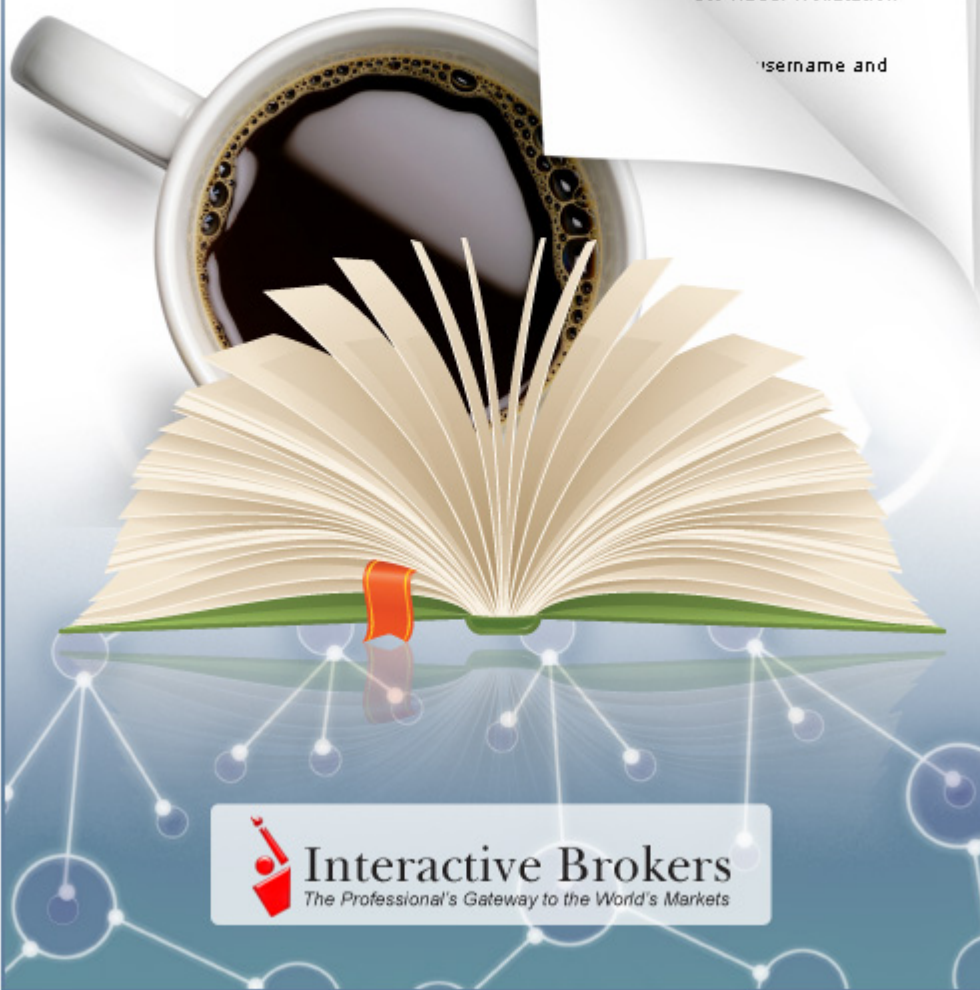


Getting Started with the TWS Java API

Download the software to your PC and
the demo version allows you to access your
account via an internet browser, and is always
free. The full version is available for purchase,
which uses more memory and may run faster, but
includes many new features. To download to

Interactive Trader Workstation

your username and



Interactive Brokers

The Professional's Gateway to the World's Markets

Getting Started with the TWS Java API
November 2013
Supports TWS API Release 9.69

© 2013 Interactive Brokers LLC. All rights reserved.

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

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

Contents

1	Introduction	7
	How to Use this Book	8
	Organization	8
	Part 1: Introducing the TWS Java API	8
	Part 2: Preparing to Use the TWS Java API	8
	Part 3: Getting to Know the Java Test Client	9
	Part 4: Where to Go from Here.....	9
	Footnotes and References	9
	Icons	10
	Document Conventions.....	11
2	TWS and the Java API	13
	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 Java API?.....	18
	TWS and the API	18
	Available API Technologies.....	19
	An Example	19
3	Preparing to Use the Java API	21
	Chapter 3 - Download the Java JDK and IDE.....	22
	Chapter 4- Download the API Software	23
	Chapter 5 - Connect to the Java Test Client.....	27
	Set Up the Java Test Client	27
4	Market Data.....	31
	Chapter 6 - Connect the Java Test Client to TWS	32

Java API Basic Framework	32
Log Into TWS	33
About Logging In.....	33
Enable the API Connection through TWS	35
Connect to TWS.....	36
What Happens When I Click Connect?	38
Disconnecting from a Running Instance of TWS	38
Chapter 7: Requesting and Canceling Market Data	40
What Happens When I Click the Request Top Market Data Link?.....	41
The reqMktData() Method.....	42
EWrapper Methods that Return Market Data.....	44
Getting Frozen Market Data.....	46
Getting a Snapshot of Market Data.....	47
Canceling Market Data.....	47
Chapter 8 - Requesting and Canceling Market Depth	49
What Happens When I Click the Request Deep Market Data Link?.....	50
The reqMktDepth() Method.....	50
The updateMktDepth() and updateMktDepthL2() Methods.....	52
Canceling Market Depth.....	52
Chapter 9 - Requesting and Canceling Historical Data	53
What Happens When I Click the Historical Data Link?	54
The reqHistoricalData() Method	54
The historicalData() Method.....	56
Canceling Historical Data	57
Chapter 10 - Requesting and Canceling Real Time Bars.....	58
What Happens When I Click the Request real-time bars Link?	59
The reqRealTimeBars() Method	59
The realtimeBar() Method.....	60
Canceling Real Time Bars.....	60
Chapter 11 - Subscribing to and Canceling Market Scanner Subscriptions.....	61
What Happens When I Subscribe to a Market Scanner?.....	62
The scannerData() Method	63
The scannerDataEnd() Method	63
The reqScannerParameters() Method	64
Cancel Methods	64
Chapter 12: Requesting Contract Data.....	65

What Happens When I Request Contract Data?	65
The reqContractDetails() Method	66
The contractDetails() Method	67
5 Options.....	69
Chapter 13: Viewing Option Chains	70
What Happens When I Submit a Request to View Option Chains?	70
The reqMktData() Method.....	71
EWrapper Methods that Return Market Data.....	72
Chapter 14: Exercising Options.....	74
What Happens When I Exercise an Option or Let an Option Lapse?	74
The exerciseOptions() Method	75
6 Orders and Executions.....	77
Chapter 15: Placing and Canceling an Order	78
What Happens When I Place an Order?	80
The placeOrder() Method.....	81
The orderStatus() Method	83
Order IDs	83
Modifying an Order	84
Attaching an Order to an Existing Order.....	85
Checking Post-Trade Margin Requirements	86
Canceling an Order	87
Chapter 16: Extended Order Attributes	88
Misc	88
Advisor	88
Volatility	89
Scale	90
IB Algo	91
7 Account and Portfolio Information	93
Chapter 17: Retrieving Account and Portfolio Information.....	94
What Happens When I Retrieve My Account Information?.....	95
The reqAccountUpdates() Method	95
Desubscribing	96

8 Where to Go from Here..... 99

Chapter 25 - Additional Resources	100
Help with Java Programming	100
Help with the Java API.....	100
The API Reference Guide	100
The API Beta and API Production Release Notes.....	101
The TWS API Webinars.....	101
API Bulletin Board	101
IB Customer Service	101
IB Features Poll.....	101

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 Java 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 Java API, see the [Getting Started with the TWS Java API for Advisors Guide](#).

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!



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 Java API - we don't really help you to learn Java. If you aren't a fairly proficient Java 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 Java programming book, and come back to us when you're comfortable with Java.

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 Java 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 Java platform 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 Java 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 Java Test Client 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: Getting to Know the Java Test Client

Part 3 gets you working with the Java Test Client: learning how to request, receive and cancel market data, market depth, historical data, how to place an order and monitor your account activity. 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, so instead of duplicating efforts and filling this book up with those important reference tidbits, we provide targeted links to different sections of the users' guide as we need them.

Part 4: 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 5 includes some additional information about linking to TWS using our Java 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 Java JDK working?"

Footnotes and References

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

Icons



TWS-Related



Java Tip



Important!



Take a Peek!



Go Outside!

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.

The Java 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.

This shows you where there is a particularly useful or important point being made.

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

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
Bold	Indicates: <ul style="list-style-type: none">• menus• screens• windows• dialogs• buttons• check boxes• tabs• keys you press• words you type into entry fields• names of classes and methods	On the Tickers page, select a row by clicking the row number in the far left column... Press Ctrl+C to copy...
<i>Italics</i>	Indicates: <ul style="list-style-type: none">• commands in a menu• objects on the screen that you cannot select, such as text labels• items in drop-down lists	To access the users' guide, under the Software menu, select <i>Trader Workstation</i> , then click <i>Users' Guide</i> .

In addition, Java code snippets appear in the following format:

EClientSocket **constructor**

```
EClientSocket m_client = new EClientSocket(this);
```


TWS and the Java 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 Java 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 TWS Java 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 Java 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 Java version. Why would you use Java over the other API technologies? The main reason might be that you are a Java expert. If you don't know Java 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 Java, this platform offers more flexibility than the DDE for Excel, is supported on Windows, MAC, and Unix/Linux (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 this:

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 start working with the TWS Java API. The next chapter gets you geared up and ready to go.

Preparing to Use the Java 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 - Download the Java JDK and IDE](#)
- [Chapter 4- Download the API Software](#)
- [Chapter 5 - Connect to the Java Test Client](#)



We want to tell you again that this book focuses on the TWS side of the Java API - we don't really help you to learn Java. Unless you are a fairly proficient Java 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 Java programming book, and come back to us when you're comfortable with Java.

Chapter 3 - Download the Java JDK and IDE

OK, well we've already said that you need to know Java before you can successfully implement your own TWS Java API application, and there's a good chance you already have the Java tools you'll need downloaded and installed. But in case you don't, we'll quickly walk you through what you need:

- The Java development kit (JDK)
- An integrated development environment (IDE).

We like the J2SE Development Kit and NetBeans IDE Bundle that's available (free!) from the Sun website. We're not including any version numbers of these Sun Java products, as they'll likely be different by the time you read this.

You can also use Eclipse as your IDE. You can download the Eclipse Java IDE from www.eclipse.org. Of course, you can use any IDE with which you're comfortable.



In this book we use NetBeans as the IDE of choice, so if you're using another IDE you'll have to reinterpret our instructions to fit your development environment. If you're using NetBeans and aren't totally familiar with it, we recommend browsing through the Quick Start or the tutorial, both of which are available on the Help menu.

Anyway, I know 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 the TWS Java API for beginners, not on Java for beginners. 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*.



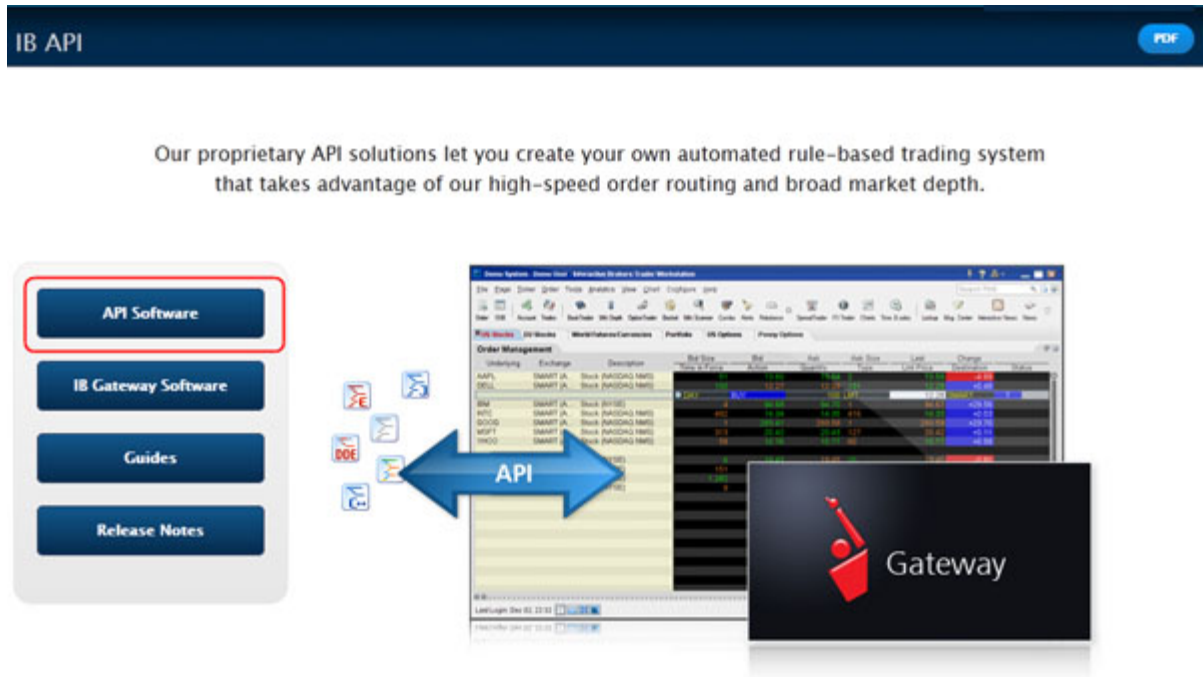
On the API Solutions page, click the **more info** button next to IB API.



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
IB API for Windows Version: API 9.69 Release Date: July 1 2012	IB API for Mac/Unix Version: API 9.69 Release Date: July 1 2012
IB API Beta for Windows Version: API beta 9.70 Release Date: Sep 9 2013	IB API Beta for Mac / Unix Version: API beta 9.70 Release Date: Sep 9 2013
IB API Previous for Windows	Click for Mac Instructions Click for Unix Instructions
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: API Reference Guide or IB Discussion Forum	

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 Java Test Client, connect to TWS, and get started using the Java API sample application!

