYSE  
 Primary Exchange: NYSE  
 Currency: USD  
 Local Symbol:   
 Include Expired: 0  
 Sec Id Type:   
 Sec Id:

**Order Description**

Action: BUY  
 Quantity: 10  
 Order Type: LMT  
 Lmt/Opt Price / Volatility: 40  
 Aux/Under Price: 0  
 Good After Time:   
 Good Till Date:   
 FA Alloc  
 Combo Legs  
 Delta Neutral  
 Algo Params

**Market Data**

Generic Tick Tags: 100,101,104,10  
☐ Snapshot

**Market Depth**

Max Market Depth Rows: 10

**Exercise Options**

Action (1 or 2): 1  
 Quantity: 1  
 Override (0 or 1): 0

**Historical Data**

End Date/Time: 20110315 09:57:15  
 Query Duration: 1 M  
 Bar Size Setting: 1 day  
 What to Show: TRADES  
 Regular Trading Hours (1 or 0): 1  
 Date Format Style (1 or 2): 1

Ok Cancel

## What Happens When I Click the Historical Data Button?

Historical Data...

Once you connect to TWS using the ActiveX sample application, you request historical data by clicking the **Historical Data** button, then entering information in the *Ticker Description* and *Historical Data* fields in the Request Historical Data dialog and clicking **OK**. The historical data you request is displayed in the *Market and Historical Data* text panel.

That's a simple process from a user's point of view. But what's going on behind the scenes?

### Historical Data Button Event Handler

Just like all the other buttons in the sample application, the **Historical Data** button has an event handler associated with it. When you click the **Historical Data** button, the event handler **cmdReqHistoricalData\_Click**, defined in `dlgMainWnd`, runs. Here is what the code for the event handler looks like:

### Historical Data Button Event Handler

```
Private Sub cmdReqHistoricalData_Click(ByVal eventSender As _
System.Object, ByVal EventArgs As System.EventArgs) Handles _
cmdReqHistoricalData.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_HISTORICAL_DATA), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        If m_dlgOrder.whatToShow = "estimates" Or _
            m_dlgOrder.whatToShow = "finstat" Or _
            m_dlgOrder.whatToShow = "snapshot" Then
            Call Tws1.reqFundamentalData(m_dlgOrder.orderId, m_contractInfo, _
                m_dlgOrder.whatToShow)
        Else
            Call Tws1.reqHistoricalDataEx(m_dlgOrder.orderId, _
                m_contractInfo, m_dlgOrder.histEndDateTime, _
                m_dlgOrder.histDuration, m_dlgOrder.histBarSizeSetting, _
                m_dlgOrder.whatToShow, m_dlgOrder.userRTH, _
                m_dlgOrder.formatDate)
        End If
    End If
End Sub
```

**cmdReqHistoricalData\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.REQ_HISTORICAL_DATA_DLG`. This initializes the Request Historical Data dialog, yet another "version" of the `dlgOrder` object.
- Displays the Request Historical Data dialog.



- If ESTIMATES, FINSTAT or SNAPSHOT are entered into the *What To Show* field in the dialog, then the event handler calls the ActiveX method **reqFundamentalData()** when the **OK** button is clicked.
- If anything other than ESTIMATES, FINSTAT or SNAPSHOT is entered into the *What To Show* field, the event handler calls the ActiveX method **reqHistoricalDataEx()** when the **OK** button is clicked.



*Don't worry about the `reqFundamentalData()` method for now. It has to do with Reuters global fundamental data, and is outside the scope of our present discussion.*

### The **reqHistoricalDataEx()** Method

When you click **OK** in the sample application to request historical data, the **reqHistoricalDataEx()** method sends your request to TWS and, if all the entries are valid, the requested data is returned by way of the **historicalData()** event. So the **reqHistoricalDataEx()** method triggers this event in order to return historical data to the sample application.

Now let's see which parameters are used to request historical data. The **reqHistoricalDataEx()** method looks like this:

```
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)
```

As with the other ActiveX methods we've seen so far, the parameters used to request historical data correspond to the fields you complete in the Request Historical Data dialog. The parameters are:

Parameter	Description
<b>tickerId</b>	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.
<b>contract</b>	This object contains a description of the contract for which historical data is being requested. The contract description is defined by the fields you completed in the <i>Ticker Description</i> section of the Request Historical Data dialog. By the way, this is the same structure that is used to get market data and many other trading tasks! For a complete list of the properties in the contract structure, see the API Reference Guide.
<b>endDateTime</b>	This is the end data and time of the historical data request and corresponds to the field of the same name in the Historical Data section of the Request Historical Data dialog. Use the format <code>yyyymmdd hh:mm:ss tmz</code> , where the time zone is allowed (optionally) after a space at the end.

Parameter	Description
<b>durationStr</b>	This is the time span the request will cover, and is specified using the format: <integer> <unit>, i.e., 1 D, where valid units are S (seconds), D (days), W (weeks), M (months), and Y (years) . If no unit is specified, seconds are used. Also, note "years" is currently limited to one.
<b>barSize</b>	This parameter specifies the size of the bars that will be returned and corresponds to the field of the same name in the Request Historical Data dialog. For a complete list of the valid values for this parameter, see the <a href="#">reqHistoricalDataEx</a> topic in the <i>API Reference Guide</i> .
<b>whatToShow</b>	This specifies the type of data to show (trades, midpoints, bids, ask, bid/ask, option implied volatility and historical volatility) and corresponds to the field of the same name in the Request Historical Data dialog.
<b>useRTH</b>	This parameter determines whether to return all data available during the requested time span (value = 0), or only data that falls within regular trading hours (value = 1). It corresponds to the <i>Regular Trading Hours</i> field in the dialog.
<b>formatDate</b>	This is the end data and time of the historical data request and corresponds to the <i>Date Format</i> field in the Request Historical Data dialog. Valid values include 1, which returns dates applying to bars in the format: <code>yyyymmdd{space}{space}hh:mm:dd</code> , or 2, which returns dates as a long integer specifying the number of seconds since 1/1/1970 GMT.

### ActiveX Events that Return Historical Data

There is one ActiveX event that return historical data: **historicalData()**. This event is triggered by the **reqHistoricalDataEx()** method in `dlgMainWnd`. The **historicalData()** event, its parameters and the event handler in `dlgMainWnd` are shown below. You can see the data that is returned by looking at the the event's parameters in the tables.

#### historicalData()

```
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
<b>reqId</b>	The ticker ID of the request to which this bar is responding.
<b>date</b>	The date-time stamp of the start of the bar. The format is determined by the <code>reqHistoricalData()</code> <code>formatDate</code> parameter.
<b>open</b>	The bar opening price.
<b>high</b>	The high price during the time covered by the bar.
<b>low</b>	The low price during the time covered by the bar.
<b>close</b>	The bar closing price.

Parameter	Description
<b>volume</b>	The volume during the time covered by the bar.
<b>WAP</b>	The weighted average price during the time covered by the bar.
<b>hasGaps</b>	Identifies whether or not there are gaps in the data.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

### historicalData Event Handler

```
Private Sub Tws1_historicalData(ByVal eventSender As System.Object, ByVal _  
eventArgs As AxTWSLib._DTwsEvents_historicalDataEvent) Handles _  
Tws1.historicalData  
    Dim mktDataStr As String  
    mktDataStr = "id=" & eventArgs.reqId & " date=" & eventArgs.date & " open=" & _  
    & eventArgs.open & " high=" & eventArgs.high & _  
    & " low=" & eventArgs.low & " close=" & eventArgs.close & " volume=" & _  
    & eventArgs.volume & _  
    & " barCount=" & eventArgs.barCount & " WAP=" & eventArgs.wAP  
    If (eventArgs.hasGaps <> 0) Then  
        mktDataStr = mktDataStr & " has gaps"  
    Else  
        mktDataStr = mktDataStr & " no gaps"  
    End If  
    Call m_utils.addItem(Utils.List_Types.MKT_DATA, mktDataStr)  
  
    ' move into view  
    lstMktData.TopIndex = lstMktData.Items.Count - 1  
End Sub
```

## Canceling Historical Data

A rectangular button with a light beige background and a thin black border. The text "Cancel Hist. Data..." is centered on the button in a black, sans-serif font.

When you click the **Cancel Hist. Data** button, the Cancel Historical Data dialog appears. Yes, this is yet another version of the same dialog saw when we requested market data and market depth; the only difference is that when you cancel historical data, all the fields in the dialog are grayed out except *Ticker Id* and *Contract Id*. Simply click **OK** to submit historical data.

Now turn the page to see what happens in the code when you do this.

When you click the **Cancel Hist. Data** button, the **cmdCancelHistData\_Click** button event handler in **dlgMainWnd** runs.

### Cancel Hist. Data Button Event Handler

```
Private Sub cmdCancelHistData_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdCancelHistData.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.CANCEL_HIST_DATA_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        If m_dlgOrder.whatToShow = "estimates" Or _
            m_dlgOrder.whatToShow = "finstat" Or _
            m_dlgOrder.whatToShow = "snapshot" Then
            Call Tws1.cancelFundamentalData(m_dlgOrder.orderId)
        Else
            Call Tws1.cancelHistoricalData(m_dlgOrder.orderId)
        End If
    End If
End Sub
```

The **cmdCancelHistData\_Click** button event handler does the following:

- Sets the state of the **dlgOrder** object by setting **dlgOrder.Dlg\_Type.CANCEL\_HIST\_DATA\_DLG**. This initializes the Cancel Historical Data dialog.
- Displays the Cancel Historical Data dialog.
- If Reuters Fundamental Data was requested, calls the **cancelFundamentalData()** method when the **OK** button is clicked.
- Calls the ActiveX method **cancelHistoricalData()** when the **OK** button is clicked.

### The **cancelHistoricalData()** Method

This method has a single parameter, *tickerId*, which is the same Ticker ID that was specified in the **reqHistoricalDataEx()** call for historical data. The **cancelHistoricalData()** method is shown below.

#### **cancelHistoricalData()**

```
Sub cancelHistoricalData(ByVal tickerId As Integer)
```



*There are some limitations in the way TWS API handles historical data requests. Specifically, requesting the same historical data in a short period of time can cause extra load on the backend and cause pacing violations. For more information on these pacing violations, see [Historical Data Limitations](#) in the API Reference Guide.*

In the next section we'll look at how the sample application handles another kind of data request, the real-time bars request.

## Chapter 10 - Requesting and Canceling Real Time Bars

This chapter discusses the methods for requesting and canceling real time bars. 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. We show you the methods and parameters behind the Sample GUI, and how they call the methods in the TWS Java API.

For requesting real time bars, you need to use the fields circled in the Request Real Time Bars dialog shown below.

**Request Real Time Bars**

Id: 0

**Contract Description**

Contract Id: 0

Symbol: QQQQ

Type: STK

Expiry:

Strike: 0

Right:

Multiplier:

Exchange: SMART

Primary Exchange:

Currency: USD

Local Symbol:

Include Expired: 0

Sec Id Type:

Sec Id:

**Order Description**

Action: BUY

Quantity: 10

Order Type: LMT

Lmt/Opt Price / Volatility: 40

Aux/Under Price: 0

Good After Time:

Good Till Date:

FA Alloc

Combo Legs

Delta Neutral

Algo Params

**Market Data**

Generic Tick Tags: 100,101,104,10

☐ Snapshot

**Market Depth**

Max Market Depth Rows: 20

**Exercise Options**

Action (1 or 2): 1

Quantity: 1

Override (0 or 1): 0

**Historical Data**

End Date/Time: YYYYMMDD hh:mm:ss

Query Duration: 1 M

Bar Size Setting: 1 day

**What to Show**

What to Show: TRADES

Regular Trading Hours (1 or 0): 1

Date Format Style (1 or 2): 1

Ok

Cancel

## What Happens When I Click the Real Time Bars Button?

### Real Time Bars

You request real time bars by, well, clicking the **Real Time Bars** button, then entering information in the *Ticker Description* and a couple of the *Historical Data* fields in the Request Real Time Bars dialog and clicking **OK**. The data you request is displayed in the *Market and Historical Data* text panel.

Once again, this is a simple process from a user's point of view. Let's take a look at what's happening in the code when you request real time bars.

### Real Time Bars Button Event Handler

Just like all the other buttons in the sample application, the **Real Time Bars** button has an event handler associated with it. When you click the button, the event handler **cmdReqRealTimeBars\_Click**, defined in `dlgMainWnd`, runs. Here is what the code for the event handler looks like:

### Real Time Bars Button Event Handler

```
Private Sub cmdReqRealTimeBars_Click(ByVal sender As Object, ByVal e As _
    System.EventArgs) Handles cmdReqRealTimeBars.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_REAL_TIME_BARS_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.reqRealTimeBarsEx(m_dlgOrder.orderId, m_contractInfo, _
            5, m_dlgOrder.whatToShow, m_dlgOrder.useRTH)
    End If
End Sub
```

**cmdReqRealTimeBars\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.REQ_REAL_TIME_BARS_DLG`. This initializes the Request Real Time Bars dialog (yes, this is yet another version of the `dlgOrder` object).
- Displays the Request Real Time Bars dialog.
- Calls the ActiveX method **reqRealTimeBarsEx()** when the **OK** button is clicked.

## The reqRealTimeBarsEx() Method

When you click **OK** in the sample application to request real time bars, the **reqRealTimeBarsEx()** method sends your request to TWS and, if all the entries are valid, the requested data is returned by way of the **realTimeBar()** event. So the **reqRealTimeBarsEx()** method triggers this event in order to return real time bars to the sample application.

Now let's see which parameters are used to request real time bars. The **reqRealTimeBarsEx()** method looks like this:

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

As with the other ActiveX methods we've seen so far, the parameters used to request real time bars correspond to the fields you complete in the Request Real Time Bars dialog. The fields used to request real time bars are some of the same fields you used to request historical data.

Parameter	Description
<b>tickerId</b>	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.
<b>contract</b>	This object contains a description of the contract for which real time bars are being requested. The contract description is defined by the fields you completed in the <i>Ticker Description</i> section of the Request Real Time Bars dialog. And yes, this is the same structure that is used to get market data and many other trading tasks. For a complete list of the properties in the contract structure, see the API Reference Guide.
<b>barSize</b>	This parameter specifies the size of the bars that will be returned and corresponds to the field of the same name in the Historical Data section of the Request Real Time Bars dialog.
<b>whatToShow</b>	This specifies the type of data to show (trades, bid, ask, or midpoints), and corresponds to the field of the same name in the Historical Data section of the Request Real Time Bars dialog.
<b>useRTH</b>	This parameter determines whether to return all data available during the requested time span (value = 0), or only data that falls within regular trading hours (the value = 1). It corresponds to the Regular Trading Hours field in the Historical Data section of the Request Real Time Bars dialog.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

## ActiveX Events that Return Real Time Bars

There is one ActiveX event that return historical data: **realtimeBar()**. This event is triggered by the **reqRealTimeBarsEx()** method in `dlgMainWnd`.

The **realtimeBar()** event, its parameters and the event handler in `dlgMainWnd` are shown below. You can see the data that is returned by looking at the the event's parameters in the tables.

### realtimeBar()

```
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
<b>reqId</b>	The ticker Id of the request to which this bar is responding.
<b>time</b>	The date-time stamp of the start of the bar. The format is determined by the <code>reqHistoricalData()</code> <code>formatDate</code> parameter.
<b>open</b>	The bar opening price.
<b>high</b>	The high price during the time covered by the bar.
<b>low</b>	The low price during the time covered by the bar.
<b>close</b>	The bar closing price.
<b>volume</b>	The volume during the time covered by the bar.
<b>wap</b>	The weighted average price during the time covered by the bar.
<b>count</b>	When TRADES historical data is returned, represents the number of trades that occurred during the time period the bar covers.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

### realtimeBar() Event Handler

```
Private Sub Tws1_realtimeBar(ByVal eventSender As Object, ByVal eventArgs As _
    AxTWSLib._DTwsEvents_realtimeBarEvent) Handles Tws1.realtimeBar

    Dim mktDataStr As String
    mktDataStr = "id=" & eventArgs.tickerId & " time=" & eventArgs.time & _
        " open=" & eventArgs.open & " high=" & eventArgs.high & _
        " low=" & eventArgs.low & " close=" & eventArgs.close & " volume=" & _
        eventArgs.volume & " WAP=" & eventArgs.wAP & " count=" & eventArgs.count

    Call m_utils.addListItem(Utills.List_Types.MKT_DATA, mktDataStr)

    ' move into view
    lstMktData.TopIndex = lstMktData.Items.Count - 1
End Sub
```



## Canceling Real Time Bars

Disconnect

When you click the **Can Real Time Bars** button, the **cmdCancelRealTimeBars\_Click** event handler in **dlgMainWnd** runs.