Chapter 5 - Connect to the Java Test Client

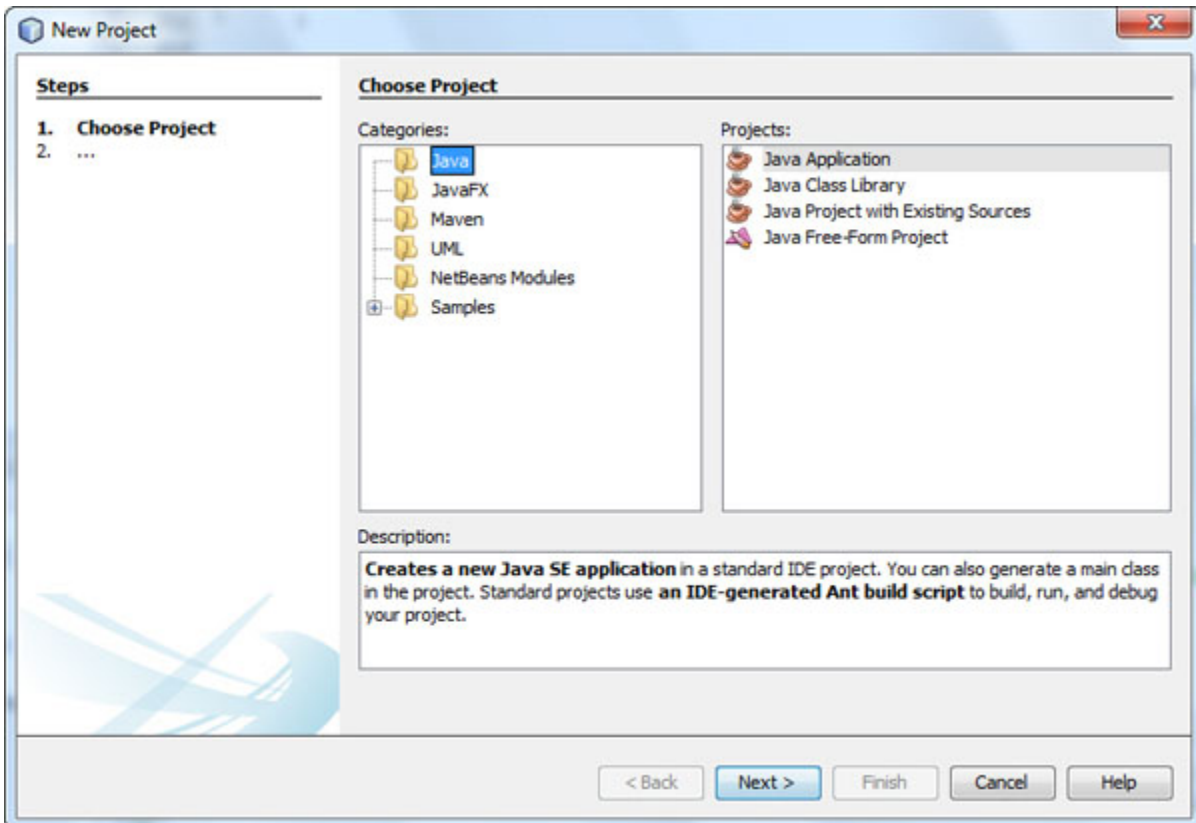
OK, you've got all the pieces in place. Now that we're done with the prep work, it's time to get down to the fun stuff.

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 Java 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. (Don't worry about running out of simulated money; you can log into Account Management with your PaperTrader test account credentials and request more at any time!)

Enabling TWS to support the API is probably the simplest step you'll encounter in this book. It's probably more difficult to actually remember to log into TWS before you run the API!

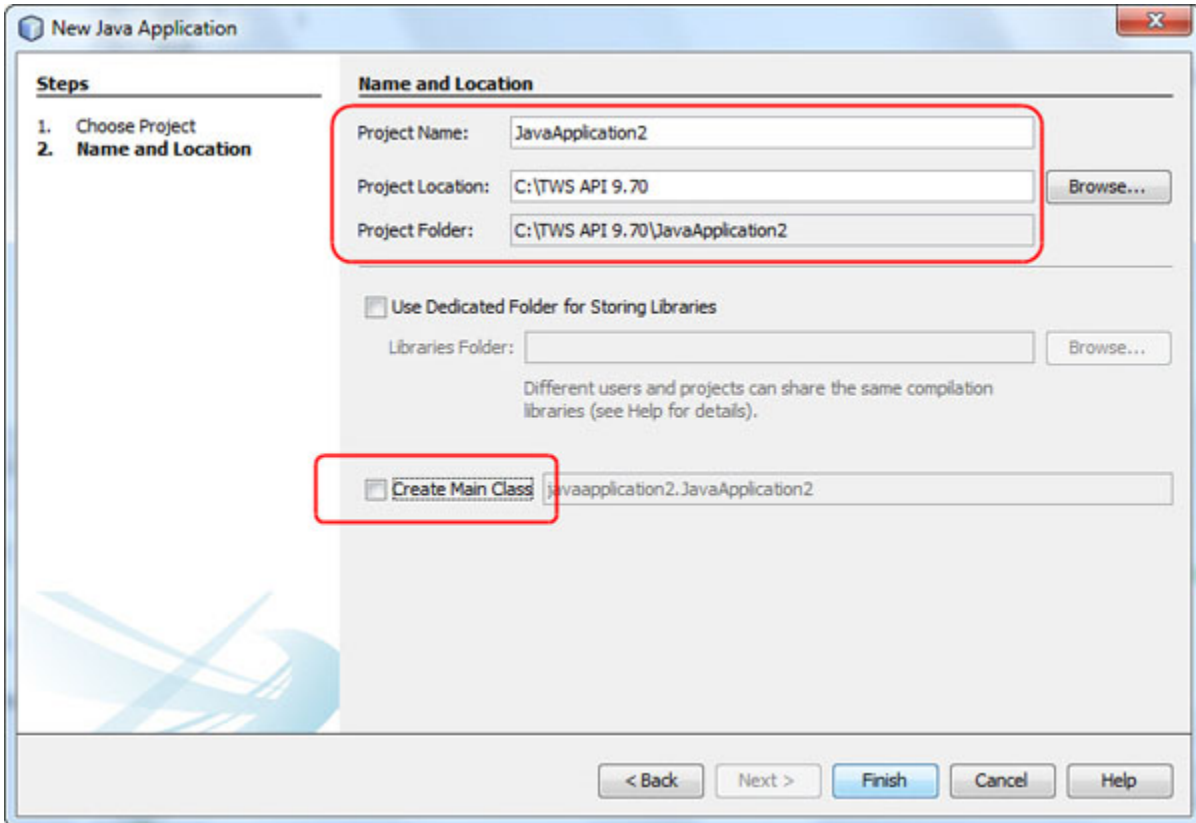
Set Up the Java Test Client

The first thing you're going to do is set up the Java Test Client. Go ahead and open NetBeans, then click *New Project*. This starts the project wizard.



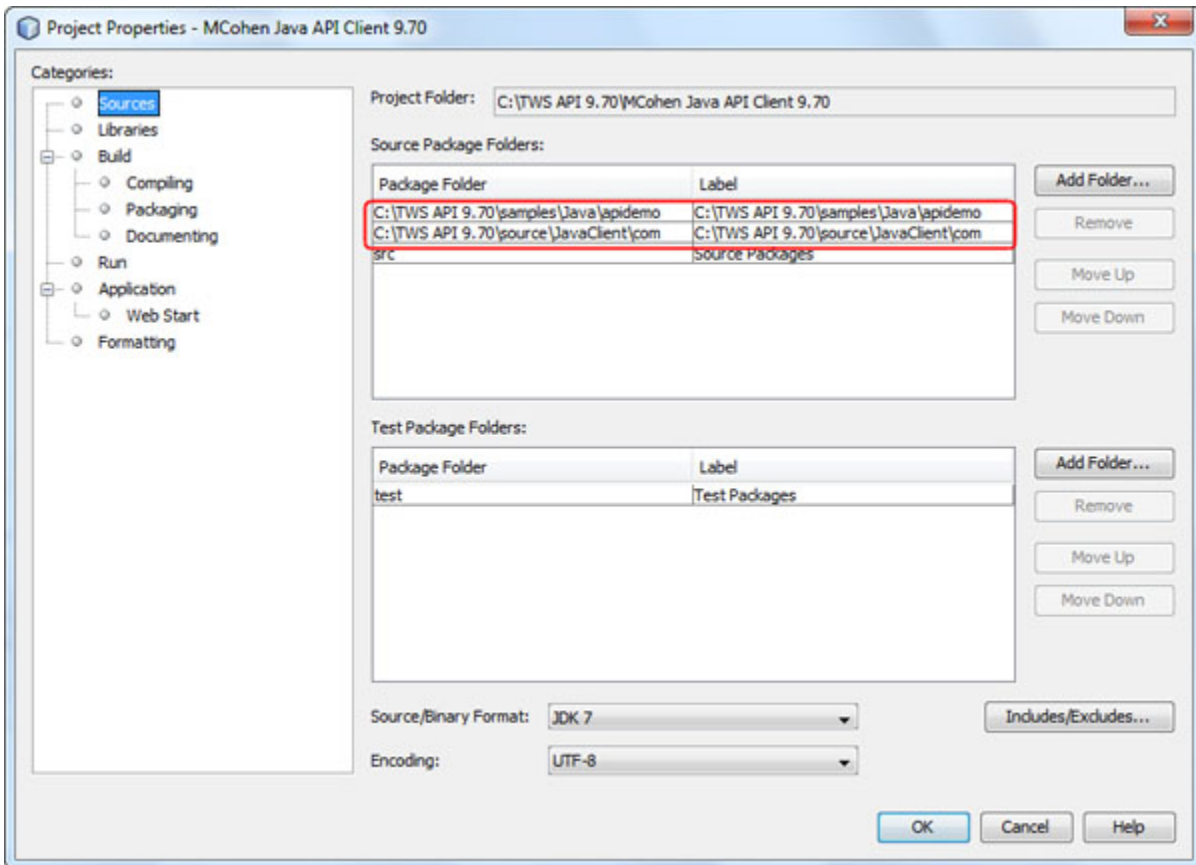
By default, Netbeans starts a new project with Java selected as the Category and Java Application selected as the Project (as shown in the above screen). These are the correct settings for you, so just click **Next**.

On the next screen in the wizard, enter a project name, project location and project folder. Uncheck the box for *Create Main Class* and click **Finish**.



Your new Java project is created and opens. Now right-click your new project from the Projects list and select *Properties*.

The Project Properties dialog opens.



In the Source Package Folders area, click **Add Folder** and navigate to the directory where you installed the API sample program. Add two folders:

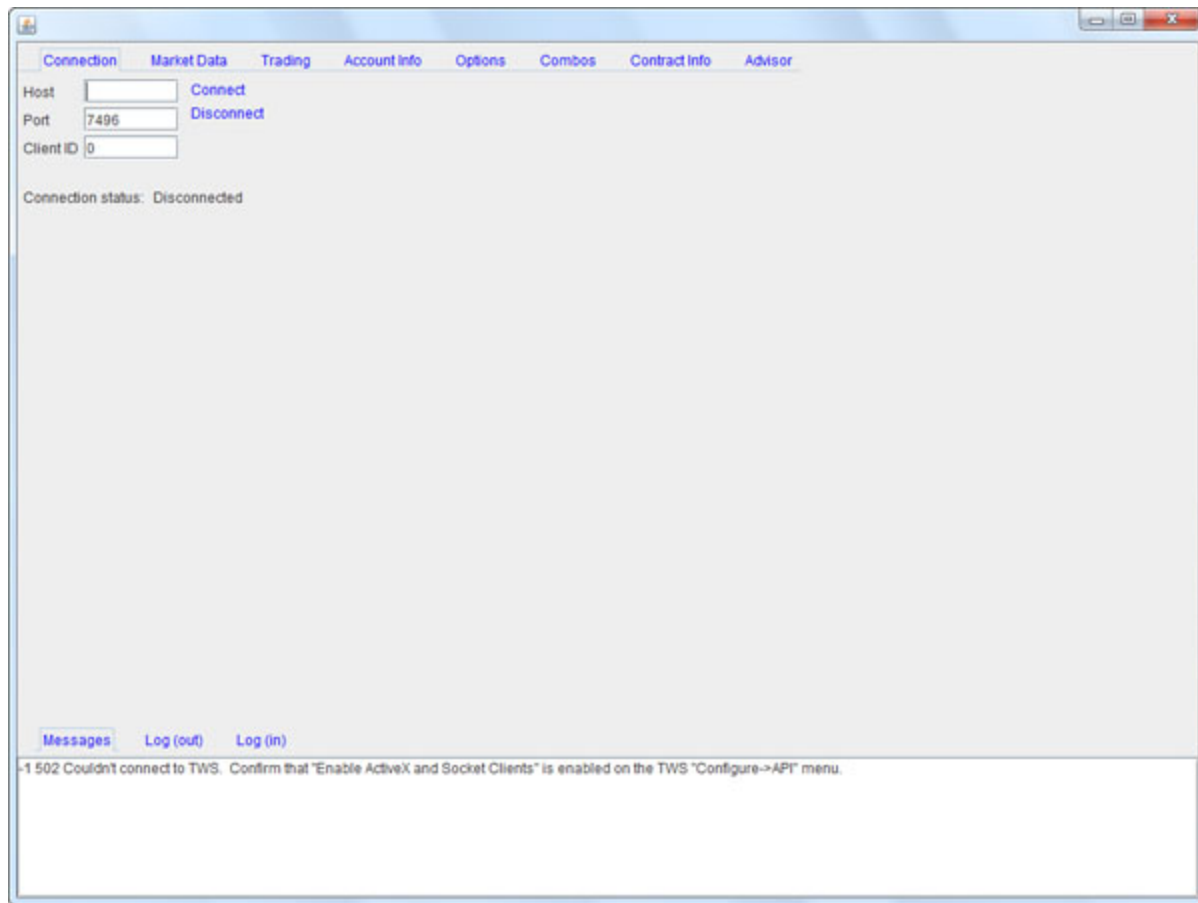
- \samples\Java\apidemo
- \source\JavaClient\com

Then click **OK**.

Step 3: Run the Java Test Client.

Now it's time to run the application. Press **F6** to run. When the system prompts you to select a main class, click **OK** (recall that earlier, you had to uncheck the *Create Main Class* box when you first set up the project; now is the time when you assign the main class). And of course, click **OK** again.

Now press **F6** to run again. You're looking at the java test client, and you should see something like this thing below:



Here you are. What now? Part II focuses on performing trading tasks supported 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 the TWS API, in the order defined by the tabs in the Java Test Client layout, pictures above.



The TWS API does not have to be written as a GUI program, but to completely understand how the Java Test Client works, you should have some general understanding of Java Swing. We recommend searching for Swing-related materials on the Java web site at oracle.com, which include full documentation and tutorials as well.

Market Data

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

In the following chapters, we'll show you the methods and parameters behind this sample application, and how they call the methods and parameters in the TWS Java API.

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

- [Chapter 6 - Connect the Java Test Client 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](#)

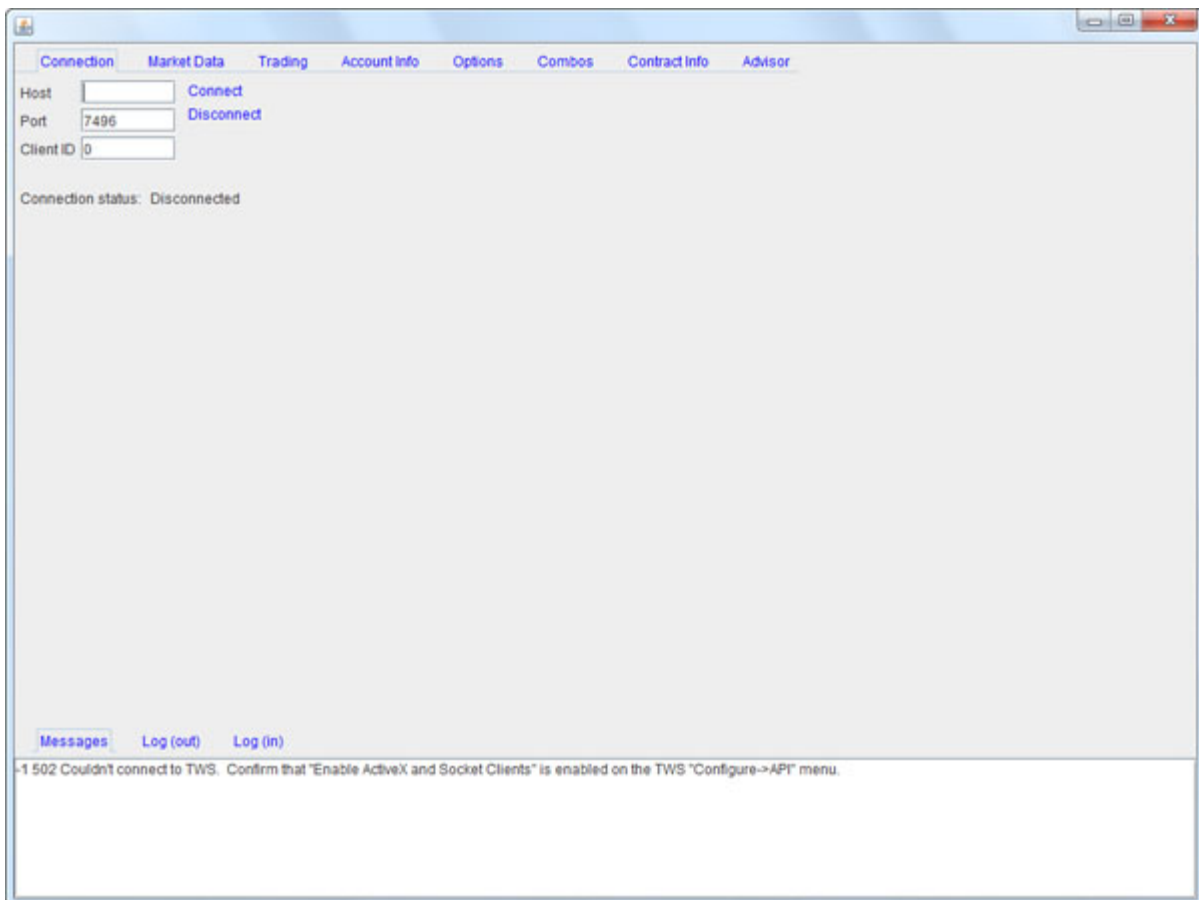
Using the Java Test Client is a good way to practice locating and using the reference information in the API Reference Guide. With the sample program, you can compare the data in the sample message with the method parameters in the API Reference Guide.

Chapter 6 - Connect the Java Test Client to TWS

This chapter describes the basic framework of the Java Test Client and what happens when you connect and disconnect to a running instance of TWS.

Java API Basic Framework

Let's take a look at the basic framework of the Java API. Here's the Java Test Client when you first run it:



As you can see, we've designed the Java Test Client to make it easy to find the most common functions; each group of function is displayed on its own tabbed panel and you can easily switch between panels by clicking the appropriate tab at the top of the Java Test Client window.

The Java API contains the following packages:

- **apidemo** - The classes in apidemo make up the actual Java Test Client. If you look at the code, you will see that the names of the classes correspond to the various panels in the Test Client.
- **apidemo.util** - This package contains utility classes used by the Java Test Client. In case you were wondering, you can also reuse these classes for your own test client user interface.
- **com.ib.client** - This package contains all of the classes that you will need to actually use the Java API. Two or the more notable classes in this package are **EclientSocket**, which contains the methods used to send messages to TWS, and the **EWrapper** interface, which defines the methods that receive messages from TWS.
- **com.ib.contract** - This package contains subclasses of the Contract class that were used in the old Java Test Client. We have kept this package intact for backward compatibility, but you can consider these subclasses to be deprecated.
- **com.ib.controller** - This is a new package that provides an alternate way of using the Java API. The new Java Test Client uses the classes in this package but because this information is more advanced than this Getting Started Guide, we won't get into details.



Throughout this book, we've included links to related help topics in the online API Reference Guide. So if you see a link, feel free to click it if you want to learn more about a particular class or method in our TWS Java API.

Log Into TWS

Of course, the first thing you have to do before you can even begin to use the Java Test Client is log into TWS. Once you're logged in, you can connect the Java Test Client to TWS.

About Logging In

Logging into TWS is easy. You can run the TWS from your Internet browser (which is the recommended method), or download the software to your PC and launch it directly from your desktop as a standalone application.

The browser-based version:

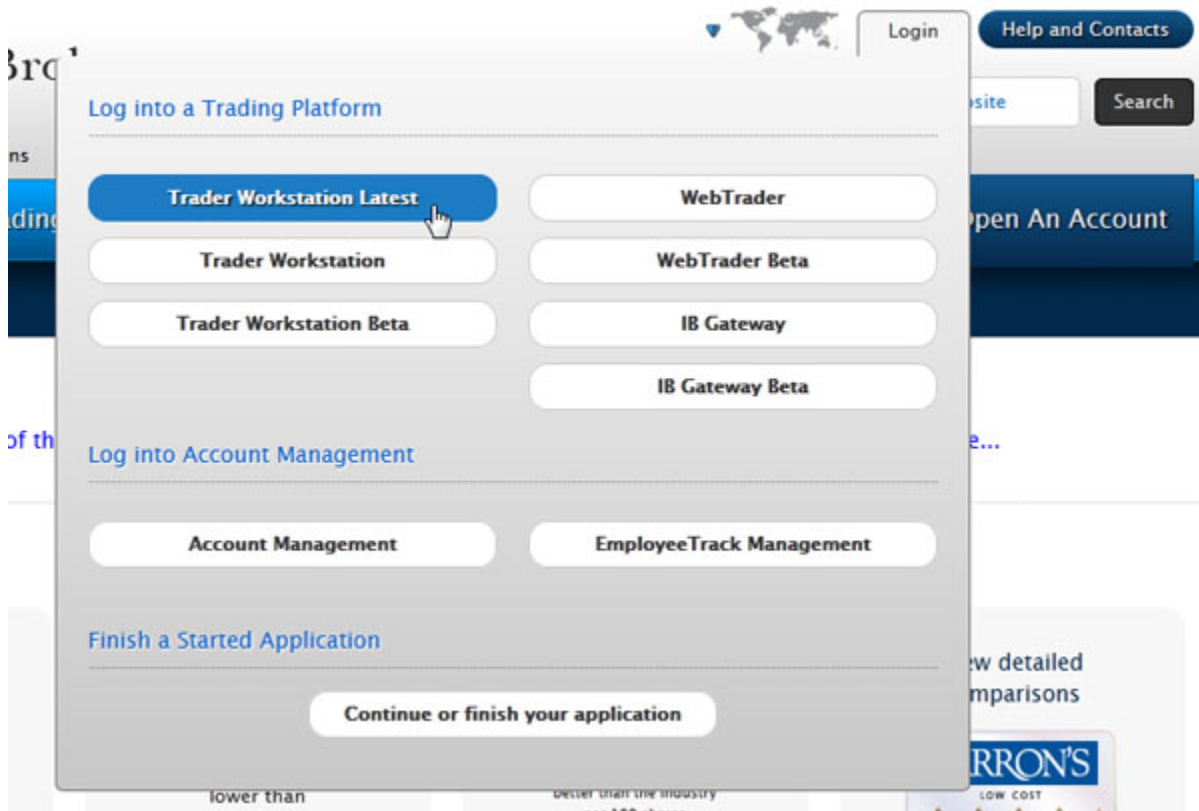
- Allows you to access your account and execute trades from any Java™-enabled internet browser.
- Is always running the latest release.
- Allows you to save your settings from your primary machine to a server so that your TWS will look exactly the same regardless of what Internet machine you use to log in.



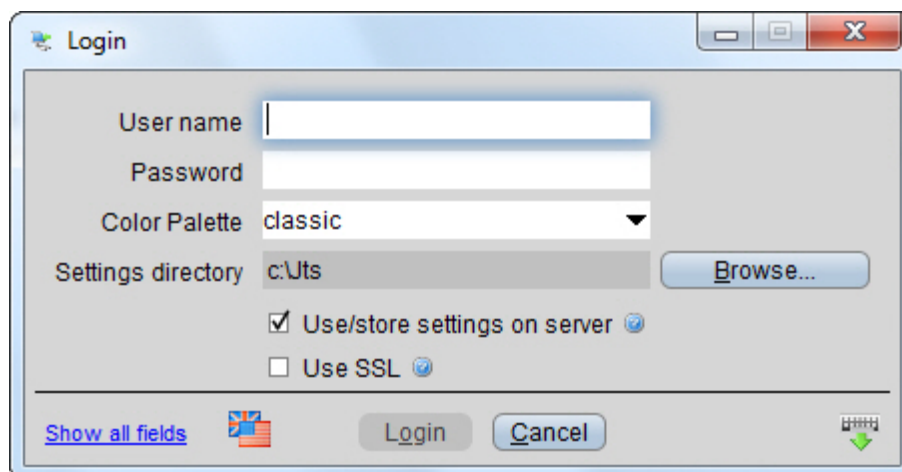
The standalone version uses less memory and may run faster, but requires you to download each release to take advantage of new features. To download to your PC, see the [Installation Instructions](#) on the website.

To log into TWS:

- 1 From the Login menu on our website, select **Trader Workstation Latest** from the drop-down list. You can also log into the previous version of TWS and the current TWS Beta version.



- 2 In the Login box, enter your username and password, and click **Login**.



Login Box Options:

- The color palette allows you to choose a new color skin for TWS. If you select a palette and want to change it once you have logged in, use the Display> Style page in Global Configuration.
- Settings Directory - By default, TWS files are saved in C:\Jts. If you would like to change the location of your settings files, use the Browse button to specify a new directory.
- Use/store settings on server - This option allows you to save your settings, including market data, pages etc., on our server instead of on your local machine. If checked, your workstation will have the same data and look regardless of your login location. If unchecked, settings are only saved on the computer you are currently using.
- Use SSL - Your identity and trading information are always protected. Checking the Use SSL option gives added privacy and integrity to your order information as it travels over the Internet. Minor performance impacts may be experienced depending on the capabilities of your PC.
- Click Show all fields to select a different language for TWS and to have the system migrate settings that you may have saved under a different user name.
- Mouse-driven login - Click the Keyboard icon in the title bar to enter your username and password using the mouse instead of your computer keyboard. Use the mouse to click the appropriate keys on the clickable keyboard display.

Enable the API Connection through TWS

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

To enable API connection through TWS

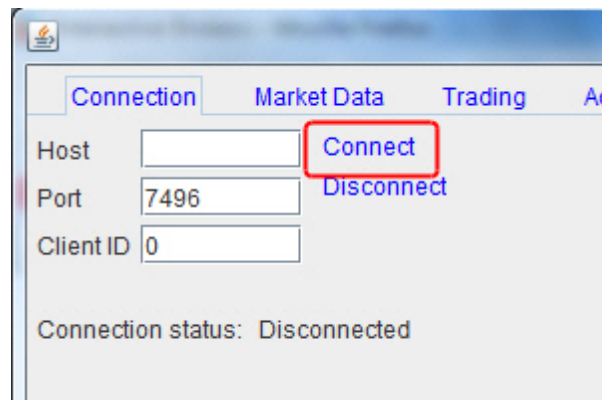
- 1 On the **Edit** menu in TWS, select *Global Configuration*. Then select **API** in the left pane, then click **Settings**.
- 2 In the right pane, click the check box for *Enable ActiveX and Socket Clients (ActiveX, C++ and Java API connections)*. You must have this setting enabled to connect to the API through TWS.

You're all set and you can now connect the Java Test Client to TWS and start learning about all the great features supported by the Java API!

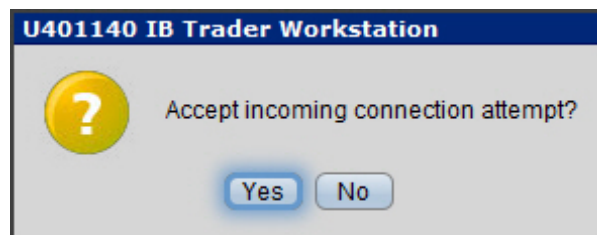
Connect to TWS

At this point, you should be logged into TWS and have the Java Test Client open. The first thing you need to do is connect the Java Test Client to TWS. The Connection panel is displayed by default.

To connect to TWS, simply fill in the fields then click the **Connect** link (pictured below). You'll notice that the Port and Client ID fields are filled in for you already; you can enter an IP address in the Host field and enter a different number in the Client ID field. Notice that the current connection status (Disconnected) is displayed for you.

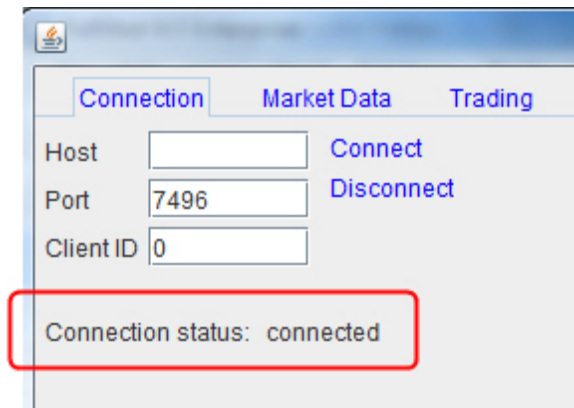


The following dialog opens when you try to connect to TWS:



Click **Yes** to tell TWS to accept your connection.

If the connection is successful, the Connection status in the Java Test Client displays "Connected" and various messages appear in the Messages panel at the bottom of the Java Test Client.



What Happens When I Click Connect?

When you click the **Connect** link, the API calls the **eConnect()** method in EClientSocket. The entries in the Connection panel fields (Host, Port and Client ID) are passed to TWS as attributes of **econnect()** as shown below.

The eConnect() method

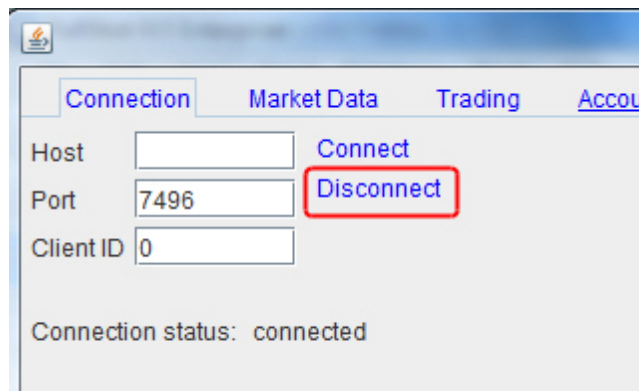
```
public synchronized void eConnect( String host, int port, int
clientId) {
    // already connected?
    host = checkConnected(host);
    if(host == null){
        return;
    }
    try{
        Socket socket = new Socket( host, port);
        eConnect(socket, clientId);
    }
    catch( Exception e) {
        eDisconnect();
        connectionError();
    }
}
```

This method must be called before any other method. Why? Well, because you have to be connected to TWS before you can send it any other messages!

There is no feedback for a successful connection, but any subsequent attempt to connect while you're already connected will return the message "Already connected." As you can see in the method code above, we have some code that runs when you click the **Connect** link but are already connected to TWS.

Disconnecting from a Running Instance of TWS

To disconnect the Java Test Client, or any API application you have built and are running, from TWS, simply click the **Disconnect** link on the Connection tab.



When you click the **Disconnect** link, we call the **eDisconnect()** method in the Java API EClientSocket object.