### Cancel Real Time Bars Button Event Handler

```
Private Sub cmdCancelRealTimeBars_Click(ByVal eventSender As Object, ByVal _
    eventArgs As System.EventArgs) Handles cmdCancelRealTimeBars.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.CANCEL_REAL_TIME_BARS_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.cancelRealTimeBars(m_dlgOrder.orderId)
    End If
End Sub
```

The **cmdCancelRealTimeBars\_Click** button event handler does the following:

- Sets the state of the **dlgOrder** object by setting **dlgOrder.Dlg\_Type.CANCEL\_REAL\_TIME\_BARS\_DLG**. This initializes the Cancel Real Time Bars dialog.
- Displays the Cancel Real Time Bars dialog.
- Calls the ActiveX method **cancelRealTimeBars()** when the **OK** button is clicked.

### The **cancelRealTimeBars()** Method

This method has a single parameter, *tickerId*, which is the same Ticker ID that was specified in the **reqRealTimeBarsEx()** call for real time bars. The **cancelRealTimeBars()** method is shown below.

#### **cancelRealTimeBars()**

```
Sub cancelRealTimeBars(int tickerId)
```

As you can see from the similarity in the code for the trading tasks you've looked at so far, we've tried to make the ActiveX API methods and events as constant and easy to understand as possible.

The next chapter takes a look at how to run a market scanner using the ActiveX API sample application.

## Chapter 11 - Subscribing to and Canceling Market Scanner Subscriptions

This chapter describes the methods and events used for requesting market scanner parameters, subscribing to a market scanner, and canceling a subscription to a market scanner in the ActiveX sample application. When you click the **Market Scanner** button in the sample application, the Market Scanner dialog opens, instead of the standard Request/Cancel dialog we've seen for the other buttons on the sample application.

**Market Scanner**

Message Id  
Id: 0

Subscription Info

Number of Rows	10
Instrument	STK
Location Code	STK.US
Scan Code	TOP_PERC_GAIN
Above Price	3
Below Price	
Above Volume	0
Average Option Volume Above	0
Market Cap Above	100000000
Market Cap Below	
Moody Rating Above	
Moody Rating Below	
S and P Rating Above	
S and P Rating Below	
Maturity Date Above	
Maturity Date Below	
Coupon Rate Above	
Coupon Rate Below	
Exclude Convertibles	0
Scanner Setting Pairs	Annual,true
Stock Type Filter	

Request Parameters    Subscribe    Cancel Subscription

## What Happens When I Click the Market Scanner Button?

Market Scanner...

You can do two things related to market scanners in the ActiveX sample application:

- You can subscribe to a market scanner.
- You can request scanner parameters via an XML document, which is displayed in the sample application. This XML document describes the valid parameters that a scanner subscription can have.

To perform either of these tasks, you click the **Market Scanner** button, then enter information in the Market Scanner dialog and click the appropriate button. For a market scan, fill in as many of the fields in the dialog as you need (especially the *Scan Code*!), then click the **Subscribe** button. To request available scan parameters, click the **Request Parameters** button.

Market scan results are displayed in the *Market and Historical Data* text panel as shown below. Market scanner parameters (the XML file) are displayed in the *TWS Server Responses* text panel.



This is a fairly simple process from a user's point of view. There's more going on behind the scenes, however, so let's take a look.

## Market Scanner Button Event Handler

Just like all the other buttons in the sample application, the **Market Scanner** button has an event handler associated with it. When you click the button, the event handler **cmdScanner\_Click**, defined in `dlgMainWnd`, runs. Here is what the code for the event handler looks like:

### Market Scanner Button Event Handler

```
Private Sub cmdScanner_Click(ByVal sender As System.Object, ByVal e As _  
    System.EventArgs) Handles cmdScanner.Click  
    m_dlgScanner.ShowDialog()  
End Sub
```

**cmdScanner\_Click** really does only one thing: it displays the Market Scanner dialog, or as its known in the code, **dlgScanner**.

## Market Scanner Dialog

The process of filling in the fields of the Market Scanner dialog is pretty straightforward. However, the code for the Market Scanner dialog (**dlgScanner**), does a few things in which we're interested:

- Runs the **cmdRequestParameters\_click** event when you click the **Request Parameters** button.
- Runs the **cmdSubscribe\_click** event when you click the **Subscribe** button.
- Runs the **cmdCancelSubscription\_click** event when you click the **Cancel Subscription** button. (We'll learn more about how to cancel a market scanner subscription later in this chapter.)

The **cmdRequestParameters\_click** event calls the **reqScannerParameters()** method when you click the **Request Parameters** button.

The **cmdSubscribe\_click** event calls two methods:

- **populateSubscription()** - This is a private method that sets the values of the properties that make up the scanner subscription. These properties correspond to the fields in the Market Scanner dialog, of course. We'll see in a few minutes that these properties make up the **subscription** object, which is one of the parameters of the **scannerSubscriptionEx()** method.
- **scannerSubscriptionEx()** - This method tells TWS to start sending the scan data to the API sample application.

## Requesting Scanner Parameters

As we noted earlier, when you click the **Request Parameters** button in the Market Scanner dialog, the button event handler calls the **reqScannerParameters()** method, which requests an XML string that describes all possible market scans and their available parameters.

```
Sub reqScannerParameters()
```

As you can see, this method has no parameters. The XML document containing the scanner parameters is returned from TWS by the **scannerParameters()** event, which is shown below.

```
scannerParameters(ByVal xml As String)
```

The **scannerParameters()** event has a single parameter, *xml*, which is the XML document that contains the scanner parameters. This is the XML document that is displayed in the *TWS Server Responses* text panel of the sample application.

## Subscribing to a Market Scanner

When you click the **Subscribe** button in the Market Scanner dialog, the button event handler calls the **reqScannerSubscriptionEx()** method, which is shown below,

```
Sub reqScannerSubscriptionEx(ByVal tickerId As Integer, ByVal subscription  
As TWSLib.IScannerSubscription)
```

Parameter	Description
<b>tickerId</b>	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.
<b>subscription</b>	This object contains the scanner subscription parameters.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

As you can see from the table above, this method has two parameters. The *tickerId* parameter corresponds to the *Ticker ID* field in the dialog. The *subscription* object (*IScannerSubscription* TWS COM object) contains the properties that correspond to the fields in the Market Scanner dialog, such as *Instrument* and *Scan Code*. For a complete list of the properties in the *subscription* structure, see the [API Reference Guide](#).



*IScannerSubscription* is a TWS COM object that is created by the factory method **createScannerSubscription()**. You **MUST** use the **createScannerSubscription()** factory method to create the *IScannerSubscription* object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.

The market scan results are returned by the **scannerDataEx()** event:.

```
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)
```

You can see all the parameters for this event in the [scannerDataEx\(\)](#) topic in the *API Reference Guide*.

The event handler for **scannerDataEx()** is shown below:

#### scannerDataEx() Event Handler

```
Private Sub Tws1_scannerDataEx(ByVal eventSender As System.Object, ByVal _
eventArgs As AxTWSLib._DTwsEvents_scannerDataExEvent) Handles wsl.scannerDataEx
    Dim mktDataStr As String

    Dim contractDetails As TWSLib.IContractDetails
    contractDetails = eventArgs.contractDetails

    Dim contract As TWSLib.IContract
    contract = contractDetails.summary
    mktDataStr = "id=" & eventArgs.reqId & " rank=" & eventArgs.rank & _
        " conId=" & contract.conId & " symbol=" & contract.symbol & _
        " secType=" & contract.secType & " currency=" & contract.currency & _
        " localSymbol=" & contract.localSymbol & _
        " marketName=" & contractDetails.marketName & _
        " tradingClass=" & contractDetails.tradingClass & _
        " distance=" & eventArgs.distance & " benchmark=" & eventArgs.benchmark & _
        " projection=" & eventArgs.projection & " legsStr=" & eventArgs.legsStr
    Call m_utils.addListItem(Utils.List_Types.MKT_DATA, mktDataStr)

    ' move into view
    lstMktData.TopIndex = lstMktData.Items.Count - 1
End Sub
```

#### The scannerDataEnd() Event

There is one additional event used in conjunction with scanner subscriptions:  
**scannerDataEnd()**.

```
Sub scannerData(ByVal reqId As Integer
```

This event is called after a full snapshot of a scanner window has been received and functions as a sort of end tag. It helps define the end of one scanner snapshot and the beginning of the next. The **scannerDataEnd()** event handler is shown below.

#### scannerDataEnd() Event Handler

```
Private Sub Tws1_scannerDataEnd(ByVal sender As Object, ByVal eventArgs As _
AxTWSLib._DTwsEvents_scannerDataEndEvent) Handles Tws1.scannerDataEnd
    Dim str As String

    str = "id=" & eventArgs.reqId & " ===== end ====="
    Call m_utils.addItem(Utls.List_Types.MKT_DATA, str)

    ' move into view
    lstMktData.TopIndex = lstMktData.Items.Count - 1
End Sub
```

### Cancel a Market Scanner Subscription

To cancel a market scanner subscription in the ActiveX sample application, first click **Market Scanner** button, then click the **Cancel Subscription** button in the Market Scanner dialog. When you click this button, the **cmdCancelSubscription\_Click** event runs.

#### Cancel Subscription Button Event Handler

```
Private Sub cmdCancelSubscription_Click(ByVal eventSender As _
System.Object, ByVal eventArgs As System.EventArgs) Handles _
cmdCancelSubscription.Click
    Call populateSubscription()
    Call m_mainWnd.cancelScannerSubscription(m_id)
    Hide()
End Sub
```

The **cmdCancelSubscription\_Click** calls the **cancelScannerSubscription()** event, shown below, which cancels the market scanner subscription.

```
Sub cancelScannerSubscription(ByVal tickerId As Integer)
```

## Chapter 12: Requesting Contract Data

This chapter shows you how to request contract data, including details such as the local symbol, conid, trading class, valid order types, and exchanges. We'll walk you through everything that happens from the time you click the **Req Contract Data** button in the sample application, to the moment you're taking in the fascinating details of your desired contract. It all happens fast, so pay attention! To request contract data in the sample application, you need to fill in the fields in the Request Contract Details dialog shown below.

To request contract data using the ActiveX sample application, you'll need to enter data in the fields circled in the Request Contract Details dialog pictured below. The Request Contract Details dialog appears when you click the **Req Contract Data** button.

**Request Contract Details**

Id: 0

**Contract Description**

Contract Id: 0

Symbol: QQQQ

Type: STK

Expiry:

Strike: 0

Right:

Multiplier:

Exchange: SMART

Primary Exchange:

Currency: USD

Local Symbol:

Include Expired: 0

Sec Id Type:

Sec Id:

**Order Description**

Action: BUY

Quantity: 10

Order Type: LMT

Lmt/Opt Price / Volatility: 40

Aux/Under Price: 0

Good After Time:

Good Till Date:

FA Alloc

Combo Legs

Delta Neutral

Algo Params

**Market Data**

Generic Tick Tags: 100,101,104,10

☐ Snapshot

**Market Depth**

Max Market Depth Rows: 20

**Exercise Options**

Action (1 or 2): 1

Quantity: 1

Override (0 or 1): 0

**Historical Data**

End Date/Time: YYYYMMDD hh:mm:ss

Query Duration: 1 M

Bar Size Setting: 1 day

What to Show: TRADES

Regular Trading Hours (1 or 0): 1

Date Format Style (1 or 2): 1

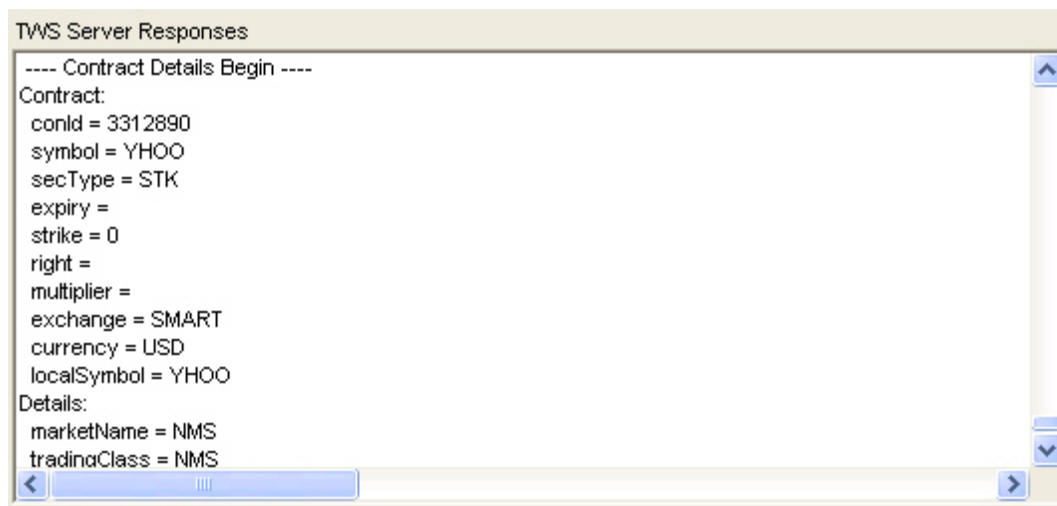
Ok Cancel



## What Happens When I Click the Req Contract Data Button?

Req Contract Data...