The edisconnect() method

```
public synchronized void eDisconnect() {
    // not connected?
    if( m_dos == null) {
        return;
    }
    m_connected = false;
    m_serverVersion = 0;
    m_TwsTime = "";

    FilterOutputStream dos = m_dos;
    m_dos = null;
    EReader reader = m_reader;
    m_reader = null;

    try { // stop reader thread; reader thread will close input stream
        if( reader != null) {
            reader.interrupt();
        }
    }
    catch( Exception e) {
    }

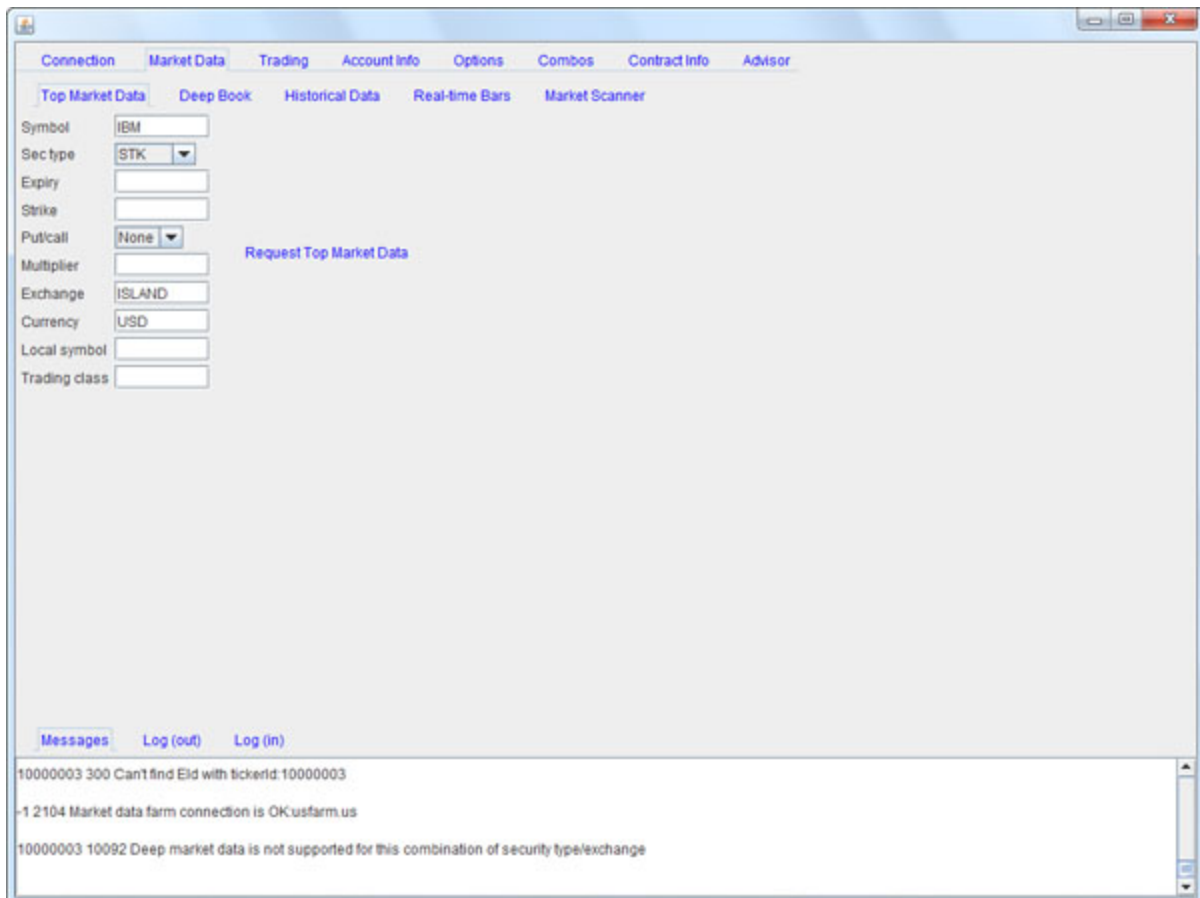
    try {
        // close output stream
        if( dos != null) {
            dos.close();
        }
    }
    catch( Exception e) {
    }
}
```

As you can see, there's more going on in the edisconnect() method than we're describing here, but if you're an experienced Java programmer, you can look into the code in more detail yourself. For the rest of this guide, we'll usually just show you the header portion of a method.

Now let's move on, and see what happens when you call for market data.

Chapter 7: Requesting and Canceling Market Data

This chapter describes how the Java Test Client requests and cancels market data. As you can see, the Java Test Client has conveniently put the Market Data interface on its own tab:



What Happens When I Click the Request Top Market Data Link?

Once you connect to TWS using the Java Test Client, you get market data by clicking the Market Data tab, then entering an underlying and some other information in the Top Market Data fields, and clicking **Request Top Market Data**.

The screenshot shows the TWS Java Test Client interface with the 'Market Data' tab selected. Within this tab, the 'Top Market Data' sub-tab is active. A red rounded rectangle highlights the input fields on the left, which include: Symbol (containing 'IBM'), Sec type (containing 'STK'), Expiry (empty), Strike (empty), Put/call (containing 'None'), Multiplier (empty), Exchange (containing 'ISLAND'), Currency (containing 'USD'), Local symbol (empty), and Trading class (empty). To the right of these fields, a red rectangle highlights the 'Request Top Market Data' button.

When you click the **Request Top Market Data** link, we call the **reqMktData()** method, and the information you entered into the fields shown above are sent to TWS as parameters of that method.

The reqMktData() Method

Let's find out which parameters to use for requesting market data. The Class EClientSocket **reqMktData()** method looks like this:

```
public synchronized void reqMktData(int tickerId, Contract
contract, String genericTickList, boolean snapshot) {
.
.
.
```

If you look at the actual code, you'll see that reqMktData() has a lot more going on than we're discussing here. For example, the method also checks the server version of TWS and displays appropriate messages alerting you of features that are not supported in that TWS version. For now though, we're just concerned with the initial parameters that **reqMktData()** sends to TWS in your quest for market data.

As you can see in the following table, this method has four parameters:

Parameter	Description
tickerId	The ticker id. Must be a unique value. When the market data returns, it will be identified by this tag. This is also used when canceling the market data.
contract	This class contains attributes used to describe the contract.
genericTicklist	A comma delimited list of generic tick types.
snapshot	Check to return a single snapshot of market data and have the market data subscription cancel. Do not enter any genericTicklist values if you use snapshot.

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

Now let's take a closer look at the Top Market Data entry fields you filled in when you clicked the **Req Mkt Data** link and see how and where the two relate.

You wouldn't know it by looking at it, but the circled section in the picture above corresponds to the parameters in the **reqMktData()** method, specifically attributes that are sent with *contract*, itself a parameter of the method. The *Symbol* field corresponds to the String attribute *m_symbol*, the *Sec type* field corresponds to the String attribute *m_secType*, and so on.



For a complete list of ALL of the attributes in the contract class, see the [API Reference Guide](#).

Once you have these parameters filled out to your satisfaction and click **Request Top Market Data**, you're basically sending a message to TWS asking to see market data for the specific contract. TWS will receive this message and reply with your requested market data. Without changing anything, let's use the data in the image above to see what happens next.



The Symbol, Security Type, Exchange and Currency values are required for all instrument types. If your security type is STK, the values to the left 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. I mean, it's really cool and can do a lot of things, but it can't read minds! The moral: be sure you include values in the appropriate fields based on what return values you want to get.

TWS returns the market data values on a separate Top Data tabbed area as shown in the screen below:

Description	Bid Size	Bid	Ask	Ask Size	Last	Time	Change	Volume
IBM STK ISLAND	1	192.71	192.73	1	192.73	10:56:13		8,461

EWWrapper Methods that Return Market Data

The Java API includes an EWWrapper interface. EWWrapper includes all of the methods that return data to the API in response to a method being sent. In the case of market data, it is returned from TWS via the following methods in the **EWWrapper** interface:

tickPrice()

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

tickSize()

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

tickOptionComputation()

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

tickGeneric()

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

tickString()

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

tickEFP()

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

Let's take a closer look at these different methods. First, you should notice that they all share the parameter *tickerID*, which simply binds the returned market data to the correct **reqMktData()** call.

Also take a look at the actual data that is returned to the Java Test Client: Description, Bid Size, Bid (price), Ask Size, Ask (price), Last, Time, Change and Volume. Our original request for market data was for stock, so we can ignore **tickOptionComputation()** and **tickEFP()** for now because those methods return data for option underlyings and EFPs, respectively.

Now let's look at the other tick methods. **tickPrice()** and **tickSize()** return, as you might expect, price and size information. Both methods contain the *field* parameter, which is an integer that specifies the type of price and size, respectively.

- In **tickPrice()**, the *field* integer can be 1 for bid, 2, for ask, 4 for last, 6 for high, 7 for low and 9 for close. The actual price is sent in the double parameter *price*, and the *canAutoExecute* parameter specifies whether the price tick is available for automatic execution.
- In **tickSize()**, the *field* integer can be 0 for bid size, 3 for ask size, 5 for last size, or 8 for volume. The actual size is sent in the integer parameter *size*.

tickGeneric() and **tickString()** both pass an integer parameter called *tickType*, which represents some aspect of market data, and an additional parameter called *value*, which is the actual value of specified tick type. There are a whole lot of these tick types, so we won't show you all of them here. Examples of integers that represent tick types are 31 for BID_EXCH and 23 for OPTION_HISTORICAL_VOL. Each tick type can only be sent with a specific tick method. There's a lot you can do with market data requests, so we won't get into all of that in this guide. You can see all of the tick types in the API Reference Guide [here](#).

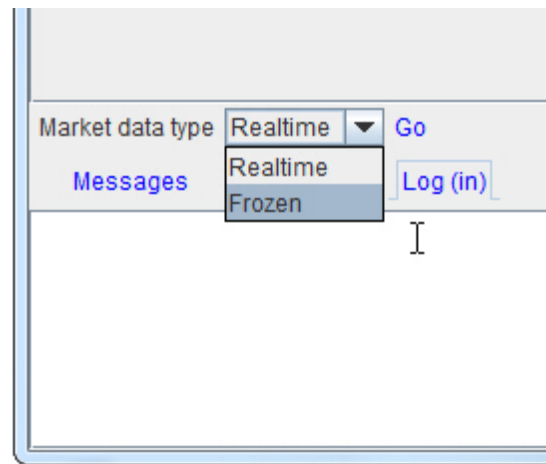


The tick methods are described in the [Java EWrapper Methods](#) section of the API Reference Guide.

Getting Frozen Market Data

The Java Test Client also lets you request frozen market data. What's frozen market data, you ask? Well, frozen market data is simply the last data recorded in our system.

Near the bottom of the Market data tab of the Java Test Client, you will notice a Market data type drop-down menu. To request frozen market data from TWS, simply select *Frozen* from the drop-down, then click **Go**.



When you request frozen market data, you're actually sending an EClientSocket method called **reqMktDataType()**, which is shown below.

mktDataType()

```
void marketDataType(int reqId, int marketDataType)
```

The *marketDataType* parameter is an integer that can be set to 1 for real-time data or 2 for frozen market data. In the Java Test Client, when you select *Frozen* as the Market data type, you are actually setting this parameter to 2, for frozen market data. Needless to say, you can change the Market data type back to *Realtime* to get real-time streaming market data, which would be the same thing as setting the *marketDataType* parameter to 1.

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

During normal trading hours, the API receives real-time market data. If you use the **reqMarketDataType()** call for frozen market data, you are telling TWS to automatically switch to frozen market data after the close. Then, before the opening of the next trading day, market data will automatically switch back to real-time market data.

Whew! Got all that?

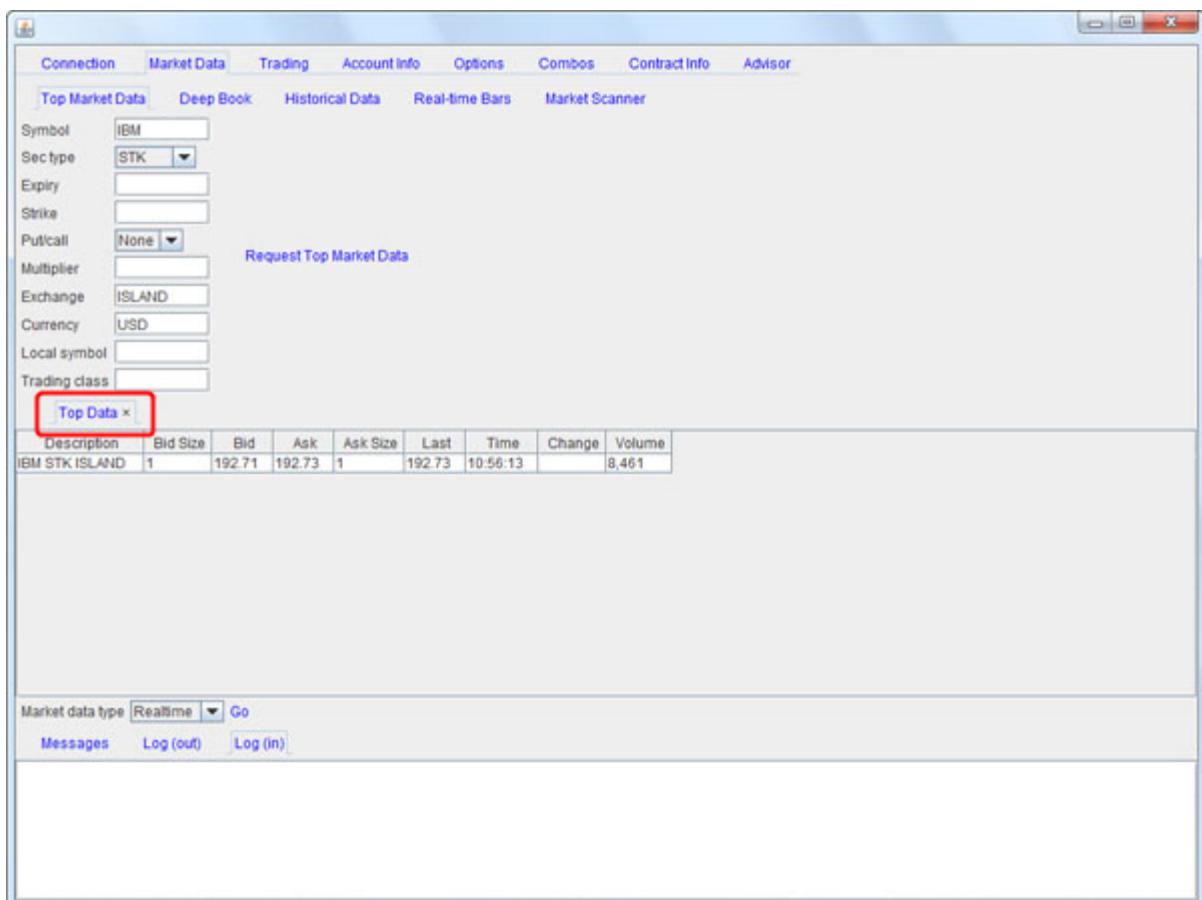
Getting a Snapshot of Market Data

Another way to get market data from TWS to the Java Test Client 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 update on the Top Market Data tab of the Java Test Client, you get a single "snapshot" of the data. This frees you from having to keep up with the changing market data and having to cancel the market data request when you are finished.

While you can't do this directly in the Java Test Client, you can do this in your code by setting the boolean *snapshot* parameter in **reqMktData()** to true. When you get snapshot market data, an additional method in the **EWwrapper** interface called **tickSnapshotEnd()** is returned from TWS to signal the completion of the snapshot.

Canceling Market Data

In the Java Test Client, you can cancel market data by clicking the little "x" next to the Top Data tab, which is circled in red in the following image.



When you elect to cancel market data, the API sends the `EClientSocket` **`cancelMktData()`** method to TWS, and market data for the specified ticker id is canceled. Of course, there's more going on in this method than we're going to discuss in this guide, so go ahead and take a deeper look into the code when you're ready.

`cancelMktData()`

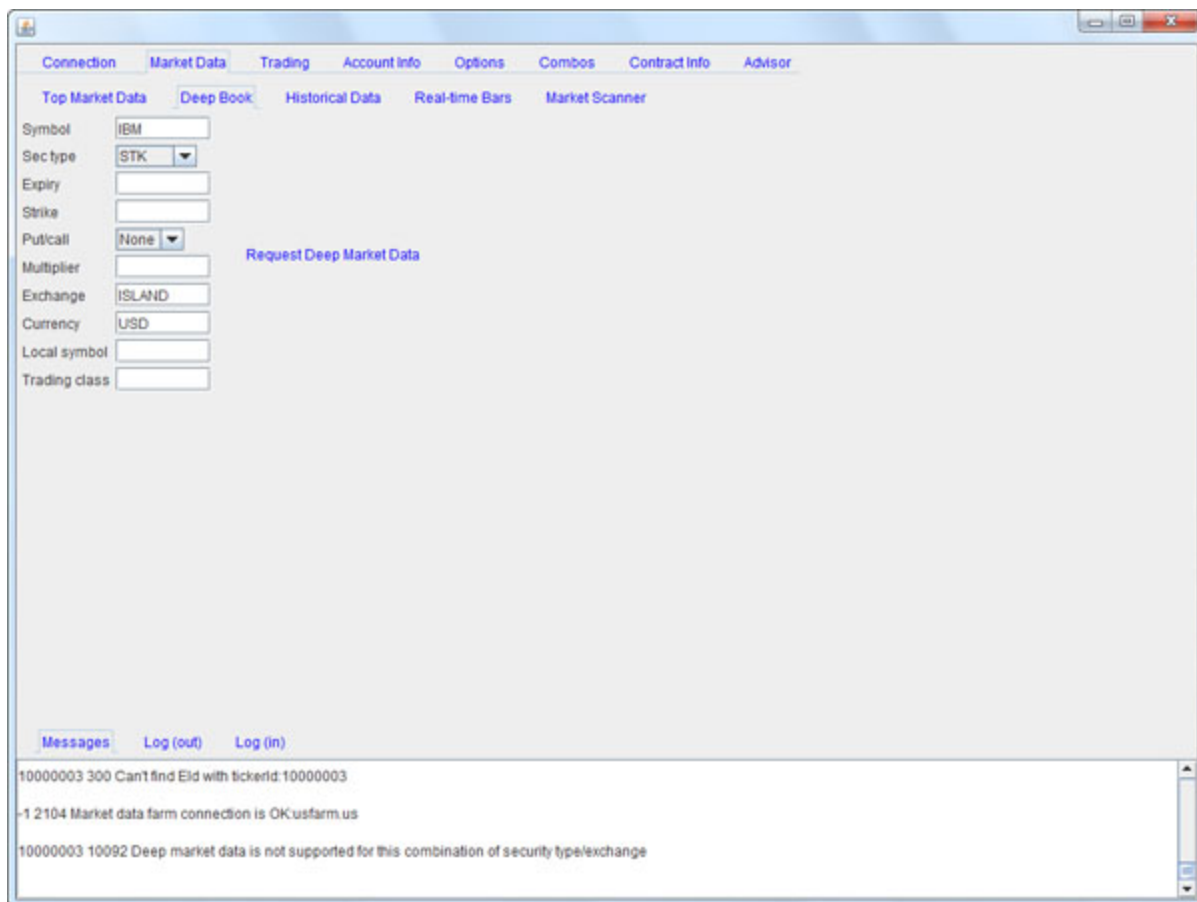
```
public synchronized void cancelMktData( int tickerId)
```

And that's all there is to canceling market data! Next we'll show you how to request market depth, or as it's called in the Java Test Client, "Deep Book."

Chapter 8 - Requesting and Canceling Market Depth

This chapter discusses the methods for requesting and canceling market depth in the Java Test Client. We'll show you the methods and parameters behind the sample application and the methods that return data from TWS.

To request market depth in the Java Test Client, you need to click the Deep Book tab under Market Data. You'll probably notice that this tab looks very similar to the Top Market Data tab. So it stands to reason that requesting market depth (AKA deep market data) in the Java Test Client is pretty much the same as requesting top market data.



What Happens When I Click the Request Deep Market Data Link?

Once you connect to TWS using the Java Test Client, you get deep market data by clicking the Market Data tab, then clicking the Deep Book tab, then entering an underlying and some other information in the Deep Book fields, and clicking **Request Deep Market Data**.

The screenshot shows the TWS Java Test Client interface. The 'Market Data' tab is selected, and the 'Deep Book' sub-tab is active. The 'Deep Book' fields are highlighted with a red rounded rectangle. These fields include: Symbol (IBM), Sec type (STK), Expiry, Strike, Put/call (None), Multiplier, Exchange (ISLAND), Currency (USD), Local symbol, and Trading class. A red rectangle highlights the 'Request Deep Market Data' button.

When you click the link, we make a call to the `EClientSocket reqMktDepth()` method below, which sends the values you entered in the market data fields to TWS.

The `reqMktDepth()` Method

Let's find out which parameters are used when you request market depth. The Class `EClientSocket reqMktDepth()` method header looks like this:

```
public synchronized void reqMktDepth( int tickerId, Contract
contract, int numRows)
```

Parameter	Description
tickerId	The ticker Id. Must be a unique value. When the market depth data returns, it will be identified by this tag. This is also used when canceling the market depth.
contract	This class contains attributes used to describe the contract.
numRows	Specifies the number of market depth rows to return.

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

As you can see from the table, this method has three parameters, one of which, *contract*, we've seen before. And just like before, the fields you fill out on the Deep Book tab in the Java Test Client are attributes of the *contract* class. Once again, we reference example attributes in the *contract* class: the *Symbol* field corresponds to the String attribute *m_symbol*, the *Sec type* field corresponds to the String attribute *m_secType*, and so on.



For a complete list of ALL of the attributes in the *contract* class, see the [API Reference Guide](#).

The deep market data (AKA market depth) will be returned via the **updateMktDepth()** and **updateMktDepthL2()** methods, both part of the **EWwrapper** interface.

In the Java Test Client, the returned deep market data looks something like this:

Connection Market Data Trading Account Info Options Combos Contract Info Advisor

Top Market Data Deep Book Historical Data Real-time Bars Market Scanner

Symbol: IBM
 Sec type: STK
 Expiry:
 Strike:
 Put/call: None
 Multiplier:
 Exchange: ARCA
 Currency: USD
 Local symbol:
 Trading class:
 Request Deep Market Data

Deep IBM x

Mkt Maker	Price	Size
192.4	1	
192.35	1	
192.33	2	
192.32	15	
192.3	4	
192.29	2	

Mkt Maker	Price	Size
192.48	4	
192.49	2	
192.53	1	
192.59	15	
192.6	1	
192.62	1	

Desubscribe

Messages Log (out) Log (in)

10000004 310 Can't find the subscribed market depth with tickerid:10000004
 10000006 354 Requested market data is not subscribed.Error&NYSE/STK/Deep
 10000006 310 Can't find the subscribed market depth with tickerid:10000006

The `updateMktDepth()` and `updateMktDepthL2()` Methods

These EWrapper methods return deep market data:

`updateMktDepth()` returns market depth.

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

`updateMktDepthL2()` returns Level II market depth.

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

Canceling Market Depth

To cancel the deep market data in the Java Test Client, simply click the little “x” next the market data tab labeled “Deep XXX” where XXX is the ticker symbol. When you do this, we call the EClientSocket method **`cancelMktDepth()`**, which sends a message to TWS to stop sending the deep market data.

`cancelMktDepth()`

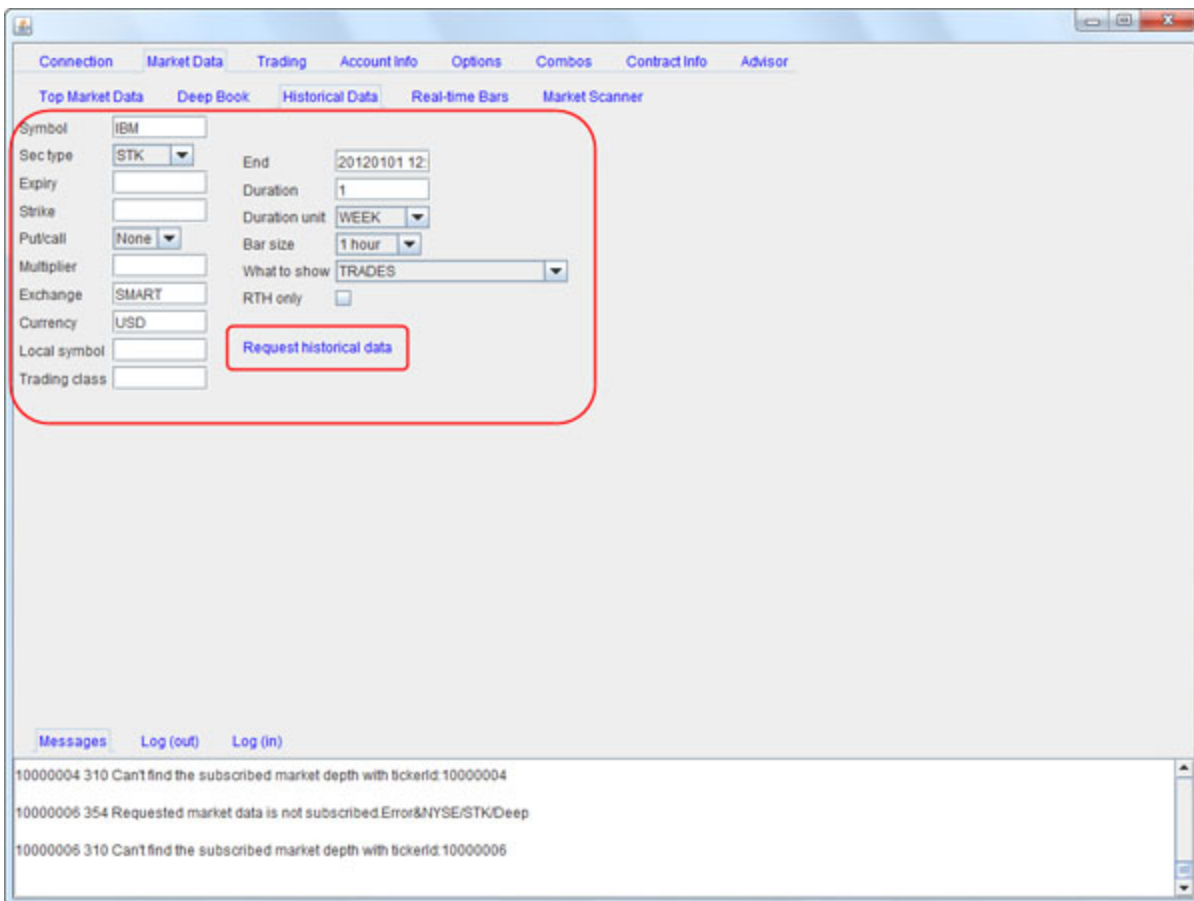
```
public synchronized void cancelMktDepth( int tickerId)
```

Voila! Market depth is canceled!

Chapter 9 - Requesting and Canceling Historical Data

This chapter focuses on requesting and canceling historical data. We'll show you the methods and parameters behind the Java Test Client and how they call the methods in the TWS Java API.

To request historical data in the Java Test Client, you need to click the Historical Data tab under the Market Data tab and use the fields and link circled below:



What Happens When I Click the Historical Data Link?

When you click the **Request historical data** link, we call the EClientSocket method **reqHistoricalData()**, which uses the fields on the Historical Data tab as its parameters.

The reqHistoricalData() Method

So which parameters are used when you request historical data? The parameters in the EClientSocket **reqHistoricalData()** method return the data you request. The **reqHistoricalData()** method header looks like this:

```
public synchronized void reqHistoricalData( int tickerId, Contract
contract, String endDateTime, String durationStr, String
barSizeSetting, String whatToShow,int useRTH, int formatDate)
```

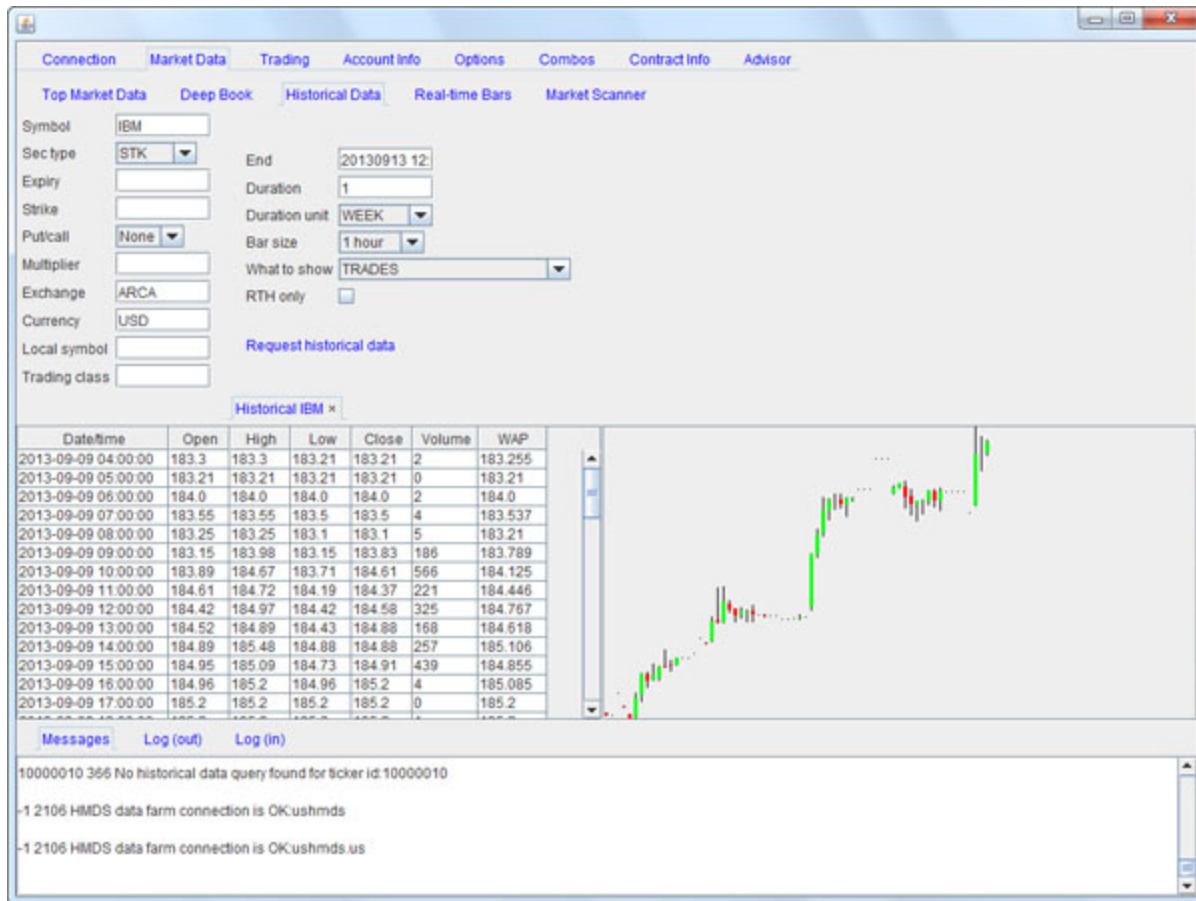
This method has numerous parameters that correspond to the fields on the Historical Data tab in the Java Test Client that you fill in, including end date and time, duration, bar size setting, what to show, regular trading hours, and date format style. You can easily match the parameter to the entry field in the Java Test Client.