In the ActiveX sample application, you request contract details by clicking the **Req Contract Data** button, then entering an underlying and other information in the *Ticker Description* fields of the Request Contract Details dialog. When you click **OK**, the contract data you requested are displayed in the *TWS Server Responses* text panel, as shown here:



Let's take a look at the code behind this simple process.

### Req Contract Data Button Event Handler

When you click the **Req Contract Data** button, **cmdReqContractData\_Click**, the event handler in **dlgMainWnd** runs. Here is what that event handler looks like:

### Req Contract Data Button Event Handler

```
Private Sub cmdReqContractData_Click(ByVal eventSender As System.Object, _  
    ByVal eventArgs As System.EventArgs) Handles cmdReqContractData.Click  
    ' Set the dialog state  
    m_dlgOrder.init((dlgOrder.Dlg_Type.REQ_CONTRACT_DETAILS_DLG), _  
        m_contractInfo, m_orderInfo, m_underComp, Me)  
  
    m_dlgOrder.ShowDialog()  
    If m_dlgOrder.ok Then  
        Tws1.reqContractDetailsEx(m_dlgOrder.orderId, _  
            m_contractInfo)  
    End If  
End Sub
```

**cmdReqContractData\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.REQ_CONTRACT_DETAILS_DLG`. This initializes the Request Contract Details dialog. (This is another instance of our old friend, the `dlgOrder` object).
- Displays the Request Contract Details.
- Calls the ActiveX method `reqContractDetailsEx()` when the **OK** button is clicked.

### The `reqContractDetailsEx()` Method

Your contract data request is passed to TWS via the **`reqContractDetailsEx()`** method.

```
Sub reqContractDetailsEx(ByVal reqId As Integer, ByVal contract As
TWSLib.IContract)
```

This method contains two parameters, *reqId* and *contract*. *reqId* passes the ID of the data request to TWS and ensures that responses are matched to requests if several requests are in process. If you recall from earlier chapters, the *contract* parameter (*IContract* TWS COM object) contains all the attributes used to describe the requested contract, which in this case means all the values you entered in the fields in the *Ticker Description* section of the Request Contract Details dialog.



*IContract* is a TWS COM object that is created by the factory method **`createContract()`**. You **MUST** use the **`createContract()`** factory method to create the *IContract* object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.

### The `contractDetailsEx()` Event

The actual contract data is returned from TWS via the **`contractDetailsEx()`** event.

```
Sub contractDetailsEx(ByVal reqId As integer, ByVal contractDetails As
TWSLib.IContractDetails)
```

This event also contains two parameters, *reqId* and *contractDetails*. Just as in the method that requested the contract data, the *reqID* parameter here contains the ID of the data request to ensure matching with the correct request. The *contractDetails* parameter is a structure that contains all the attributes used to describe the requested contract, including the valid order types and exchanges on which the requested contract can be traded.

For a complete list of the properties in the *contractDetails* structure, see the [API Reference Guide](#).

### The **contractDetailsEnd()** Event

There is one additional contract data event used in the ActiveX API, the **contractDetailsEnd()** event. This event, which has a single parameter, *reqId*, 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)
```

The event handler for **contractDetailsEnd()** displays the end marker for the contract information displayed in the *TWS Server Responses* text panel of the sample application main window.

This concludes our discussion of market and contract data-related trading tasks. The next section describes how to place orders and exercise options using the sample application and ActiveX API.



# Orders and Executions

This section describes how the ActiveX API sample application handles orders. We'll show you the methods, events and parameters behind such trading tasks as placing and canceling orders, exercising options and viewing open orders and executions.

Here's what you'll find in this section:

- [Chapter 13 - Placing an Order](#)
- [Chapter 14 - Exercising Options](#)
- [Chapter 15 - Using Extended Order Attributes](#)
- [Chapter 16 - Requesting Open Orders](#)
- [Chapter 17 - Requesting Executions](#)

## Chapter 13: Placing and Canceling an Order

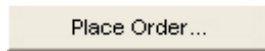
In this chapter, we describe what happens when you place and cancel an order. When you click the **Place Order** button, another version of the standard Request/Cancel dialog (the `dlgOrder` object) opens. To place an order, you fill in the fields circled in the dialog as shown below.

The screenshot shows the 'Place Order' dialog box with the following fields and sections:

- Contract Description (circled in red):**
  - Contract Id: 0
  - Symbol: QQQQ
  - Type: STK
  - Expiry:
  - Strike: 0
  - Right:
  - Multiplier:
  - Exchange: SMART
  - Primary Exchange:
  - Currency: USD
  - Local Symbol:
  - Include Expired: 0
  - Sec Id Type:
  - Sec Id:
- Order Description (circled in red):**
  - Action: BUY
  - Quantity: 100
  - Order Type: LMT
  - Lmt/Opt Price / Volatility: 54
  - Aux/Under Price: 0
  - Good After Time:
  - Good Till Date:
- Buttons:** FA Alloc, Combo Legs, Delta Neutral, Algo Params
- Market Data:**
  - Generic Tick Tags: 100,101,104,10
  - Snapshot: ☐
- Market Depth:**
  - Max Market Depth Rows: 20
- Exercise Options:**
  - Action (1 or 2): 1
  - Quantity: 1
  - Override (0 or 1): 0
- Historical Data:**
  - End Date/Time: YYYYMMDD hh:mm:ss
  - Query Duration: 1 M
  - Bar Size Setting: 1 day
  - What to Show: TRADES
  - Regular Trading Hours (1 or 0): 1
  - Date Format Style (1 or 2): 1
- Buttons:** Ok, Cancel

## What Happens When I Place an Order?

Let's take a look at what happens when you place an order. First we'll look at how a typical user would place an order using the ActiveX sample application, then we'll look at the code behind that process.



When you click the **Place Order** button, the Place Order dialog appears. As we mentioned earlier, this is yet another version of the `dlgOrder` object. You enter the contract information in the *Ticker Description* fields, enter the order information in the *Order Description* fields, then click the **OK** button to place the order. Your order information is displayed in the *TWS Server Responses* text panel on the sample application window.



*There are additional order options available in the ActiveX sample application that are supported by the API. You can enter Delta Neutral information, create a combo order or enter Algo parameters for IBAIgo orders. These options are described later in this chapter.*

That's pretty straightforward. Now let's see how the code works during this process.

### Place Order Button Event Handler

When you click the **Place Order** button, two things happen:

- The **cmdPlaceOrder\_Click** event handler runs, and calls the private method **placeOrderImpl()**.
- The **placeOrderImpl()** method runs.

**placeOrderImpl()**, shown on the next page, does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.PLACE_ORDER_DLG`. This initializes the Place Order dialog.
- Displays the Place Order dialog.
- Calls the ActiveX method **placeOrderEx()** when the **OK** button is clicked.

### placeOrderImpl() Method

```
Private Sub placeOrderImpl(ByVal whatIf As Boolean)

    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.PLACE_ORDER_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.orderId = m_nextOrderId
    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Dim savedWhatIf As Boolean
        savedWhatIf = m_orderInfo.whatIf()
        m_orderInfo.whatIf = whatIf
        Call Tws1.placeOrderEx(m_dlgOrder.orderId, m_contractInfo, _
            m_orderInfo)
        m_orderInfo.whatIf = savedWhatIf
    End If
End Sub
```

But wait, that's not all that happens! After you enter the contract and order information in the Place Order dialog and click the **OK** button, the information you entered into the *Ticker Description* fields are stored in the *m\_contractInfo* variable, and the information you entered into the *Order Description* fields are stored in the *m\_orderInfo* variable.

What are these variables? Well, in *dlgMainWnd*, these are set to the TWS COM objects *TWSLib.IContract* and *TWSLib.IOrder*, respectively. And these objects happen to be parameters of the **placeOrderEx()** method. So, all the contract and order information required to place your order are passed to TWS by **placeOrderEx()** in these two parameters.

### The placeOrderEx() Method

This is the method that passes your contract and order information to TWS.

```
Sub placeOrderEx(ByVal orderId As Integer, ByVal contract As
TWSLib.IContract, ByVal order As TWSLib.IOrder)
```

We mentioned above that the objects *TWSLib.IContract* and *TWSLib.IOrder* are actually parameters of **placeOrderEx()**. These objects pass your contract and order information to TWS. Here's the list of all of the method's parameters:

Parameter	Description
<b>orderId</b>	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.
<b>contract</b>	This object contains attributes used to describe the contract.
<b>order</b>	This object contains the details of the order.

This table is for illustrative purposes only and is not intended to portray valid API documentation.





*IContract is a TWS COM object that is created by the factory method **createContract()**. IOrder is a TWS COM object that is created by the factory method **createOrder()**. You MUST use the **createContract()** and **createOrder()** factory methods to create these COM objects. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.*

For a complete list of the properties in the *order* and *contract* structures, see the [API Reference Guide](#).

### The **orderStatus()** Event

Once the order has been placed, and assuming there are no errors in the order, the ActiveX event **orderStatus()** returns the current status of the order from TWS.

#### **orderStatus()** Event


```
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 Double, ByVal clientId As Integer, ByVal whyHeld As String)
```

The *status* parameter returns the status of your order and the sample application displays that information in the *TWS Server Responses* text panel. The possible values for the *status* parameter, and therefore, the possible order statuses, are:

- *PendingSubmit* - you have transmitted the order, but have not yet received confirmation that it has been accepted by the order destination.
  - *PendingCancel* - 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:** 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.
- *PreSubmitted* - 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.
  - *Submitted* - your order has been accepted at the order destination and is working.
  - *Cancelled* - the balance of your order has been confirmed canceled by the system. This could occur unexpectedly when the destination has rejected your order.
  - *Filled* - the order has been completely filled.
  - *Inactive* - the order has been accepted by the system (simulated orders) or an exchange (native orders) but that currently the order is inactive due to sytem, exchange or other issues.
  - *ApiPending* - the order has been reported to TWS by the API using reqAllOpenOrders() or reqOpenOrders().
  - *ApiCancelled* - the order reported by the API has been cancelled.

The rest of the details of your order are returned by the other parameters of the **orderStatus()** event. For complete descriptions of these parameters, see the [API Reference Guide](#).

## Canceling an Order



To cancel an order, click the **Cancel Order** button on the main sample window, then click the **OK** button in the Place Order dialog. When you cancel your order, make sure the *Ticker ID* is the same as the orderID for your order (you can find this in the order status display in the *TWS Server Responses* text panel).

Let's take a look at the code behind this operation.

When you click the **Cancel Order** button, the **cmdCancelOrder\_Click** button event handler in *dlgMainWnd* runs.

### Cancel Order Button Event Handler

```
Private Sub cmdCancelOrder_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdCancelOrder.Click
    ' Set the dialog state
    m_dlgOrder.init((dlgOrder.Dlg_Type.CANCEL_ORDER_DLG), _
        m_contractInfo, m_orderInfo, m_underComp, Me)

    m_dlgOrder.ShowDialog()
    If m_dlgOrder.ok Then
        Call Tws1.cancelOrder(m_dlgOrder.orderId)
    End If
End Sub
```

The *cmdCancelMktData* button event handler does the following:

- Sets the state of the *dlgOrder* object by setting *dlgOrder.Dlg\_Type.CANCEL\_ORDER\_DLG*. This initializes the Cancel Order dialog.
- Displays the Cancel Order dialog.
- Calls the ActiveX method **cancelOrder()** when the **OK** button in the Cancel Order dialog is clicked.

### The **cancelOrder()** Method

This method has a single parameter, *id*, which is the same order ID that was specified in the *placeOrderEx()* method. The **cancelOrder()** method is shown below.

```
Sub cancelOrder(ByVal id as Integer)
```

## Modifying an Order

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.

## Requesting "What-If" Data before You Place an Order



Another feature supported by the ActiveX sample application is the ability to request margin and commission "what if" data before you place an order. This means that you can click the **What If** button, set up your order as if you were actually placing it, then see what the margins and commissions would be if the trade went through.

The *IOrder* object, as you recall from earlier in this chapter, is one of the parameters in the **placeOrderEx()** method. Within the *IOrder* object, there is a property called *whatIf()*. When this value is set to true, the margin and commission data is received by the sample application and displayed in the *TWS Server Responses* text panel.

## Placing Combo Orders

The TWS ActiveX API supports combination orders, which means that you can use the ActiveX API sample application to place combo orders that include options, stock and futures legs (or you can build your own application to place combo orders using the methods, events and parameters in the API).

To place a combo order in the sample application, simply place an order as you normally would by clicking the **Place Order** button, fill in the fields in the Ticker Description and Order Description sections of the Place Order dialog, then click the **Combo Legs** button as shown below, and add combo legs to the order in the Combination Order Legs dialog.

The screenshot shows the 'Place Order' dialog box with the following fields and sections:

- Contract Description (Left Panel):** Contract Id (0), Symbol (QQQQ), Type (STK), Expiry, Strike (0), Right, Multiplier, Exchange (SMART), Primary Exchange, Currency (USD), Local Symbol, Include Expired (0), Sec Id Type, Sec Id.
- Order Description (Right Panel):** Action (BUY), Quantity (100), Order Type (LMT), Lmt/Opt Price / Volatility (54), Aux/Under Price (0), Good After Time, Good Till Date.
- Buttons:** FA Alloc, Combo Legs (highlighted with a red arrow), Delta Neutral, Algo Params.
- Market Data:** Generic Tick Tags (100,101,104,10), Snapshot checkbox.
- Market Depth:** Max Market Depth Rows (20).
- Exercise Options:** Action (1 or 2) (1), Quantity (1), Override (0 or 1) (0).
- Historical Data:** End Date/Time (YYYYMMDD hh:mm:ss), Query Duration (1 M), Bar Size Setting (1 day), What to Show (TRADES), Regular Trading Hours (1 or 0) (1), Date Format Style (1 or 2) (1).

The Combination Order Legs dialog appears.

Conid	Ratio	Side	Exchange	Open/Close	Short Sale Slot	Location
0	1	BUY	SMART	0	0	

To add combo legs, simply fill in the fields in the *Leg Details* section of the dialog, then click **Add**. Each leg you add appears in the list of combo legs on the left side of the dialog. You can remove unwanted legs by clicking the row to select it, then clicking the **Remove** button. When you click **OK**, you are returned to the Place Order dialog, where you click **OK** to place your order.

How is this process different in the code from the standard order placement? Let's find out!

## Combo Legs Processing

Well, as you might have expected, the first part of the process is the same as when you place a standard order: the **cmdPlaceOrder\_Click** event handler runs and does its thing, and the Place Order dialog is displayed, and your input values are stored as parameters of the **placeOrderEx()** method. But when you click the **Combo Legs** button, a different event handler, **cmdAddCmboLegs\_Click** runs and displays the Combination Order Legs dialog. The actual combo leg list is created by the code in the Combination Order Legs (dlgComboOrderLegs.vb), then passed to TWS as a property of the *IContract* COM object.

We can summarize the process this way:

- A container for the combo legs is declared and created. This container is the *IComboLegList* COM object. It is created by the **createComboLegList()** factory method.

In the button event handler for the **Combo Legs** button, this process can be seen in the If statement:

```
Private Sub cmdAddCmboLegs_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdAddCmboLegs.Click
    Dim dlgComboLegs As New dlgComboOrderLegs

    dlgComboLegs.Init(m_contractInfo.comboLegs, m_mainWnd.Tws1)
    dlgComboLegs.ShowDialog()
    If dlgComboLegs.ok Then
        Dim comboLegs As TWSLib.IComboLegList
        comboLegs = dlgComboLegs.comboLegs
        m_contractInfo.comboLegs = comboLegs
    End If
End Sub
```

- In the code for the Combination Order Legs dialog, each individual combo leg entered in the dialog is stored in another COM object, *IComboLeg*. As you click the **Add** button to add each leg to the list, the legs are added to the *IComboLegList* object via the *mComboLegs* property.
- The *mComboLegs* property is set to *comboLegs*, which is a property of the *IContract* COM object.
- The completed combo leg list, stored in the *comboLegs* property, is passed to TWS as part of the *IContract* object in the **placeOrderEx()** method.



*You MUST use the **createComboLegList()** factory method to create the *IComboLegList* object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.*

Just as in standard orders, once the order has been placed, and assuming there are no errors in the order, the ActiveX event **orderStatus()** returns the current status of the order from TWS.

## Combo Legs Code Example

To demonstrate how this works, here is a *greatly* simplified skeleton VB code example showing a typical Combo Order. In this example, the user is placing a two-legged combo order. One leg is for GOOG stock and the other leg is for a GOOG option.

```
Dim mComboLegs As IComboLegList
Set mComboLegs = mTws.createComboLegList()

'*****
'First Leg

Dim cmbLeg1 As IComboLeg
Set cmbLeg1 = mComboLegs.Add()

With cmbLeg1

    .conId = 30351181
    .ratio = 100
    .Action = "BUY"
    .exchange = "SMART"
    .openClose = 0
    .shortSalesSlot = 0
    .designatedLocation = ""

End With

'*****
'Second Leg

Dim cmbLeg2 As IComboLeg
Set cmbLeg2 = mComboLegs.Add()

With cmbLeg2

    .conId = 49316734
    .ratio = 1
    .Action = "SELL"
    .exchange = "SMART"
    .openClose = 0
    .shortSalesSlot = 0
    .designatedLocation = ""

End With

'*****
'contract description

Dim oContract As IContract
Set oContract = mTws.createContract

With oContract
```

```
.symbol = "GOOG"
.secType = "BAG"
.exchange = "SMART"
.currency = "USD"
.comboLegs = mComboLegs 'including combo description in contract object

End With

'*****
'order description

Dim oOrder As IOrder
Set oOrder = mTws.createOrder

With oOrder

.Action = "BUY"
.totalQuantity = 1
.orderType = "LMT"
.lmtPrice = 1.5

End With

'*****
'Place order to TWS

Call mTws.placeOrderEx(nextvalidid, oContract, oOrder)
```



## Placing Algo Orders

The TWS ActiveX API supports IBAIgo orders for US Equities and US Equity Options. Use IBAIgo orders to automatically balance market impact with risk on your large volume orders.

To place an IBAIgo order in the sample application, simply place an order as you normally would by clicking the **Place Order** button, fill in the fields in the *Ticker Description* and *Order Description* sections of the Place Order dialog, then click the **Algo Params** button as shown below and add an Algo strategy and Algo parameter/value pairs in the Algo Order Parameters dialog..

The screenshot shows the 'Place Order' dialog box with the following fields and values:

- Contract Description:**
  - Id: 0
  - Contract Id: 0
  - Symbol: QQQQ
  - Type: STK
  - Expiry:
  - Strike: 0
  - Right:
  - Multiplier:
  - Exchange: SMART
  - Primary Exchange:
  - Currency: USD
  - Local Symbol:
  - Include Expired: 0
  - Sec Id Type:
  - Sec Id:
- Order Description:**
  - Action: BUY
  - Quantity: 100
  - Order Type: LMT
  - Lmt/Dpt Price / Volatility: 54
  - Aux/Under Price: 0
  - Good After Time:
  - Good Till Date:
- Buttons:** FA Alloc, Combo Legs, Delta Neutral, Algo Params (highlighted with a red arrow)
- Market Data:**
  - Generic Tick Tags: 100,101,104,10
  - Snapshot: ☐
- Market Depth:**
  - Max Market Depth Rows: 20
- Exercise Options:**
  - Action (1 or 2): 1
  - Quantity: 1
  - Override (0 or 1): 0
- Historical Data:**
  - End Date/Time: YYYYMMDD hh:mm:ss
  - Query Duration: 1 M
  - Bar Size Setting: 1 day
  - What to Show: TRADES
  - Regular Trading Hours (1 or 0): 1
  - Date Format Style (1 or 2): 1

At the bottom are 'Ok' and 'Cancel' buttons.

The Algo Order Parameters dialog appears:

The screenshot shows the 'Algo Order Parameters' dialog box. It has a title bar with a close button. Inside, there's a section for 'Algorithm' with a 'Strategy' field containing 'minImpact'. Below that is a 'Parameters' section containing a table with two columns: 'Param' and 'Value'. The table has one row with 'maxPctVol' in the 'Param' column and '.1' in the 'Value' column. Below the table, there are two input fields: 'Param' with 'maxPctVol' and 'Value' with '.1'. There are 'Add' and 'Remove' buttons below these fields. At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Param	Value
maxPctVol	.1

To add Algo parameters to an order, type the name of the Algo strategy in the *Strategy* field, then type a parameter in the *Param* field, the value for the parameter in the *Value* field, and click **Add**. Repeat for each Algo parameter/value pair you want to add. For example, you might want to minimize market impact by slicing an options order over time as defined by the Max Percentage value. In this case you would type *minImpact* in the *Strategy* field, then type maxPctVol in the *Param* field and a percentage in the *Value* field. When you click **OK**, the Algo parameters are added to your order.

That's what happens on the user side of things; now let's take a look at the code.

## Algo Order Processing

As expected, the first part of the process is the same as when you place a standard order: the **cmdPlaceOrder\_Click** event handler runs and does its thing, and the Place Order dialog is displayed, and your input values are stored as parameters of the **placeOrderEx()** method. But when you click the **Algo Params** button, a different event handler, **cmdAlgoParams\_Click** runs and displays the Algo Order Parameters dialog (dlgAlgoParams.vb).

We can summarize the process this way:

- A container for the Algo parameters is declared and created. This container is the *ITagValueList* COM object. It is created by the **createTagValueList()** factory method.
- In the code for the Algo Order Parameters dialog, each individual parameter/value pair entered in the dialog is stored in another COM object, *ITagValue*. As you click the **Add** button to add each parameter/value pair to the list, the pairs are added to the *ITagValueList* object via the *m\_algoParams* property.
- In **cmdAlgoParams\_Click**, the Algo strategy and parameters are declared as properties of *m\_orderInfo*, which is set to the *IOrder* COM object in the Private Members section of the code in *VB\_API\_Sample.vb*. The *IOrder* object, if you recall, is responsible for sending the details of your order information to TWS.

In the button event handler for the **Algo Params** button, this process can be seen below:

```
Private Sub cmdAlgoParams_Click(ByVal sender As Object, ByVal e As _  
System.EventArgs) Handles cmdAlgoParams.Click  
    Dim dlg As New dlgAlgoParams  
  
    dlg.init(m_orderInfo.algoStrategy, m_orderInfo.algoParams, _  
        m_mainWnd.Tws1)  
    Dim res As DialogResult  
    res = dlg.ShowDialog()  
    If res = Windows.Forms.DialogResult.OK Then  
        m_orderInfo.algoStrategy = dlg.algoStrategy  
        m_orderInfo.algoParams = dlg.algoParams  
    End If  
End Sub
```



*You MUST use the **createTagValueList()** factory method to create the *ITagValueList* object, and you must use the **createOrder()** factory method to create the *IOrder* COM object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.*

Just as in standard orders, once the order has been placed, and assuming there are no errors in the order, the ActiveX event **orderStatus()** returns the current status of the order from TWS.

## Chapter 14: Exercising Options

This chapter discusses how the ActiveX sample application exercises options prior to expiration, and instructs options to lapse. We'll also show you the methods, events and parameters behind the Options Exercise area of the sample application. The fields you complete in the Exercise Options dialog (another instance of the `dlgOrder` object, by the way) are shown below.

The screenshot shows the 'ExerciseOptions' dialog box with the following fields and sections:

- Id:** 0
- Contract Description (highlighted):**
  - Contract Id: 0
  - Symbol: QQQQ
  - Type: STK
  - Expiry:
  - Strike: 0
  - Right:
  - Multiplier:
  - Exchange: SMART
  - Primary Exchange:
  - Currency: USD
  - Local Symbol:
  - Include Expired: 0
  - Sec Id Type:
  - Sec Id:
- Order Description:**
  - Action: BUY
  - Quantity: 10
  - Order Type: LMT
  - Lmt/Opt Price / Volatility: 40
  - Aux/Under Price: 0
  - Good After Time:
  - Good Till Date:
  - Buttons: FA Alloc, Combo Legs, Delta Neutral, Algo Params
- Market Data:**
  - Generic Tick Tags: 100,101,104,10
  - Snapshot: ☐
- Market Depth:**
  - Max Market Depth Rows: 20
- Exercise Options (highlighted):**
  - Action (1 or 2): 1
  - Quantity: 1
  - Override (0 or 1): 0
- Historical Data:**
  - End Date/Time: YYYYMMDD hh:mm:ss
  - Query Duration: 1 M
  - Bar Size Setting: 1 day
  - What to Show: TRADES
  - Regular Trading Hours (1 or 0): 1
  - Date Format Style (1 or 2): 1
- Buttons:** Ok, Cancel

## What Happens When I Click the Exercise Options Button?

Exercise Options...

When you click the **Exercise Options** button, the Exercise Options dialog appears. As we mentioned earlier, this is yet another version of the `dlgOrder` object. You enter the contract information in the *Ticker Description* fields, then enter the option exercise information in the fields in the *Option Exercise* section. In the *Action* field, enter *1* to exercise the option specified in the *Ticker Description* fields, or *2* to let the option expire. In the *Override* field, enter *1* to override the system's natural action, or *2* to not override. Click the **OK** button to execute your desired action (exercise or expire the option).

So what happens in the code when you do all this?

### Exercise Options Button Event Handler

When you click the **Exercise Options** button, the event handler **cmdExerciseOptions\_Click**, defined in `dlgMainWnd`, runs. Here is what the code for the event handler looks like:

#### Exercise Options Button Event Handler

```
Private Sub cmdExerciseOptions_Click(ByVal sender As System.Object, ByVal e _  
As System.EventArgs) Handles cmdExerciseOptions.Click  
    ' Set the dialog state  
    m_dlgOrder.init((dlgOrder.Dlg_Type.EXERCISE_OPTIONS), _  
        m_contractInfo, m_orderInfo, m_underComp, Me)  
  
    m_dlgOrder.ShowDialog()  
    If m_dlgOrder.ok Then  
        ' TODO: get account in a less convoluted way  
        Call Tws1.exerciseOptionsEx(m_dlgOrder.orderId, _  
            m_contractInfo, m_dlgOrder.exerciseAction, _  
            m_dlgOrder.exerciseQuantity, m_orderInfo.account, _  
            m_dlgOrder.exerciseOverride)  
    End If  
  
End Sub
```

**cmdExerciseOptions\_Click** does the following:

- Sets the state of the `dlgOrder` object by setting `dlgOrder.Dlg_Type.EXERCISE_OPTIONS_DLG`. This initializes the Exercise Options dialog, which is, as you might have expected, a version of the `dlgOrder` object.
- Displays the Exercise Options dialog.
- Calls the ActiveX method **exerciseOptionsEx()** when the **OK** button is clicked.

## The exerciseOptionsEx() Method

When you click **OK** in the Exercise Options dialog, the **exerciseOptionsEx()** method sends your request to TWS and, if all the entries are valid, your desired action is executed.

Now let's see which parameters are used when you exercise an option. The **exerciseOptionsEx()** method looks like this:

```
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)
```

Let's take a look at the parameters of this method.

Parameter	Description
<b>tickerId</b>	The Id for the exercise request.
<b>contract</b>	This structure contains a description of the contract for which market data is being requested.
<b>exerciseAction</b>	This represents the action you want to take on the specified option. A value of <i>1</i> indicates that you want to exercise the option. <i>2</i> means that you want to let the option expire.
<b>exerciseQuantity</b>	The number of contracts to be exercised.
<b>account</b>	This parameter specifies the IB account for institutional orders.
<b>override</b>	This parameter 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 <i>0</i> don't override or <i>1</i> to override.

This table is for illustrative purposes only and is not intended to portray valid API documentation.

As you can see from the table above, this method has several parameters that correspond to the fields in the Exercise Options dialog. This means that the values you entered in the dialog are passed to TWS by the parameters in the **exerciseOptionsEx()** method.

The *contract* parameter corresponds to the *Ticker Description* section of the dialog. *exerciseAction*, *exerciseQuantity*, and *override* correspond to fields in the *Options Exercise* section of the dialog. The *tickerId* parameter corresponds to the *Ticker ID* field in the dialog. The *contract* COM object (TWS API COM object *IContract*) used here is the same COM object that is used to get market data (and many other trading tasks!). For a complete list of the properties in the *contract* structure, see the [API Reference Guide](#).

In this case, there is no event that returns values from TWS.



*IContract* is a TWS COM object that is created by the factory method **createContract()**. You **MUST** use the **createContract()** factory method to create this COM objects. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.

## Chapter 15: Extended Order Attributes

This chapter discusses how to apply extended, or non-essential, order attributes to your order. This sample action is different from many of the others we've looked at, as the extended order attributes for the ActiveX API are actually included in the *IOrder* object, which you remember from our discussion on placing orders. For ease of use, the sample application has a separate dialog in which you can assign values to the extended order attributes. So although you will see a new dialog when you click the **Extended** button, the selections you're setting do not come from a new API method.

Here is the Extended Order Attributes dialog:

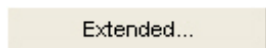


The dialog box, titled "Extended Order Attributes", contains a grid of 30 input fields for various order attributes. The fields are organized into two columns. The first column includes attributes like Time in Force, OCA group, OCA type, Account, Settling Firm, Clearing Account, Clearing Intent, Open/Close, Origin, Order Ref, Parent Id, Transmit, Block Order, Sweep to Fill, Display Size, Trigger Method, Outside RTH, Hidden, Discretionary Amt, Short Sales Slot, Designated Location, and Rule 80 A. The second column includes Trailing Stop Price, All or None, Minimum Quantity, Percent Offset, Electronic Exchange Only, Firm Quote Only, NBBO Price Cap, Override Percentage Constraints, BOX: Auction Strategy, BOX: Starting Price, BOX: Stock Ref Price, BOX: Delta, BOX/VOL: Stock Range Lower, BOX/VOL: Stock Range Upper, VOL: Volatility, VOL: Volatility Type (1 or 2), VOL: Hedge Delta Order Type, VOL: Hedge Delta Aux Price, VOL: Continuous Update, VOL: Reference Price Type (1 or 2), Scale: Init Level Size, Scale: Subs Level Size, and Scale: Price Increment. Each field has a text input box, some with pre-filled values (e.g., 0, 1). At the bottom are "Ok" and "Cancel" buttons.

Time in Force		Trailing Stop Price	
OCA group		All or None	0
OCA type	0	Minimum Quantity	
Account		Percent Offset	
Settling Firm		Electronic Exchange Only	1
Clearing Account		Firm Quote Only	1
Clearing Intent		NBBO Price Cap	
Open/Close	0	Override Percentage Constraints	0
Origin	0	BOX: Auction Strategy	0
Order Ref		BOX: Starting Price	
Parent Id	0	BOX: Stock Ref Price	
Transmit	1	BOX: Delta	
Block Order	0	BOX/VOL: Stock Range Lower	
Sweep to Fill	0	BOX/VOL: Stock Range Upper	
Display Size	0	VOL: Volatility	
Trigger Method	0	VOL: Volatility Type (1 or 2)	
Outside RTH	0	VOL: Hedge Delta Order Type	
Hidden	0	VOL: Hedge Delta Aux Price	
Discretionary Amt :	0	VOL: Continuous Update	0
Short Sales Slot (Institutional only)	0	VOL: Reference Price Type (1 or 2)	
Designated Location (Institutional only)		Scale: Init Level Size	
Rule 80 A		Scale: Subs Level Size	
		Scale: Price Increment	

Ok Cancel

## What Happens When I Click the Extended Button?



Extended order attributes have no function by themselves. However, any value you enter in the Extended Order Attributes dialog **WILL** be applied to every order you place in the sample application. You use these attributes to place advanced orders such as trailing stop limit, VOL and scale orders, as well modify other values for orders.



*For a complete description of all of the extended order attributes in the ActiveX API, see the [Extended Order Attributes](#) topic in the API Reference Guide.*

The code behind this process is fairly simple. When you click the **Extended** button, the event handler **cmdExtendedOrderAttribs\_Click** runs. This Click event simply displays the Extended Order Attributes dialog, called `dlgOrderAttribs`.

When you click the **OK** button in the Extended Order Attributes dialog, the values entered in the fields in the dialog are passed to the *IOrder* object, which stores all the information about your order.



*For a complete description of all of the properties in the *IOrder* object, see the [IOrder](#) topic in the API Reference Guide.*

That's all the **Extended** button does. Until you place an order, the extended attributes are just that - attributes just sitting there waiting for something to happen. But once you create and place an order, the values you entered/modified in the Extended Order Attributes dialog are used in your order, and will continue to be applied to every order until you change them.

Next we'll take a look at some of the other buttons in the sample application.



## Chapter 16: Requesting Open Orders

In this chapter, we're going to take a look at three related tasks in the ActiveX API sample application:

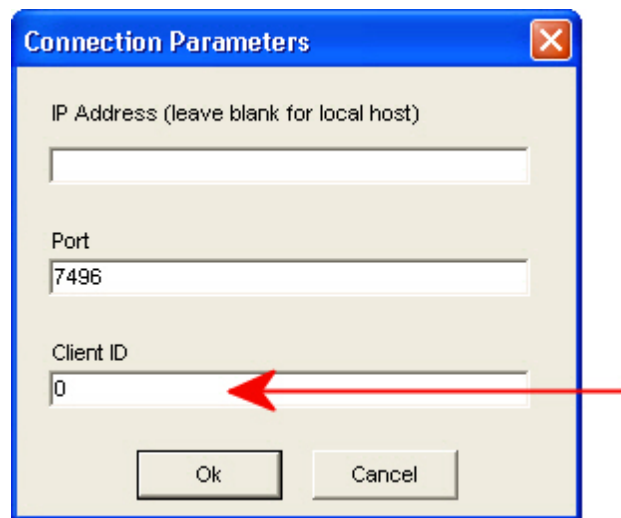
- Requesting Open Orders
- Requesting All Open Orders
- Requesting Auto Open Orders

How are they related?

Well, obviously they all give you information about open orders. The difference between them is the Client ID, which you set (or not!) when you connect to TWS.

### Running Multiple API Sessions

You can connect up to eight API sessions to one TWS client, but the catch is that you have to assign a new client ID for each API session. Therefore, any orders sent from these clients can be tracked through the life of the order, and everyone knows where they came from and who's responsible for them. So be careful!:



If you happen to have TWS up and running now and want to try this out, simply run multiple sample API sessions as described in the following steps:

- 1 Click the **Connect** button and connect to the first session. Note that the *Client ID* is set to "0."
- 2 Do the same for another session. If you don't change anything, you'll see that you are not able to connect to this second session. In the *Errors and Messages* text panel on the sample application, the API will kindly tell you "Already connected."
- 3 Now try it with a unique Client ID. Click **Connect** again, only this time type **1** (or any other unique Client ID) in the *Client ID* field, then click **OK**.

## The Difference between the Three Request Open Orders Buttons

Now you're ready to learn the difference between the three Request Open Orders methods/buttons:

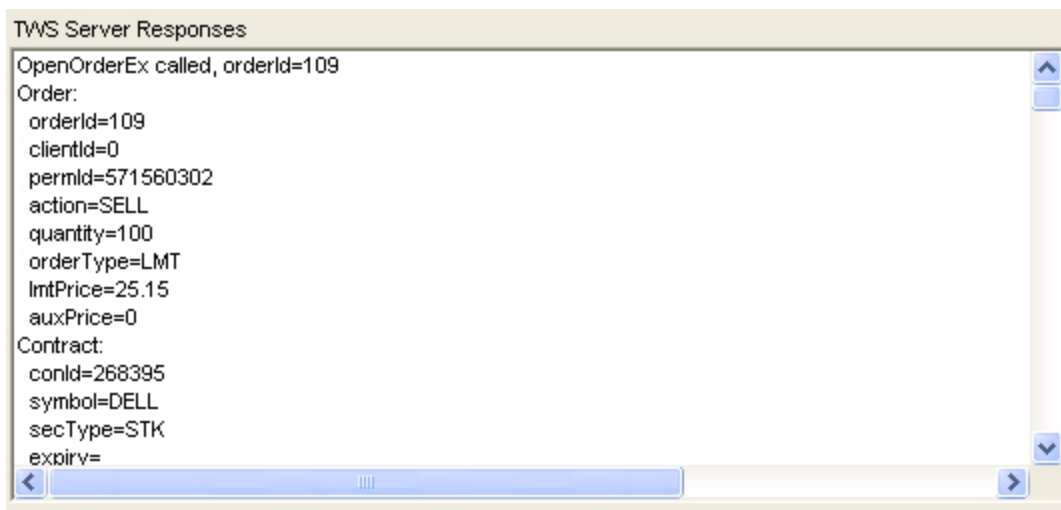
- **Request Open Orders** shows you any open orders made from that client, and if it's the "0" client ID client, you'll also see open orders sent from TWS.
- **Request All Open Orders** method shows you open orders sent from ALL clients connected to TWS, and all open orders that were sent from that TWS.
- **Request Auto Open Orders** method can only be used by the API with the client ID of "0." Clicking this button sets the boolean parameter to "True" and forever binds TWS orders to the API client. From that day forward, any time an open order exists on TWS it will automatically be returned via the Ewrapper methods, and in this case be displayed in the TWS Server Responses text panel of the sample application.

Got all that? Good, let's see the details.

## What Happens When I Click the Req Open Orders Button?

Req Open Orders

When you click the **Req Open Orders** button, any open orders that currently exist are displayed in the *TWS Server Responses* text panel of the main sample application window, as shown below.



If there are no open orders however, nothing will display in the panel.

Simple, right? So now let's see what happens in the code.

## Req Open Orders Button Event Handler

When you click the **Req Open Orders** button, the event handler **cmdReqOpenOrders\_Click** calls the ActiveX method **reqOpenOrders()**.

### Req Open Orders Button Event Handler

```
Private Sub cmdReqOpenOrders_Click(ByVal eventSender As System.Object,  
ByVal eventArgs As System.EventArgs) Handles cmdReqOpenOrders.Click  
    Call Tws1.reqOpenOrders()  
End Sub
```

The **reqOpenOrders()** method, shown below, has no parameters.

```
sub reqOpenOrders()
```

## The openOrderEx() Event

The **reqOpenOrders()** method triggers the **openOrderEx()** event, which returns information about open orders and displays it in the *TWS Server Responses* text panel. The **openOrderEx()** event is shown below.

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

If you've read this book from the beginning, by now you should be familiar with the *IContract* and *IOrder* COM objects, which contain properties that represent, respectively, a contract and an order. The **openOrderEx()** event receives open order information via these two structures, and also receives information from the *IOrderState* object, which contains properties representing the margin and commissions fields for both pre- and post-trade data.



*IContract* is a TWS COM object that is created by the factory method **createContract()**. *IOrder* is a TWS COM object that is created by the factory method **createOrder()**. You **MUST** use the **createContract()** and **createOrder()** factory methods to create these COM objects.

Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.

Unfortunately, the **openOrderEx()** event handler is very long, so we won't include it here. You can view it in the code for *VB\_API\_Sample.vb* in your IDE software.

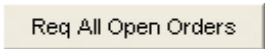
## The openOrderEnd() Event

There is one additional open order event used in the ActiveX API, the **openOrderEnd()** event. This event, which has no parameters, serves as an end marker for a set of received open orders. **openOrderEnd()** is called when all orders are sent to a client as a response to the `reqOpenOrders()` method.

```
sub openOrderEnd()
```

The event handler for **openOrderEnd()** displays the end marker for the open orders information displayed in the *TWS Server Responses* text panel of the sample application main window.

## What Happens When I Click the Req All Open Orders Button?



When you click the **Req All Open Orders** button, open orders sent from ALL clients connected to TWS, and all open orders that were sent from this client are displayed in the *TWS Server Responses* text panel of the main sample application window, as shown below.

Let's see what happens in the code.

### Req All Open Orders Button Event Handler

When you click the **Req All Open Orders** button, the event handler **cmdReqAllOpenOrders\_Click** calls the ActiveX method **reqAllOpenOrders()**.

#### Req All Open Orders Button Event Handler

```
Private Sub cmdReqAllOpenOrders_Click(ByVal eventSender As System.Object,  
ByVal eventArgs As System.EventArgs) Handles cmdReqAllOpenOrders.Click  
    Call Tws1.reqAllOpenOrders()  
End Sub
```

The **reqAllOpenOrders()** method, shown below, has no parameters.

```
sub reqAllOpenOrders()
```

The **reqAllOpenOrders()** method triggers the **openOrderEx()** event, which returns information about open orders and displays it in the *TWS Server Responses* text panel. See the section on [openOrderEx\(\)](#) earlier in this chapter for details about this event.

## What Happens When I Click the Req Auto Open Orders Button?

### Req Auto Open Orders

When you click the **Req All Open Orders** button, TWS orders are bound to the API client (the client you are running!). From that day forward, any time an open order exists on TWS it will automatically be returned via ActiveX **openOrderEx()** event, and displayed in the *TWS Server Responses* text panel of the sample application.

Note that this function can only be used by the API with the client ID of "0."

Let's see what happens in the code.

### Req Auto Open Orders Button Event Handler

When you click the **Req All Open Orders** button, the event handler **cmdReqAutoOpenOrders\_Click** calls the ActiveX method **reqAutoOpenOrders()**.

### Req Auto Open Orders Button Event Handler

```
Private Sub cmdReqAutoOpenOrders_Click(ByVal eventSender As System.Object,  
    ByVal eventArgs As System.EventArgs) Handles cmdReqAutoOpenOrders.Click  
    Call Tws1.reqAutoOpenOrders(True)  
End S
```

### The reqAutoOpenOrders() Method

The reqAutoOpenOrders() method, shown below, has no parameters.