Parameter	Description
tickerId	The Id for the request. Must be a unique value. When the data is received, it will be identified by this Id. This is also used when canceling the historical data request.
contract	This class contains attributes used to describe the contract.
endDateTime	Use the format yyyyymmdd hh:mm:ss tmz, where the time zone is allowed (optionally) after a space at the end.
durationStr	<p>This is the time span the request will cover, and is specified using the format: <integer> <unit>, i.e., 1 D, where valid units are:</p> <ul style="list-style-type: none"> • " S (seconds) • " D (days) • "W (weeks) • " M (months) • " Y (years) <p>If no unit is specified, seconds are used. Also, note "years" is currently limited to one.</p>

Parameter	Description
barSizeSetting	<p>Specifies the size of the bars that will be returned (within IB/TWS limits). Valid bar size values include:</p> <ul style="list-style-type: none"> • 1 sec • 5 secs • 15 secs • 30 secs • 1 min • 2 mins • 3 mins • 5 mins • 15 mins • 30 mins • 1 hour • 1 day
whatToShow	<p>Determines the nature of data being extracted. Valid values include:</p> <ul style="list-style-type: none"> • TRADES • MIDPOINT • BID • ASK • BID_ASK • HISTORICAL_VOLATILITY • OPTION_IMPLIED_VOLATILITY
useRTH	<p>Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include:</p> <ul style="list-style-type: none"> • 0 - all data is returned even where the market in question was outside of its regular trading hours. • 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.
formatDate	<p>Determines the date format applied to returned bars. Valid values include:</p> <ul style="list-style-type: none"> • 1 - dates applying to bars returned in the format: <code>yyyymmdd{space}{space}hh:mm:ss</code> • 2 - dates are returned as a long integer specifying the number of seconds since 1/1/1970 GMT.

There are too many to display the entire list of parameters and their values here, so you'll have to check out the [API Reference Guide](#) for more details.

The historical data returned from TWS looks like this in the Java Test Client:



The historicalData() Method

The values are returned via the parameters in the EWrapper interface **historicalData()** method, whose header is shown below.

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

You can see all of this method's parameters in the [historicalData\(\)](#) method topic of the *API Reference Guide*.

Canceling Historical Data

Like the other functions in the Java Test Client that we've already seen, you can cancel your historical data request by clicking the little "x" on the Historical Data results tab.

When you cancel historical data requests, we call the EClientSocket method **cancelHistoricalData()**, and historical data for the specified id is canceled:

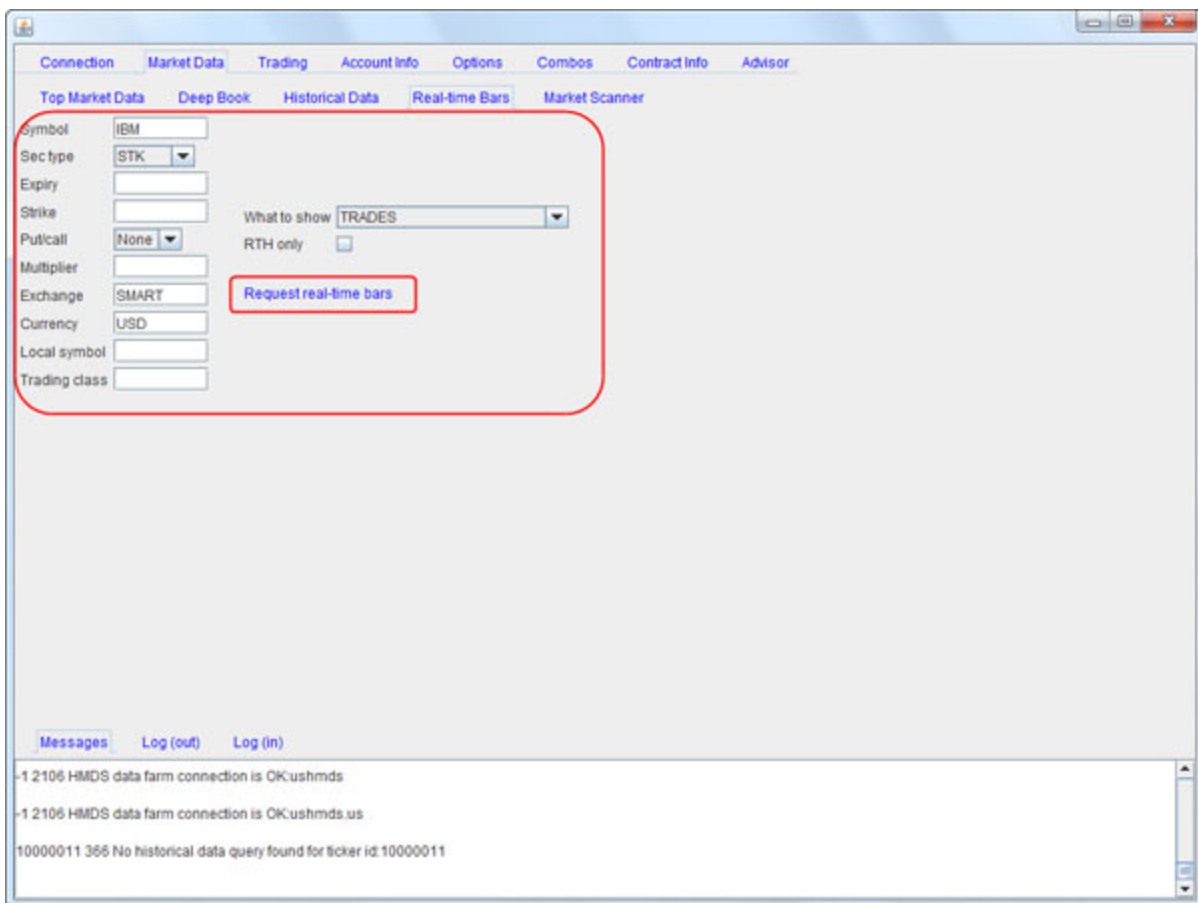
cancelHistoricalData() Method

```
public synchronized void cancelHistoricalData( int tickerId )
```

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 the methods that are called in the TWS Java API.

To request real time bars, you need to use the fields and link on the Real-time Bars tab, which is located under the Market Data tab in the Java Test Client. These fields are circled in the image below:



What Happens When I Click the Request real-time bars Link?

When you click the **Request real-time bars** link, we call the EClientSocket method **reqRealTimeBars()**, which sends the values you entered to TWS (contract information, what to show and whether or not to include data outside regular trading hours).

In the API release supported by this document, the real-time bars default to a size of five seconds. This means that no matter what you enter in the *Bar Size Setting* field in the Sample dialog, the size of the real-time bars you get will be five seconds.

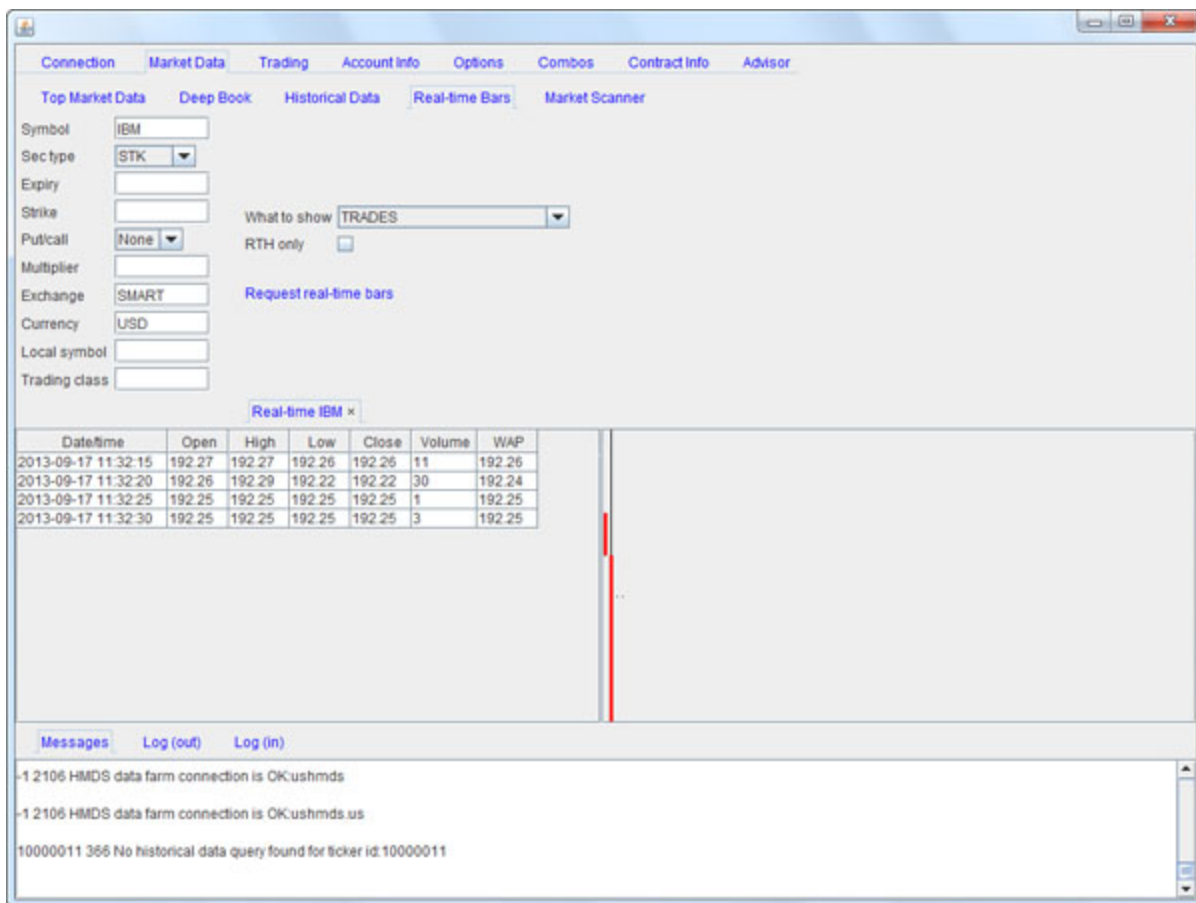
The reqRealTimeBars() Method

The parameters in the EClientSocket **reqRealTimeBars()** method return the data you request. The **reqRealTimeBars()** method header looks like this:

```
public synchronized void reqRealTimeBars(int tickerId, Contract
contract, int barSize, String whatToShow, boolean useRTH)
```

Parameter	Description
tickerId	The Id for the request. Must be a unique value. When the data is received, it will be identified by this Id. This is also used when canceling the request.
contract	This class contains attributes used to describe the contract.
barSizeSetting	Currently only 5 second bars are supported, if any other value is used, an exception will be thrown.
whatToShow	Determines the nature of data being extracted. Valid values include: <ul style="list-style-type: none"> • TRADES • MIDPOINT • BID • ASK
useRTH	Determines whether to return all data available during the requested time span, or only data that falls within regular trading hours. Valid values include: <ul style="list-style-type: none"> • 0 - all data is returned even where the market in question was outside of its regular trading hours. • 1 - only data within the regular trading hours is returned, even if the requested time span falls partially or completely outside of the RTH.

And here's what real-time bars returned to the Java Test Client looks like:



The realtimeBar() Method

The real time bars are returned via the parameters in the EWrapper interface **realtimeBar()** method, whose header is shown below. You'll notice that the column headings in the returned data correspond to the parameters in **realtimeBar()**.

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

Canceling Real Time Bars

Like the other functions in the Java Test Client that we've already seen, you can cancel your real-time bars request by clicking the little "x" on the Real-time results tab.

When you cancel a real-time bars request, we call the EClientSocket method **cancelRealTimeBars()**, and the real-time bars for the specified id is canceled:

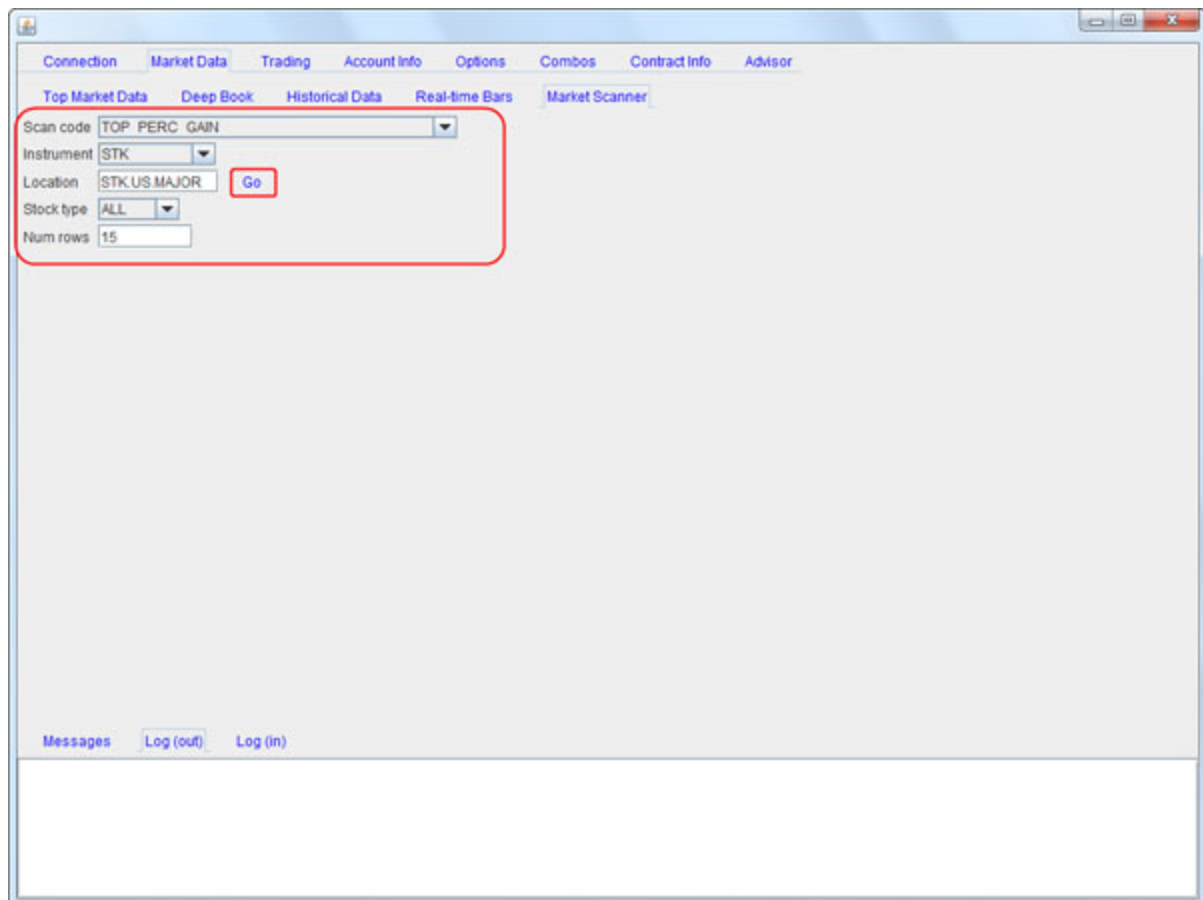
cancelRealTimeBars() Method

```
public void cancelRealTimeBars(int tickerId)
```

Chapter 11 - Subscribing to and Canceling Market Scanner Subscriptions

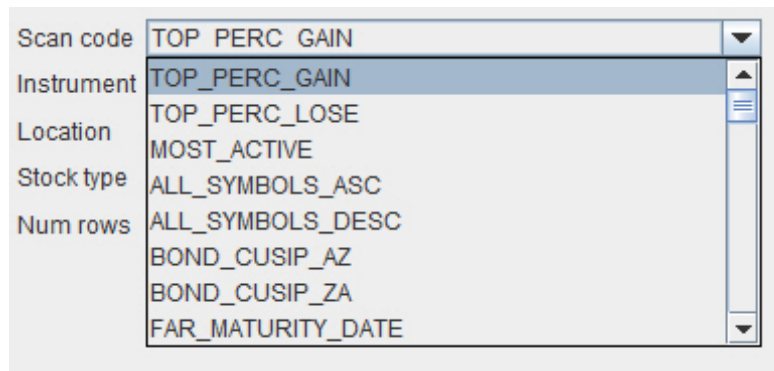
This chapter describes the methods used for requesting market scanner parameters, subscribing to a market scanner, and canceling a subscription to a market scanner.