```
sub reqAutoOpenOrders(ByVal bAutoBind As Integer)
```

This method has a single parameter, *bAutoBind*. If this parameter is set to *true* (and notice "(True)" in the call to the **reqAutoOpenOrders()** method above), newly created orders will be implicitly associated with the client making the Auto Open Orders request. If the parameter is set to *false*, no association is made.

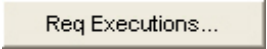
In other words, if you are using an API with a Client ID of "0," the *autoBind* parameter in **reqAutoOpenOrders** is set to true and orders from ALL clients connected to TWS will be reported to the sample application. If you're not Client ID 0, you'll receive an error message and the auto-binding won't be enabled.

The **reqAutoOpenOrders()** method triggers the **openOrderEx()** event, which returns information about open orders and displays it in the *TWS Server Responses* text panel. See the section on [openOrderEx\(\)](#) earlier in this chapter for details about this event.

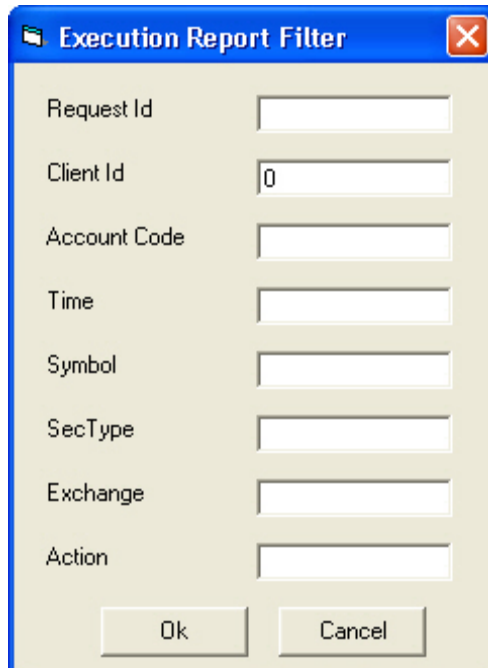
## Chapter 17: Requesting Executions

This chapter shows you how to request execution reports using the Execution Report Filter dialog in the ActiveX sample application. You can retrieve all execution reports, or only those you want by entering specific criteria such as time, symbol, exchange and more. We'll show you how to use the sample application to get these execution reports, and we'll see the methods, events and parameters behind the process.

### What Happens When I Click the Req Executions Button?

A rectangular button with a light beige background and a thin grey border. The text "Req Executions..." is centered on the button in a small, black, sans-serif font.

When you click the **Req Executions** button, the Execution Report Filter dialog appears. You enter filter criteria for your execution reports by filling in the fields. You can filter your execution reports by request ID, client ID, account code, time, symbol, security type, exchange or action. If you leave all the fields blank except the *Client ID* field (which is filled in with "0" by default), you will get reports of all of your executions. The report is displayed in the *TWS Server Responses* text panel.

A standard Windows-style dialog box titled "Execution Report Filter" in a blue header bar. The dialog has a close button (X) in the top right corner. It contains eight text input fields arranged vertically, each with a label to its left: "Request Id", "Client Id" (containing the value "0"), "Account Code", "Time", "Symbol", "SecType", "Exchange", and "Action". At the bottom of the dialog are two buttons: "Ok" and "Cancel".

Now let's see what happens in the code behind the scenes.

## Req Executions Button Event Handler

When you click the **Req Executions** button, the event handler **cmdReqExecutions\_Click**, defined in **dlgMainWnd**, runs. Here is what the code for the event handler looks like:

### Req Executions Button Event Handler

```
Private Sub cmdReqExecutions_Click(ByVal eventSender As System.Object, _  
    ByVal eventArgs As System.EventArgs) Handles cmdReqExecutions.Click  
    Dim dlgExecFilter As New dlgExecFilter  
  
    dlgExecFilter.init(m_execFilter)  
    dlgExecFilter.ShowDialog()  
    If dlgExecFilter.ok Then  
        Call Tws1.reqExecutionsEx(m_execFilter)  
    End If  
End Sub
```

**cmdExerciseOptions\_Click** does the following:

- Initializes and displays the Execution Filter dialog, **dlgExecFilter**.
- Calls the ActiveX method **reqExecutionsEx()**.

The Execution Filter dialog includes one important piece of code that passes the values you enter in the fields (*Client ID*, *Account Code*, *Time*, and so on), to the *IExecutionFilter* COM object.

We mentioned before that the *Client ID* field comes with a default value of "0." This isn't by chance! You can leave all of the other fields blank and everything will be fine. But if you leave the *Client ID* field blank, you'll get nothing, no matter what other field values you may enter. After you define the filter criteria and click **OK**, your values are passed to TWS via the **reqExecutionsEx()** method.

### The reqExecutionsEx() Method

The **reqExecutionsEx()** method sends the values you entered in the Execution Filter dialog to TWS. Another way of saying this is that the filter criteria you entered in the Execution Filter dialog are the parameters for this method, which is shown below.

```
Sub reqExecutionsEx(ByVal filter As TWSLib.IExecutionFilter, ByVal reqId  
    As Integer)
```



*IExecutionFilter* is a TWS COM object that is created by the factory method **createExecutionFilter()**. You **MUST** use the **createExecutionFilter()** factory method to create this COM object. Once created by a factory method, a COM object is tied to a corresponding TWS COM object. If you try to pass a COM object to another TWS COM object instance, you may get unpredictable results.

For a complete list of the properties in the *IExecutionFilter* COM object, see the [API Reference Guide](#).

## The `execDetails()` Method

Execution reports are returned via the **`execDetailsEx()`** event.

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

As you can see from the event, **`execDetailsEx()`** contains the following parameters:

Parameter	Description
<b>orderId</b>	The order Id that was specified previously in the call to <code>placeOrder()</code> .
<b>contract</b>	This structure contains a full description of the contract that was executed.
<b>execution</b>	This structure contains additional order execution details.
<b>reqId</b>	The ID of the data request. Ensures that responses are matched to requests if several requests are in process.

Tables are for illustrative purposes only and are not intended to represent valid API information.

The detailed information about your executions are included as properties of the *execution* structure. The *contract* structure contains information about the contract that was traded. For a complete list of the properties in the *execution* and *contract* structures, see the [API Reference Guide](#).

The event handler for **`execDetailsEx()`** is too long to include here, but you can look at it in Microsoft Visual Studio (or your favorite compatible IDE) to see the details.

## The `execDetailEnd()` Event

There is one additional event involved in getting execution reports from TWS: the **`execDetailsEnd()`** event. This event, which has a single parameter (*reqId*), serves as an end marker for a set of received execution reports. It is called when all executions have been sent to a client as a response to the **`reqExecutionsEx()`** method. Here is what the **`execDetailsEnd()`** event looks like:

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

The event handler for **`execDetailsEnd()`** displays the end marker for the execution reports displayed in the TWS Server Responses text panel of the sample application main window.

This concludes the section on orders and order information. The next section discusses the remaining tasks that you can perform using the ActiveX sample application (or using your own custom application!), including requesting the current server time and subscribing to news bulletins.



# Additional Tasks

This section describes some additional tasks that you can perform using the ActiveX API sample application. We'll show you the methods, events and parameters behind such tasks as requesting the current server time, the next ID, subscribing and unsubscribing to news bulletins, and changing the server logging level.

Here's what you'll find in this section:

- [Chapter 18 - Requesting the Current Time](#)
- [Chapter 19 - Requesting the Next ID](#)
- [Chapter 20 - Subscribing to News Bulletins](#)
- [Chapter 21 - Viewing and Changing the Server Logging Level](#)

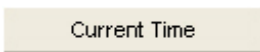


*In addition to the tasks described in this chapter, the ActiveX API sample application also includes a few more advanced functions, including the ability to calculate volatility and option price, and support for IBAlgos. For more information on these and other advanced capabilities of the ActiveX API, see our API Reference Guide, available from the Reference Guide tab on our IB API web page.*

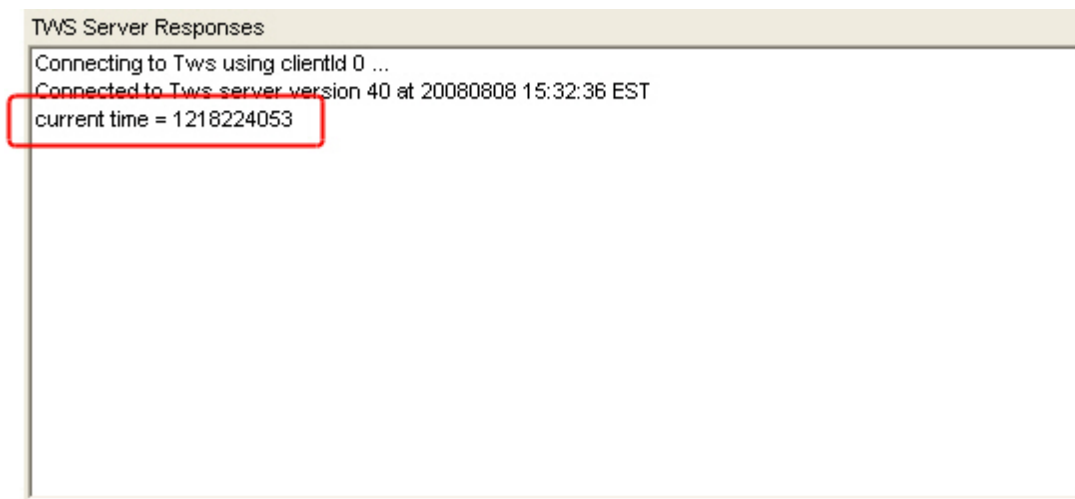
## Chapter 18 - Requesting the Current Time

This chapter discusses the method for requesting the current server time. Actually, "discusses" is really not the correct word. It merely "states" the method, which is quite solitary with no parameters to call its own.

### What Happens When I Click the Current Time Button?



You request the current server time by clicking the **Current Time** button. The server time is displayed in the *TWS Server Responses* text panel, as shown below.



This is a very simple process from a user's point of view and the code that makes this happen is also quite simple.

### Current Time Button Event Handler

Just like the other buttons in the sample application, the **Current Time** button has an event handler associated with it. When you click the button, the event handler **cmdReqCurrentTime\_Click**, defined in *dlgMainWnd*, runs. Here is what the code for the event handler looks like:

### Current Time Button Event Handler

```
Private Sub cmdReqCurrentTime_Click(ByVal sender As Object, ByVal e As _  
    System.EventArgs) Handles cmdReqCurrentTime.Click  
    Call Tws1.reqCurrentTime()  
End Sub
```

**cmdReqCurrentTime\_Click** calls the ActiveX method **reqCurrentTime()**, and that's pretty much all it does!

### The reqCurrentTime() Method

The **reqCurrentTime()** method is so simple that it doesn't even have any parameters. All it does is trigger the **currentTime()** event, which returns the current server time.

```
Sub reqCurrentTime()
```

### The currentTime() Event Handler

The **currentTime()** event has only one parameter, *time*, which as you might have guessed, returns the current time on the server.

The **currentTime()** event and event handler are shown below.

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

### Current Time Event

```
Private Sub Tws1_currentTime(ByVal sender As Object, ByVal EventArgs As  
AxTWSLib._DTwsEvents_currentTimeEvent) Handles Tws1.currentTime  
  
    Dim displayString As String  
    displayString = "current time = " & EventArgs.time  
  
    Call m_utils.addListItem(Utils.List_Types.SERVER_RESPONSES, displayString)  
  
    ' move into view  
    lstServerResponses.TopIndex = lstServerResponses.Items.Count - 1  
End Sub
```

## Chapter 19: Requesting the Next Order ID

Each order you place in TWS (and in an API application) has a unique order ID assigned to it. There is a rule about order IDs in TWS: Each successive order ID must be greater than the most recently used order ID. As an example, consider the situation in which you place an orders using order ID 1, then place an order using order ID 5. The next available order ID would be 6; you can never go back and use 2, 3 or 4. Order ID 6 is greater than order ID 5, the most recently used order ID.

You can use the TWS ActiveX API to request the next valid order ID that can be used when placing an order. You might use this functionality if you are creating your own custom trading application and wish to ensure that each order uses a legal order ID.

### The **reqIds()** Method

The ActiveX method that you use to request the next valid ID is **reqIds()**. After calling this method, the **nextValidId()** event is triggered, and the next valid ID is returned from TWS. That ID will reflect any autobinding that has occurred (which generates new IDs and increments the next valid ID therein).

The **reqIds()** method is shown below.

```
Sub reqIds(ByVal numIds As Integer)
```

**reqIds()** has a single parameter, *numIds*. This parameter however is simply a placeholder and has no real purpose. Simply set this parameter to any integer to make the method work the way it's supposed to.

### The **nextValidId()** Event

As we mentioned above, the next valid order ID is returned from TWS via the **nextValidId()** event. This event is also triggered when you successfully connect to TWS. The **nextValidId()** event looks like this:

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

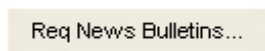
There is only one parameter returned with this event, *id*, which as you probably figured out by now returns the next available order ID received from TWS. Increment all successive orders by one based on this ID.

## Chapter 20: Subscribing to News Bulletins

This chapter shows you how to subscribe to IB news bulletins through the ActiveX sample application. Once you subscribe, all bulletins will display in the *TWS Server Responses* text panel of the sample application. The news bulletins keep you informed of important exchange disruptions.

We will show you the methods, events and parameters responsible for letting you subscribe and unsubscribe to news bulletin feature in the ActiveX sample application.

### What Happens When I Click the Req News Bulletins Button?



When you click the **Req News Bulletins** button, the IB News Bulletin Subscription dialog appears. In the dialog, you can elect to receive new messages only, or receive all the current day's messages and any new messages. These two options are presented as radio buttons. After you select your choice, click the **Subscribe** button to submit your subscription.



Once you subscribe to news bulletins, the news bulletins themselves will appear ???

That's how you subscribe to news bulletins using the ActiveX API sample application. Keep reading to learn what happens in the code during this process.

## Req News Bulletins Button Event Handler

When you click the **Req News Bulletins** button, the event handler **cmdReqNews\_Click**, defined in `dlgMainWnd`, runs. Here is what the code for the event handler looks like:

### Req News Bulletins Button Event Handler

```
Private Sub cmdReqNews_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdReqNews.Click
    m_dlgNewsBulletins.ShowDialog()
    If m_dlgNewsBulletins.ok Then
        If m_dlgNewsBulletins.subscribe = True Then
            Call Tws1.reqNewsBulletins(m_dlgNewsBulletins.allMsgs)
        Else
            Call Tws1.cancelNewsBulletins()
        End If
    End If
End Sub
```

**cmdReqNews\_Click** does the following:

- Initializes and displays the IB News Bulletin Subscription dialog, `dlgNewsBulletins`.
- Calls the ActiveX method **reqNewsBulletins()** if the user clicks the **Subscribe** button in the dialog.

### The reqNewsBulletins() method

This method tells TWS that you want to subscribe to news bulletins.

```
Sub reqNewsBulletins(ByVal allDaysMsgs As String)
```

**reqNewsBulletins()** has one parameter: *allDaysMsgs*. If you select the *receive new messages only* radio button in the IB News Bulletin Subscription dialog, the *allDaysMsgs* parameter, which asks "receive ALL messages, old and new?" will be set to false, which basically means that you will receive only new news bulletins. If you select *receive all the current day's messages and any new messages*, the *allDaysMsgs* parameter is set to true, which means, that you will receive all news bulletins for the current day PLUS any new news bulletins. Either way, you are now subscribed to news bulletins, and either way you will receive any NEW bulletins that get posted from the time you subscribe.

News bulletins are returned to the ActiveX sample application via the **updateNewsBulletin()** event.

## The updateNewsBulletin() Method

The bulletins are returned via the **updateNewsBulletin()** event.