In the Java Test Client, you subscribe to market scanners on the Market Scanner tab, which is located under the Market Data tab. As you can see in the image below, like most of the other functions supported in the Java Test Client, you fill in some fields and then click a link to submit the request.



What Happens When I Subscribe to a Market Scanner?

To subscribe to a market scanner in the Java Test Client, you first select a scan code, then fill in the rest of the fields, then click the **Go** link. The Scan Code drop-down menu is shown below. There are a lot more available scan codes than the image shows, so make sure you scroll down until you find the right one!



The reqScannerSubscription() Method

When you click the **Go** link, you are “subscribing” to your selected market scanner (AKA Scan Code), and you’re also making a call to the **reqScannerSubscription()** method in our old friend EClientSocket.

As you may have already guessed, this method sends the values you entered in the Market Scanner fields as parameters to TWS. The **reqScannerSubscription()** method header looks like this:

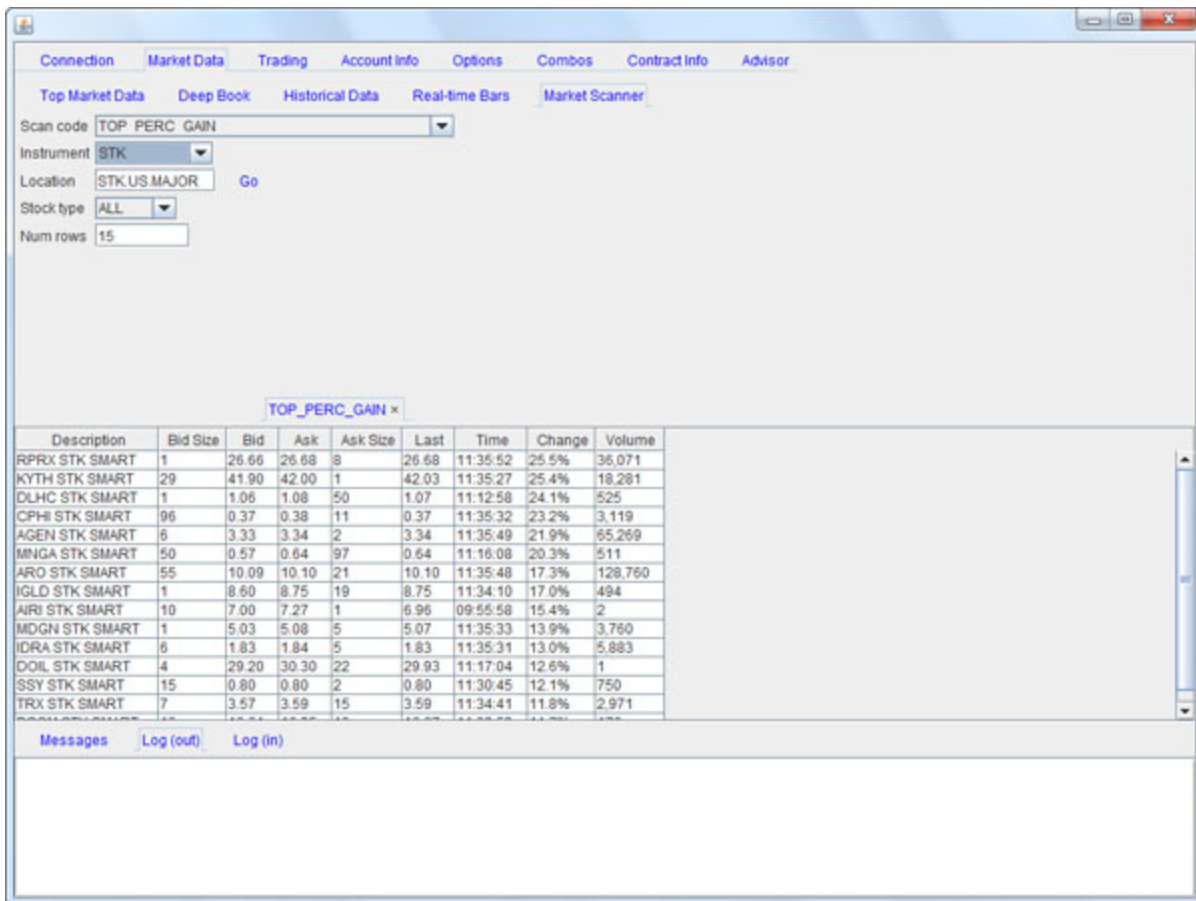
```
public synchronized void reqScannerSubscription( int tickerId,
ScannerSubscription subscription)
```

The parameter *subscription* is another one of those Java SocketClient Properties and it contains a bunch of attributes that correspond to data in the market scanner. You look at the entire list of attributes [here](#).

Parameter	Description
tickerId	The Id for the subscription. Must be a unique value. When the subscription data is received, it will be identified by this Id. This is also used when canceling the scanner.
subscription	Summary of the scanner subscription parameters including filters.

reqScannerSubscription() receives market scanner results from TWS via the EWrapper method **scannerData()** method.

In the Java Test Client, your scan results look like the image on the next page.



The scannerData() Method

All that lovely scanner data that you see is returned from TWS by the EWrapper method **scannerData()**, whose header is shown below:

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

Notice the parameter *contractDetails*. This is a Java SocketClient Property, very much like the *contract* parameter that we saw when we requested market data. *contractDetails* contains a lot of different attributes which represent different aspects of the contract you specified. You can see a list of all of these attributes [here](#).

The scannerDataEnd() Method

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

This method is called after a full snapshot of a scanner window has been received and serves as a sort of end tag. It helps define the end of one scanner snapshot and the beginning of the next.

The reqScannerParameters() Method

There is another method in EClientSocket that relates to market scanners:

reqScannerParameters(). This particular method isn't supported in the Java Test Client, but we thought we'd mention it here because it can be useful. **reqScannerParameters()** receives an XML document that describes the valid parameters that a scanner subscription can have. So in your own Java application, you can make use of this method to display the XML field that lists all valid scan codes. The **reqScannerParameters()** method header looks like this:

```
public synchronized void reqScannerParameters()
```

The EWrapper method **scannerParameters()** returns the aforementioned XML document in its only parameter, *xml*, which is a String.

Cancel Methods

By this time, you may have noticed that the methods that cancel an operation all look similar. **cancelMktData()**, **cancelHistoricalData()**, and so on, are all very simple methods that typically contain a single integer parameter used to bind it to the original request. And you also may have noticed that the cancel operation in the Java Test Client is also similar for all of these functions - you simply close the results tab.

So from here on, we're going to omit the rest of the cancel methods unless we run across one that is different from the rest. We leave it to you to peruse the *API Reference Guide* and look these methods up yourself!

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. Up until now, everything we've done in this section of the guide has taken place on the Market Data tab in the Java Test Client. This chapter introduces you to the Contract Info tab.

What Happens When I Request Contract Data?

To request contract data using the Java Test Client sample application, you'll need to enter data in the fields on the Contract Details tab under the Contract info tab as pictured below. As you can see, we've designed the sample application with consistency in mind; the entry fields on the various tabs all look similar.

The screenshot shows the 'Contract Info' tab in the Java Test Client. The 'Contract details' sub-tab is active. The form contains the following fields and values:

Field	Value
Symbol	IBM
Sectype	STK
Expiry	
Strike	
Put/call	None
Multiplier	
Exchange	SMART
Currency	USD
Local symbol	
Trading class	

A red box highlights the 'Query' button, which is located to the right of the 'Multiplier' field.

Just like the other tabs where you entered information in fields then clicked a link to submit your request, you do the same thing on the Contract details tab. Fill in the fields to define the desired contract, then click the **Query** link to submit your request.

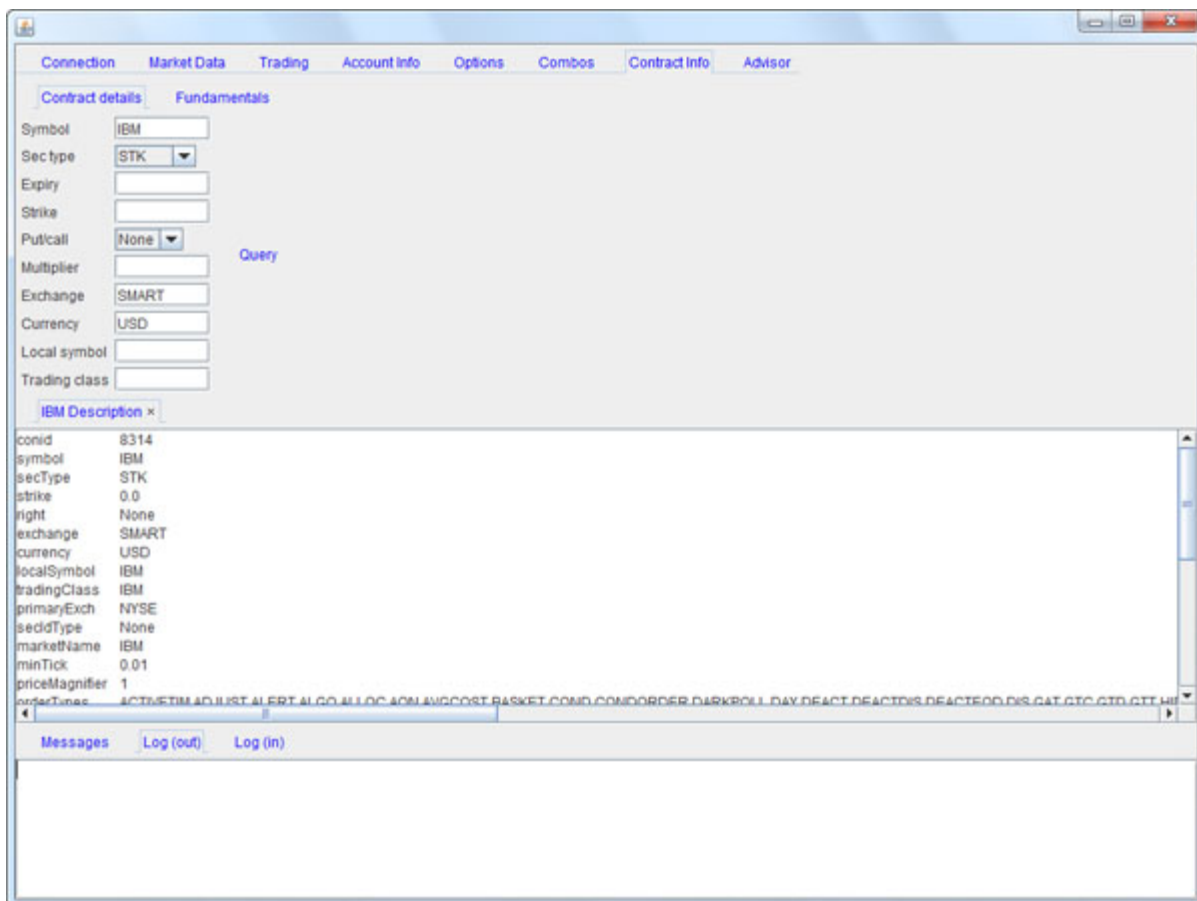
And when you click that link, we call the EClientSocket method **reqContractDetails()**.

The reqContractDetails() Method

The **reqContractDetails()** method, whose header is shown below, contains two parameters: reqID, which is simply the ID of the request, and *contract*. If you recall from earlier chapters, the *contract* parameter contains all the attributes used to describe the requested contract. And, just as we described earlier, the entry fields you fill in correspond to the attributes in *contract*.

```
public synchronized void reqContractDetails(int reqId, Contract
contract) {
```

In the Java Test Client, your contract details are displayed on a tab below the Contract details entry fields, as shown below. There are a lot of details so you'll probably have to use the vertical scrollbar on the right to view all of it.



The `contractDetails()` Method

All that contract data is returned from TWS via the Java API EWrapper method **`contractDetails()`**. This method contains two parameters: *reqID*, the integer that binds this data to the original request, and *ContractDetails*, a `SocketClient Property` that we've run into before that contains all the attributes used to describe the requested contract. So every line of information in the returned contract details corresponds to an attribute in *contractDetails*. For example, the *marketName* line of data in the returned contract details corresponds to a String attribute called *m_marketName* in *contractDetails*.

```
void contractDetails(int reqId, ContractDetails contractDetails)
```

By the way, you can see a list of all of the attributes in *contractDetails* [here](#).

Options

This section describes how the Java API sample application handles option chains and the exercising of options. An option chain is a list of all options available for a specific security. Option chains typically list all put and call option strike prices for a given maturity date. .

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

- [Chapter 13: Viewing Option Chains](#)
- [Chapter 14: Exercising Options](#)

Using the Java Test Client 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 13: Viewing Option Chains

These next two chapters look at the option-related actions in the Java Test Client sample application, which are conveniently grouped together on the Options tab. Actions viewing option chains and exercising options are included here.

What Happens When I Submit a Request to View Option Chains?

Let's take a look at what happens when you submit a request to view option chains in the Java Test Client. In this section, we'll show you the methods and parameters behind the sample application, and how they relate to what you see on the screen.

Just as you did when you performed other tasks in the sample application, you fill in some entry fields and click a link to view option chains. Go ahead and click on the Options tab and you'll see the Option Chains tab with some familiar-looking entry fields.

The screenshot shows the Java Test Client application window with the 'Options' tab selected. Within the 'Options' tab, the 'Option Chains' sub-tab is active. The form contains the following fields and controls:

- Symbol: IBM
- Currency: USD
- Underlying sec type: STK (dropdown menu)
- Underlying exchange: SMART
- Option exchange: SMART
- Use snapshot data: ☐
- Go button (highlighted with a red box)

At the bottom of the window, there is a 'Messages' pane with the following text:

```
Log (out) Log (in)
10000038 430 We are sorry, but fundamentals data for the security specified is not available. Not allowed
10000039 430 We are sorry, but fundamentals data for the security specified is not available. Not allowed
10000040 430 We are sorry, but fundamentals data for the security specified is not available. Not allowed
```

After you fill in the fields to define the contract for which you want to view option chains and click Go, the API makes a call to **reqMktData()**, which you may recall from our previous chapter on requesting market data. Yes, requesting option chains works the same way.