```
Sub updateNewsBulletin(ByVal msgId As Short, ByVal msgType As Short,  
    ByVal message As String, ByVal origExchange As String)
```

**updateNewsBulletin()** contains the following parameters:

Parameter	Description
<b>msgId</b>	The bulletin ID, incrementing for each new bulletin.
<b>msgType</b>	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>
<b>message</b>	The bulletin's message text.
<b>origExchange</b>	The exchange from which this message originated.

Tables are for illustrative purposes only and are not intended to represent valid API information.

## Canceling News Bulletins

If you're tired of knowing what's going on around you, you can elect to unsubscribe, or cancel the news bulletins. To unsubscribe to news bulletin, you first need to click the **Req News Bulletins** button in the ActiveX sample application. Then you click **Unsubscribe** in the IB News Bulletin Subscription dialog, and we call the **cancelNewsBulletins()** method, which as the name implies, cancels your news bulletin subscription.

The **cancelNewsBulletins()** method header looks like this:

```
Sub cancelNewsBulletins()
```

Because you are simply canceling a request, there are no values returned by this method.

## Chapter 21: Viewing and Changing the Server Logging Level

This chapter shows you how to view and change the server logging level.

As client requests are processed (both system and API clients), TWS logs certain information to its log.txt log file 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. In the ActiveX sample application, you can specify how detailed the information will be when entered into the log.txt file. Basically, the higher the log level, the more performance overhead that may be incurred. By default, the server logging level is set to "2" for error logging.

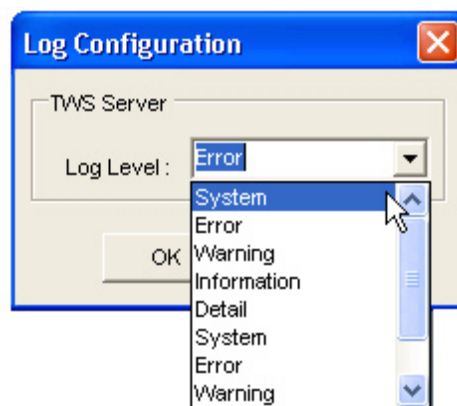


See our [API Reference Guide](#) for more information about API logging. The API Reference Guide is available from the **Application Programming Interfaces** page on our web site as an online guide or a downloadable/printable PDF.

### What Happens When I Click the Log Configuration Button?

Log Configuration...

To see or change the server logging level, you first click the **Log Configuration** button on the ActiveX sample application. In the Log Configuration dialog that appears, you select the logging level from the drop-down. You can select *System*, *Error*, *Warning*, *Information* or *Detail*. After you make your selection, click **OK** to close the dialog. Of course, you won't really see any changes in the sample application unless you encounter a problem of some kind.



That's what happens on the user side of things. Let's see what happens in the code.



## Log Configuration Button Event Handler

As with all the other buttons on the sample applications, when you click the **Log Configuration** button, an event handler in `dlgMainWind` runs. The event handler for this button is called `cmdServerLogLevel_Click` and is shown below.

## Log Configuration Button Event Handler

```
Private Sub cmdServerLogLevel_Click(ByVal eventSender As System.Object, _
    ByVal eventArgs As System.EventArgs) Handles cmdServerLogLevel.Click
    m_dlgLogConfig.ShowDialog()
    If m_dlgLogConfig.ok Then
        Call Tws1.setServerLogLevel(m_dlgLogConfig.serverLogLevel())
    End If
End Sub
```

`cmdServerLogLevel_Click` does the following:

- Initializes and displays the Log Configuration dialog (the `dlgLogConfig` object).
- Calls the ActiveX method `setServerLogLevel()` when the **OK** button is clicked.

The default level appears in the *Log Level* field of the Log Configuration dialog. We've expanded the dropdown list in the figure on the previous page just to show you the available log levels. Once you select a level and click **OK**, we call the `setServerLogLevel()` method.

## The setServerLogLevel() Method

The `setServerLogLevel()` method contains a single parameter, *logLevel*. This parameter passes the log level you selected in the Log Configuration dialog to TWS.

```
Sub setServerLogLevel(ByVal logLevel As Integer)
```

The *logLevel* parameter specifies the level of log entry detail used by TWS when processing API requests. The valid values for this parameter correspond to the choices in the Log Level dropdown in the Log Configuration dialog:

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



For more information about log levels and log entries, see the [API Logging](#) topic in the API Reference Guide.

There are no parameters passed from TWS to the API in this process; therefore, there is no corresponding event.

This concludes our discussion of the ActiveX sample application for individual accounts.



## **Additional Tasks**

*Chapter 21: Viewing and Changing the Server Logging Level*

# Where to Go from Here

If you've come this far and actually read the book, you now have a pretty decent grasp on what the ActiveX API can do, and how to make it do some of the things you want. Now we give you a bit more information about how to link to TWS with our ActiveX API, and we suggest some helpful outside resources you can use to help you move forward.

This section contains the following chapters:

- [Chapter 22 - Linking to TWS using the TWS ActiveX API](#)
- [Chapter 23 - Additional Resources](#)

## Chapter 22 - Linking to TWS using the TWS ActiveX API

If you have the skill and confidence to handle Visual Basic or VB.NET on your own, you can build your own ActiveX API application to link to TWS, using the following steps as a guide.

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

### To link using the ActiveX control

- 1** Drop the *Tws.ocx* ActiveX 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 `reqMktDataEx()` method to request market data.
- 3** Call the `placeOrderEx()` 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, you must first register it.

When you install our API software, the TWS ActiveX control, *Tws.ocx*, is automatically registered for you. If you install the Beta API software, the beta version of the TWS ActiveX control is registered and the production version of the ActiveX control is no longer registered. This is important to remember because if you try to run the production version of the ActiveX sample application after having installed the Beta API software, you will get errors unless you re-register the production version of the TWS ActiveX control!

### To register the ActiveX control, follow these instructions:

- 1** In Windows Explorer, navigate to the API software installation folder (typically C:\Jts), then open the *ActiveX* folder.
- 2** Right-click the file *Tws.ocx* file and select *Open With...*
- 3** Click the **Open With...** button to clear the Caution box, then select the *Select the program from a list* radio button and click **OK**.
- 4** Click the **Browse** button, then navigate to the Windows\System32 folder and select the file *regsvr32.exe*. Click **Open**.
- 5** Click **OK**.

A message appears to tell you that the *Tws.ocx* file was successfully registered.

## Chapter 23 - Additional Resources

There are many resources out there that will be adequate in getting you where you need to go. If you have some books or places that you like, feel free to stick with them. The following are the resources we find most helpful, and perhaps they'll be good to you, too!

### Help with Visual Basic and VB.NET Programming

While this book is intended for users with Visual Basic or VB.NET programming experience, we understand that even experienced programmers need help every once in a while.

The best place to go to find additional help with Visual Basic or Visual Studio is the Microsoft web site. Just type <http://msdn.microsoft.com/en-us/vbasic/default.aspx> in your browser's address line. This is the Visual Basic Developer Center, and from here you can access complete information about Visual Studio and Visual Basic.

There are literally hundreds of additional printed and web-based resources for Visual Basic programmers. We encourage you to investigate these on your own.

### Help with the TWS ActiveX API

For help specific to the TWS ActiveX API, the one best place to go, really the ONLY place to go, is the Interactive Brokers website. Once you get there, you have lots of resources. Just type [www.interactivebrokers.com](http://www.interactivebrokers.com) in your browser's address line. Now that you're there, let me tell you where you can go.



*As of this writing, the IB website looks as I'm describing. IB has a tendency to revamp the look and organization of their site every year or two, so have a little patience if it looks slightly different from what's described here.*

### The API Reference Guide

The API Reference Guide includes sections for each API technology, including the DDE for Excel. The upper level topics which are shown directly below the main book are applicable across the board to all or multiple platforms.

To access the API Reference Guide from the IB web site, select *API Solutions* from the **Trading** menu, then click the **IB API** button, then click the **Reference Guide** tab. Click the **Online API Reference Guide** button to open the online guide, which contains a section devoted entirely to the DDE for Excel API.

### The API Beta and API Production Release Notes

The beta notes are in a single page file, and include descriptions of any new additions to the API (all platforms) that haven't yet been pushed to production. The API Release Notes opens an index page that includes links to all of the past years' release notes pages. The index provides one-line titles of all the features included in each release.

To access these notes from the IB web site, select *API Solutions* from the **Trading** menu, then click the **IB API** button, then click the **Release Notes** tab and select a link to the latest API

production release notes. You can also access the release notes for the latest API Beta release from this page.

### **The TWS API Webinars**

IB hosts free online webinars through WebEx to help educate their customers and other traders about the IB offerings. They present the API webinar about once per month, and have it recorded on the website for anyone to listen to at any time.

- To register for the API webinar, from the IB web site click **Education**, then select *Webinars*. Click the **Live Webinars** button, then click the **API** tab.
- To view the recorded version of the API webinar, from the **Live Webinars** page click the **Watch Previously Recorded Webinars** button. Links to recorded versions of previously recorded webinars are listed on the page.

### **API Customer Forums**

You can trade ideas and send out pleas for help via the IB customer base accessible through both the IB Bulletin Board and the Traders' Chat. The bulletin board includes a thread for the API, and thus provides an ongoing transcript of questions and answers in which you might find the answer to your question. The Traders' Chat is an instant-message type of medium and doesn't retain any record of conversations.

- "To view or participate in the IB Bulletin Board, go to the **Education** menu and click *Bulletin Boards & Chats*. Click the **Bulletin Board** tab, then click the **Launch IB Discussion Forum button** to access all of our bulletin boards, including the TWS API bulletin board.
- To participate in the Traders' Chat, you need to click the **Chat** icon from the menu bar on TWS. Note that both of these customer forums are for IB customers only.

### **IB Customer Service**

IB customers can also call or email customer service if you can't find the answer to your question. However, IB makes it clear that the APIs are designed for use by programmers and that their support in this area is limited. Still, the customer service crew is very knowledgeable and will do their best to help resolve your issue. Simply send an email to:

**api@interactivebrokers.com**

### **IB Features Poll**

The IB Features Poll lets IB customers submit suggestions for future product features, and vote and comment on existing suggestions.

From the IB web site, click **About IB**, then select *New Features Poll*. Suggestions are listed by category; click a plus sign next to a category to view all feature suggestions for that category. To submit a suggestion, click the *Submit Suggestion* link.





# Appendix A - Extended Order Attributes

Attribute	Valid Values
timeInForce	DAY GTC OPG IOC
ocaGroup	String
account	The account number, used for institutional and advisor accounts.
open/close	O, C (for institutions)
origin	0, 1 (for institutions)
orderRef	String
transmit	0 (don't transmit) 1 (transmit)
Parent order Id	String (the order ID used for the parent order, use for bracket and auto trailing stop orders)
blockOrder	0 (not a block order) 1 (this is a block order)
sweepToFill	0 (not a sweep-to-fill order) 1 (this is a sweep-to-fill order)
displaySize	String (publicly disclosed order size)

Attribute	Valid Values
triggerMethod	<p>Specifies how simulated Stop, Stop-Limit, and Trailing Stop orders are triggered.</p> <p><b>0</b> - 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.</p> <p><b>1</b> - use "double bid/ask" method, where stop orders are triggered based on two consecutive bid or ask prices.</p> <p><b>2</b> - "last" method, where stop orders are triggered based on the last price.</p> <p><b>3</b> - "double-last" method, where stop orders are triggered based on last two prices.</p> <p><b>4</b> - "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.</p> <p><b>7</b> - "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.</p> <p><b>8</b> - "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.</p>
Hidden	<p>Only valid for orders routed to Island.</p> <p>0 - False</p> <p>1 (order not visible when viewing market depth)</p>
Discretionary Amount	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
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 Account Group.
FA Method	For Advisor accounts only. The share allocation method. EqualQuantity NetLiq AvailableEquity PctChange
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 institutional accounts only; for SSHORT actions. 1 - If you hold the shares 2 - Shares will be delivered from elsewhere.
Short Sale Location	If shares are delivered from elsewhere, enter where in a comma-delimited list with no spaces. For institutional accounts only.

Attribute	Valid Values
OCA Type	1 = Cancel on Fill with Block 2 = Reduce on Fill with Block 3 = Reduce on Fill without Block
Rule 80A	Individual = 'I' Agency = 'A', AgentOtherMember = 'W' IndividualPTIA = 'J' AgencyPTIA = 'U' AgentOtherMemberPTIA = 'M' IndividualPT = 'K' AgencyPT = 'Y' AgentOtherMemberPT = 'N'
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	For BOX exchange only. match = 1 improvement = 2 transparent = 3
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.
Stock Range Lower	The lower value for the acceptable underlying stock price range. For price improvement option orders on BOX and VOL orders with dynamic management.
Stock Range Upper	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.

Attribute	Valid Values
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.
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 TRAILLIMIT orders.
Scale Num Components	Used for Scale orders only, this value defines the number of components in the order.
Scale Component Size	NO LONGER SUPPORTED
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

## Appendix B - Account Page Values

Field	Description	Notes
Account Code	The account number.	
Account Type	Identifies the IB account type.	
Accrued Cash	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.
Available Funds	<b>For securities:</b> Equity with Loan Value - Initial margin <b>For commodities:</b> Net Liquidation Value - Initial margin	
Buying Power	Cash Account : (Minimum (Equity with Loan Value, Previous Day Equity with Loan Value)- Initial Margin) Standard Margin Account : Available Funds*4	
Cash Balance	<b>For securities:</b> Settled cash + sales at the time of trade <b>For commodities:</b> Settled cash + sales at the time of trade + futures PNL	
Currency	Shows the currency types that are listed in the Market Value area.	
Cushion	Shows your current margin cushion.	
Day Trades Remaining	Number of day trades left for pattern day trader period.	
Day Trades Remaining T+1, T+2, T+3, T+4	The number of day trades you have left for a 4-day pattern day-trader.	

Field	Description	Notes
Equity With Loan Value	<p>For Securities:</p> <ul style="list-style-type: none"> <li>Cash Account: Settled Cash</li> <li>Margin Account:</li> <li>Total cash value + stock value + bond value + (non-U.S. &amp; Canada securities options value)</li> </ul> <p>For Commodities:</p> <ul style="list-style-type: none"> <li>Cash Account: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures PNL)</li> <li>Margin Account: Total cash value + commodities option value - futures maintenance margin requirement</li> </ul>	
Excess Liquidity	Equity with Loan Value - Maintenance margin	
Exchange Rate	The exchange rate of the currency to your base currency.	
Full Available Funds	<p>For securities:</p> <p>Equity with Loan Value - Initial margin</p> <p>For commodities:</p> <p>Net Liquidation Value - Initial margin</p>	
Full Excess Liquidity	Equity with Loan Value - Maintenance margin	
Full Init Margin Req	Overnight initial margin requirement in the base currency of the account.	
Full Maint Margin Req	Maintenance margin requirement as of next period's margin change in the base currency of the account.	
Future Option Value	Real-time mark-to-market value of futures options.	
Futures PNL	Real-time change in futures value since last settlement.	
Gross Position Value	Long Stock Value + Short Stock Value + Long Option Value + Short Option Value.	
Init Margin Req	Initial margin requirement in the base currency of the account.	

Field	Description	Notes
Leverage	For Securities: <ul style="list-style-type: none"> <li>Gross Position value / Net Liquidation value</li> </ul> For Commodities: <ul style="list-style-type: none"> <li>Net Liquidation value - Initial margin</li> </ul>	
Look Ahead Available Funds	For Securities: <ul style="list-style-type: none"> <li>Equity with loan value - look ahead initial margin.</li> </ul> For Commodities: <ul style="list-style-type: none"> <li>Net Liquidation value - look ahead initial margin.</li> </ul>	
Look Ahead Excess Liquidity	Equity with loan value - look ahead maintenance margin.	
Look Ahead Init Margin Req	Initial margin requirement as of next period's margin change in the base currency of the account.	
Look Ahead Maint Margin Req	Maintenance margin requirement as of next period's margin change in the base currency of the account.	
Look Ahead Next Change	Indicates when the next margin period begins.	
Maint Margin Req	Maintenance margin requirement in the base currency of the account.	
Net Liquidation	For Securities: <ul style="list-style-type: none"> <li>Total cash value + stock value + securities options value + bond value</li> </ul> For Commodities: <ul style="list-style-type: none"> <li>Total cash value + commodities options value</li> </ul>	
Net Liquidation by Currency	Same as above for individual currencies.	
Option Market Value	Real-time mark-to-market value of securities options.	
PNL	The difference between the current market value of your open positions and the average cost, or Value - Average Cost.	
Previous Day Equity with Loan Value	Marginable Equity with Loan Value as of 16:00 ET the previous day, only applicable to securities.	

Field	Description	Notes
Realized PnL	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).	
Reg T Equity	Initial margin requirements calculated under US Regulation T rules.	
Reg T Margin	<p>For Securities:</p> <ul style="list-style-type: none"> <li>Cash Account: Settled Cash</li> <li>Margin Account: Total cash value + stock value + bond value + (non-U.S. &amp; Canada securities options value)</li> </ul> <p>For Commodities:</p> <ul style="list-style-type: none"> <li>Cash Account: Total cash value + commodities option value - futures maintenance margin requirement + minimum (0, futures PNL)</li> <li>Margin Account: Total cash value - futures maintenance margin requirement</li> </ul>	
SMA	<p>Max ((EWL - US initial margin requirements)*, (Prior Day SMA +/- change in day's cash +/- US initial margin requirements**) for trades made during the day.))</p> <p>*calculated end of day under US Stock rules, regardless of country of trading.</p> <p>**at the time of the trade</p>	Only applicable for securities.
Stock Market Value	Real-time mark-to-market value of stock	
Total Cash Balance	Cash recognized at the time of trade + futures PNL	
Total Cash Value	Total cash value of stock, commodities and securities	



# Index

## A

- account page values B-123
- Active X API
  - preparing to use 2-21
- ActiveX
  - registering third-party controls 6-115
- ActiveX API
  - additional resources 6-116
  - installing an IDE 2-22
- ActiveX API sample application
  - connecting to 2-28
- ActiveX API, help with 6-116
- ActiveX controls
  - registering 2-27, 6-115
- ActiveX sample application
  - framework 3-34
- additional resources 6-116
- additional trading tasks 5-103
- Algo Order Parameters 4-88
- Algo order processing 4-89
- Algo orders 4-87
- Algo Params button event handler 4-89
- API
  - reasons for using 1-18
- API beta notes 6-116
- API Reference Guide 6-116
- API release notes 6-116
- API software
  - downloading 2-24
  - installing 2-27
- API support email 6-117
- API technologies 1-19
- API webinars 6-117

## C

- Canc Real Time Bars button event handler 3-63
- Cancel Hist. Data button event handler 3-58
- Cancel Mkt Data button 3-45, 3-52, 3-57
- Cancel Mkt Depth button event handler 3-52
- Cancel Subscription button event handler 3-69
- cancelHistoricalData() 3-58
- canceling a market scanner subscription 3-69
- canceling historical data 3-53, 3-57, 3-58
- canceling market data 3-39, 3-45, 3-46
- canceling market depth 3-47, 3-52
- canceling market scanner subscriptions 3-64
- canceling news bulletins 5-109
- canceling orders 4-76, 4-80
- canceling real time bars 3-59, 3-63
- cancelMktData() 3-46
- cancelMktDepth() 3-52
- cancelNewsBulletins() 5-109

- cancelOrder() 4-80
- cancelRealTimeBars() 3-63
- changing the server logging level 5-110
- Client ID and multiple API sessions 4-95
- cmdAddCmboLegs\_Click 4-84
- cmdAlgoParams\_Click 4-89
- cmdCancelHistData\_Click 3-58
- cmdCancelMktData\_Click 3-45
- cmdCancelMktDepth\_Click 3-52
- cmdCancelOrder\_Click 4-80
- cmdCancelRealTimeBars\_Click 3-63
- cmdCancelSubscription\_Click 3-69
- cmdConnect\_Click 3-37
- cmdDisconnect\_Click 3-38
- cmdExerciseOptions\_Click 4-91
- cmdExtendedOrderAttribs\_Click 4-94
- cmdReqAllOpenOrders\_Click 4-98
- cmdReqAutoOpenOrders\_Click 4-99
- cmdReqContractData\_Click 3-71
- cmdReqCurrentTime\_Click 5-104
- cmdReqExecutions\_Click 4-101
- cmdReqHistoricalData\_Click 3-54
- cmdReqMktData\_Click 3-40, 3-41
- cmdReqMktDepth\_Click 3-48
- cmdReqNews\_Click 5-108
- cmdReqOpenOrders\_Click 4-97
- cmdReqRealTimeBars\_Click 3-60
- cmdRequestParameters\_click 3-66
- cmdScanner\_Click 3-66
- cmdServerLogLevel\_Click 5-111
- cmdSubscribe\_click 3-66
- Combination Order Legs dialog 4-83
- combo legs code sample 4-85
- combo orders 4-82, 4-83, 4-84, 4-85
  - combo legs processing 4-84
- Connect button 3-35
- Connect button event handler 3-37
- connecting the sample application to TWS 3-34
- connecting to the sample application 2-28
- connecting to TWS 3-34, 3-35, 3-37
- Connection Parameters dialog 3-35
- connectionClosed() 3-38
- connectionClosed() Event Handler 3-38
- contract data 3-70, 3-71, 3-72
- contract details 3-70
- contract details results 3-71
- contract object 3-43, 3-49
- contractDetailsEnd() 3-73
- contractDetailsEx() 3-72
- createComboLegList() 4-84
- createContract() 3-44, 3-49, 3-72, 4-92
- createExecutionFilter() 4-101
- createOrder() 4-97

createScannerSubscription() 3-67  
 createTagValueList() 4-89  
 current time 5-104, 5-105  
 Current Time button event handler 5-104  
 current time results 5-104  
 currentTime() 5-105  
 customer forums 6-117  
 customer service 6-117

## D

Disconnect button 3-38  
 disconnect() 3-38  
 disconnecting from TWS 3-38  
 dlgMainWnd 3-35  
 dlgOrder object  
   states of 3-41  
 document conventions -11  
 downloading API software 2-24

## E

events  
   historical data 3-56  
   market data 3-44  
   market depth 3-50  
   real time bars 3-62  
 execDetailEnd() 4-102  
 execDetails() 4-102  
 execDetails() parameters 4-102  
 Execution Report Filter dialog 4-100  
 executions 4-75, 4-100  
 Exercise Options button 4-91  
 Exercise Options button event handler 4-91  
 Exercise Options dialog 4-90  
 exerciseOptionsEx() 4-92  
 exerciseOptionsEx() parameters 4-92  
 exercising options 4-90  
 Extended button 4-94  
 extended order attributes 4-93, 4-94  
 Extended Order Attributes dialog 4-93  
 extended order attributes A-119

## F

Features Poll 6-117  
 footnotes and references -9  
 framework of sample application 3-34

## H

historical data 3-53, 3-54, 3-55, 3-56, 3-57  
 Historical Data button 3-54  
 Historical Data button event handler 3-54  
 historical data events 3-56  
 historicalData event handler 3-57  
 historicalData() 3-56  
 historicalData() parameters 3-56  
 how to use this book -8

## I

IB bulletin boards 6-117

IB Customer Service 6-117  
 IBAIgo orders 4-87  
 IComboLeg 4-84  
 IComboLegList 4-84  
 icons used in this book -10  
 IDE, installing 2-22  
 installing API software 2-27  
 integrated development environment 2-22  
 introduction -7  
 ITagValue 4-89  
 ITagValueList 4-89

## L

Log Configuration button event handler 5-111  
 Log Configuration dialog 5-110, 5-111  
 log.txt file 5-110  
 logLevel 5-111

## M

m\_orderInfo 4-89  
 market data 3-33, 3-39, 3-40, 3-41, 3-42, 3-44  
   canceling 3-45  
   snapshot 3-45  
 market data events 3-44  
 market data results 3-40  
 market depth 3-47, 3-48, 3-49, 3-50, 3-51  
 market depth events 3-50  
 Market Depth for dialog 3-48  
 market scan results 3-65  
 market scanner  
   subscribing to 3-67  
 Market Scanner button 3-65  
 Market Scanner button event handler 3-66  
 Market Scanner dialog 3-64  
   code in 3-66  
 market scanners 3-64, 3-65, 3-66, 3-67, 3-68, 3-69  
 mComboLegs 4-84  
 Microsoft Visual Basic 2008  
   languages supported 2-22  
 modifying orders 4-81  
 multiple API sessions 4-95

## N

News Bulletin Subscription dialog 5-107  
 news bulletins 5-107, 5-108, 5-109

## O

open orders 4-95, 4-96, 4-97, 4-98, 4-99  
   different types 4-96  
 open orders results 4-96  
 openOrderEnd() 4-98  
 openOrderEx() 4-97, 4-98  
 options 4-90, 4-91, 4-92  
 orders 4-75, 4-76, 4-77, 4-80  
   Algo 4-87  
   combo orders 4-82  
   modifying 4-81  
   what-if 4-81

organization of this book -8

## P

Place Order button 4-77  
 Place Order dialog 4-76  
     Algo Params button 4-87  
     Combo Legs button 4-82  
 placing Algo orders 4-87  
 placing combination orders 4-82  
 placing orders 4-76  
 populateSubscription() 3-66  
 preparing to use the ActiveX API 2-21

## R

real time bars 3-59, 3-60, 3-61, 3-62  
 Real Time Bars button event handler 3-60  
 real time bars events 3-62  
 real-time account monitoring, in TWS 1-17  
 realtimeBar() 3-62  
 realtimeBar() event handler 3-62  
 realTimeBar() parameters 3-62  
 reasons for using an API 1-18  
 registering ActiveX controls 2-27  
 registering third-party ActiveX controls 6-115  
 Req All Open Orders button 4-98  
 Req All Open Orders button event handler 4-98  
 Req Auto Open Orders button 4-99  
 Req Auto Open Orders button event handler 4-99  
 Req Contract Data button 3-71  
 Req Contract Data button event handler 3-71  
 Req Current Time button 5-104  
 Req Executions button 4-100  
 Req Executions button event handler 4-101  
 Req Mkd Depth button 3-48  
 Req Mkt Data button 3-40  
 Req Mkt Data button event handler 3-40  
 Req Mkt Depth button event handler 3-48  
 Req News Bulletins button 5-107  
 Req News Bulletins button event handler 5-108  
 Req Open Orders button 4-96  
 Req Open Orders button event handler 4-97  
 Req Real Time Bars button 3-60  
 reqAllOpenOrders() 4-98  
 reqAutoOpenOrders() 4-99  
 reqContractDetailsEx() 3-72  
 reqCurrentTime() 5-105  
 reqExecutionsEx() 4-101  
 reqFundamentalData() 3-55  
 reqHistoricalDataEx() 3-55  
 reqHistoricalDataEx() parameters 3-55  
 reqMktDataEx() 3-42  
 reqMktDataEx() parameters 3-42  
 reqMktDepthEx() 3-49  
 reqMktDepthEx() parameters 3-49  
 reqNewsBulletins() 5-108  
 reqOpenOrders() 4-97  
 reqRealTimeBarsEx() 3-61  
 reqRealTimeBarsEx() parameters 3-61  
 reqScannerParameters() 3-67

reqScannerSubscriptionEx() 3-67  
 reqScannerSubscriptionEx() parameters 3-67  
 Request All Open Orders 4-96  
 Request Auto Open Orders 4-96  
 Request Contract Details dialog 3-70  
 Request Historical Data dialog 3-53  
 Request Market Data dialog 3-39  
 Request Market Depth dialog 3-47  
 Request Open Orders 4-96  
 Request Real Time Bars dialog 3-59  
 requesting contract data 3-70  
 requesting current time 5-104  
 requesting executions 4-100, 4-101, 4-102  
 requesting historical data 3-53  
 requesting market data 3-39, 3-40  
 requesting market depth 3-47  
 requesting open orders 4-95  
 requesting real time bars 3-59  
 requesting scanner parameters 3-65, 3-67  
 resources, for VB programming help 6-116

## S

sample application  
     connecting to 2-28  
 sample application framework 3-34  
 Sample dialog  
     market data fields 3-43  
 scanner parameters  
     requesting 3-67  
 scannerDataEnd() 3-68  
 scannerDataEnd() event handler 3-69  
 scannerDataEx() 3-68  
 scannerDataEx() event handler 3-68  
 scannerParameters() 3-67  
 scannerSubscriptionEx() 3-66  
 server log levels 5-111  
 Server Logging button 5-110  
 server logging level 5-110, 5-111  
 server time 5-104  
 setServerLogLevel() 5-111  
 snapshot 3-45  
 subscribing to market scanner subscriptions 3-64  
 subscribing to news bulletins 5-107  
 subscription object 3-67

## T

third-party controls, for ActiveX 6-115  
 tickEFP() 3-44  
 tickGeneric() 3-44  
 tickOptionComputation() 3-44  
 tickPrice() 3-44  
 tickSize() 3-44  
 tickString() 3-44  
 Trader Workstation  
     overview 1-14  
 trading window 1-16  
 TWS  
     available API technologies 1-19  
     real-time account monitoring in 1-17

TWS and the API 1-18  
TWS Order Ticket 1-16  
TWS overview 1-14, 1-16  
TWS Quote Monitor 1-16

**U**

updateMktDepth Event Handler 3-50  
updateMktDepth() 3-50  
updateMktDepth() parameters 3-50  
updateMktDepthL2 event handler 3-51  
updateMktDepthL2() 3-51  
updateMktDepthL2() parameters 3-51  
updateNewsBulletin() 5-109  
updateNewsBulletin() parameters 5-109  
using this book -8  
    document conventions -11  
    icons -10  
    organization -8

**V**

VB programming help 6-116  
VB sample application 2-28  
VB.NET 2-22  
    connecting to the sample application using 2-29  
VB.NET sample application 2-28  
viewing the server logging level 5-110  
Visual Basic and VB.NET 2-28

**W**

What If button 4-81  
what-if data 4-81  
whatIf() parameter 4-81

**X**

xml parameter, in scannerParameters() event 3-67