The reqMktData() Method

Let's take another look the parameters to use for requesting market data, including option chains. The EClientSocket **reqMktData()** method header looks like this:

```
public synchronized void reqMktData(int tickerId, Contract
contract, String genericTickList, boolean snapshot) {
.
.
.
}
```

As you can see in the following table, which you've seen before, this method has four parameters:

Parameter	Description
tickerId	The ticker id. Must be a unique value. When the market data returns, it will be identified by this tag. This is also used when canceling the market data.
contract	This class contains attributes used to describe the contract.
genericTicklist	A comma delimited list of generic tick types.
snapshot	Check to return a single snapshot of market data and have the market data subscription cancel. Do not enter any genericTicklist values if you use snapshot.

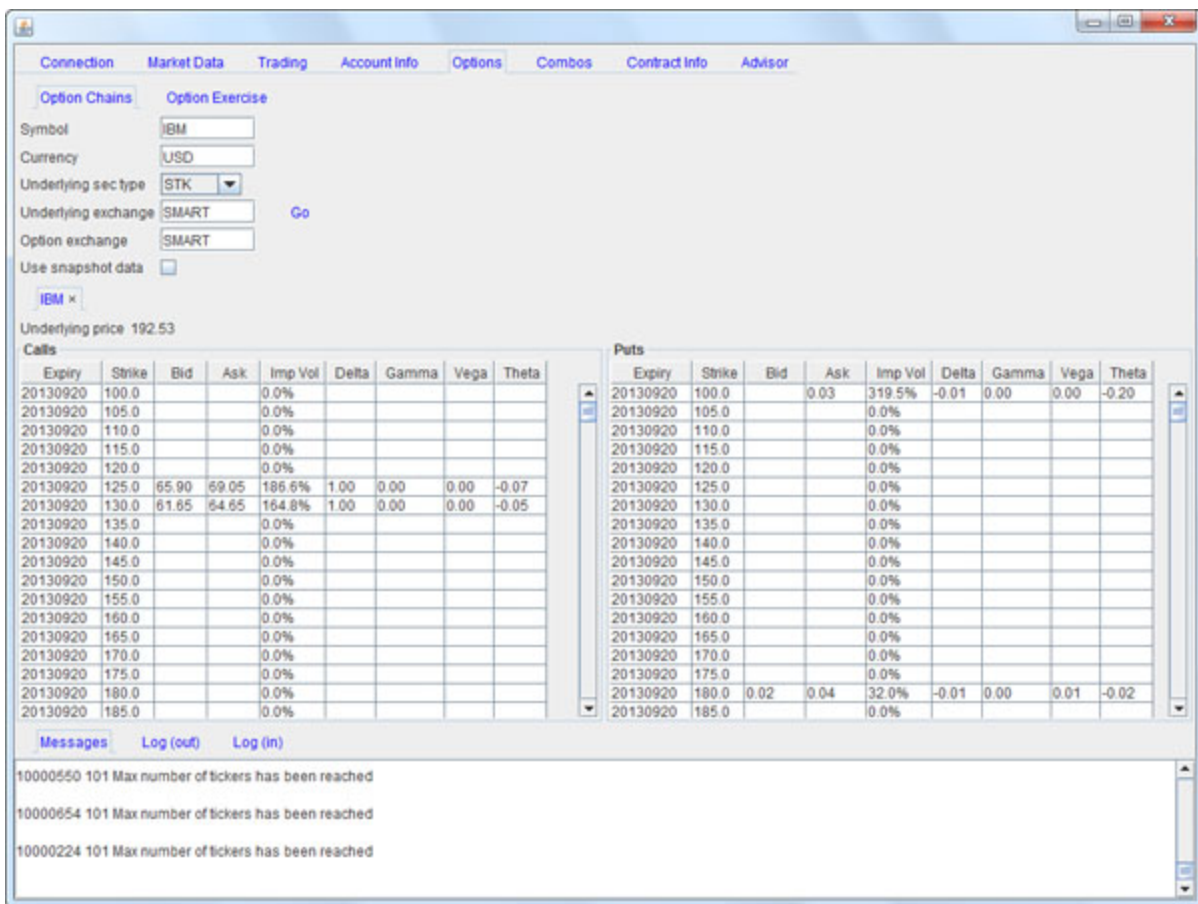
This table is for illustrative purposes only and is not intended to portray valid API documentation.

Just in our previous request to get market data from TWS, our current request for option chains sends the parameters in the **reqMktData()** method. And once again, we pay special attention to the attributes that are sent with the *contract* parameter.



For a complete list of ALL of the attributes in the contract class, see the [API Reference Guide](#).

TWS returns the option chain data, beginning with an expiry of today's date, on a separate tabbed area as shown in the screen below:



EWwrapper Methods that Return Market Data

The Java API EWrapper interface includes all of the methods that return data to the API in response to a method being sent. In the case of market data and option chain data, it is returned from TWS via the following methods in the **EWrapper** interface:

tickPrice()

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

tickSize()

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


tickOptionComputation()

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

tickGeneric()

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

tickString()

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

tickEFP()

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

We've already looked at most of these tick methods in the market data chapter. The pertinent method here is **tickOptionComputation()**, which returns data specific to options. The underlying price, and the implied volume and greek values in the option chains are all delivered by **tickOptionComputation()**.

As we've mentioned before, there's a lot you can do with market data requests, so we won't get into all of that here. You can see all of the tick types in the API Reference Guide [here](#).



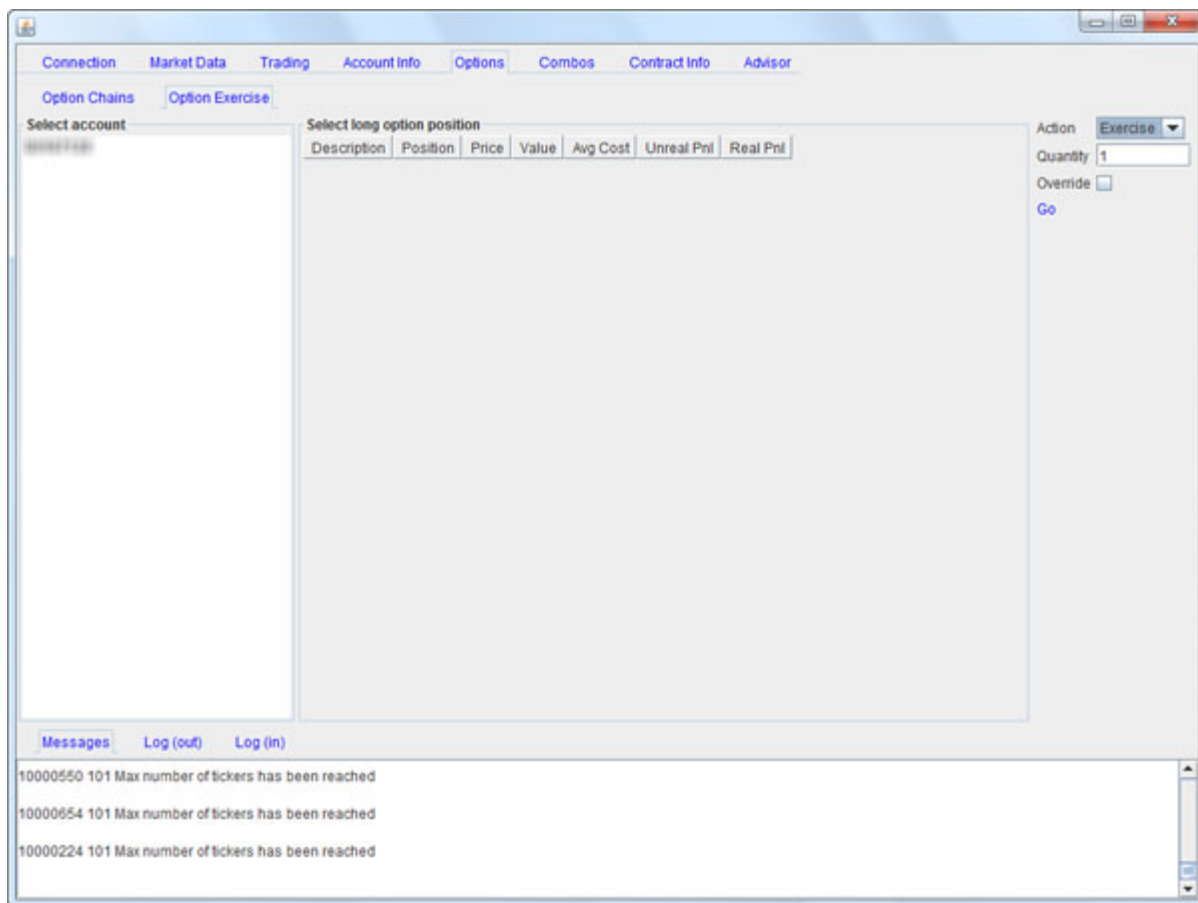
The tick methods are described in the [Java EWrapper Methods](#) section of the API Reference Guide.

Chapter 14: Exercising Options

As long as we're talking about options, let's take a look how the Java Test Client sample application exercises options prior to expiration, and instructs options to lapse. In this chapter, we'll show you the methods and parameters behind the Options Exercise area of the sample application

What Happens When I Exercise an Option or Let an Option Lapse?

In the Java Test Client, clickk the Options tab, then click the Option Exercise tab. You'll see a screen that looks like the one shown below.



To exercise an option, you first select the account on the left (and if you have an individual account, you'll only see your account ID listed). All of your exercisable options are listed in the main panel of the screen. The action takes place in the panel on the right side of the screen, where you select the action (Exercise or Lapse), the quantity of the contracts on which to perform the action, and little check box called **Override**. **Override** specifies whether your setting will override the system's natural action. For example, if your action is "exercise" and the option is not in-the-money, by natural action the option would not exercise. If you have override set to "yes" (you select the check box), the natural action would be overridden and the out-of-the money option would be exercised. When you're ready to submit your request, click the **Go** link.

Now we make a call to the EClientSocket method **exerciseOptions()**, which is shown below, and which sends the selections and values you entered to TWS.

The exerciseOptions() Method

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

And here are the descriptions of all of those parameters, which you can see correspond to your selections and entries on the Option Exercise tab of the sample application. And once again, we see our old friend *contract*, whose many attributes pass information about the contract to TWS.

Parameter	Description
tickerId	The Id for the exercise request
contract	This class contains attributes used to describe the contract.
exerciseAction	This can have two values: <ul style="list-style-type: none"> • 1 = exercise • 2 = lapse
exerciseQuantity	The number of contracts to be exercised or lapsed.
account	For institutional orders. Specifies the IB account.
override	Specifies whether your setting will override the system's natural action. For example, if your action is "exercise" and the option is not in-the-money, by natural action the option would not exercise. If you have override set to "yes" the natural action would be overridden and the out-of-the money option would be exercised. Values are: <ul style="list-style-type: none"> • 0 = do not override • 1 = override

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

When you select *Exercise* as the action, the *exerciseAction* parameter has a value of 1. When you select *Lapse* as the action, *exerciseAction* has a value of 2. In this case, no values are returned by the EWrapper interface, as is the case with many other functions in the TWS Java API.

And that's all there is to it!

Orders and Executions

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

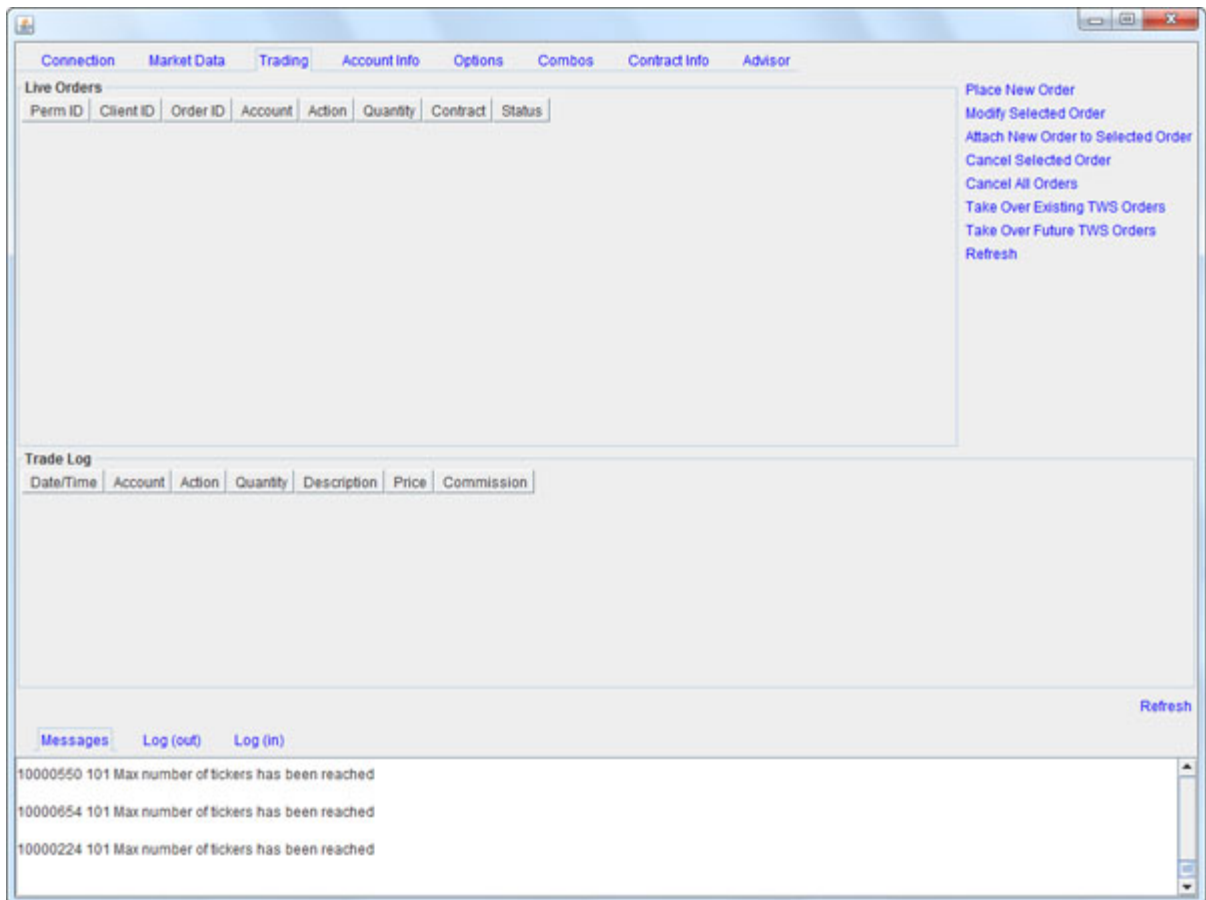
Here's what you'll find in this section:

- [Chapter 15: Placing and Canceling an Order](#)
- [Chapter 16: Extended Order Attributes](#)

Using the Java Test Client 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 15: Placing and Canceling an Order

These next few chapters look at the order-related actions in the Java Test Client sample application, which are grouped together under the Trading tab. Actions such as placing and canceling an order, applying extended order attributes, placing volatility and scale orders, and even IBAIgo orders are included here.



As you can see, all of the order-related actions you can take are represented by links on the right side of the sample application screen. Live orders and completed orders (Trade Log) are also displayed on the Trading tab.

The Order Dialog

First let's take a look at the Order Dialog, which is where you'll do most of your order setup in the Java Test Client. To place an order using the Java Test Client, the first thing you have to do is click the **Place New Order** link. This opens the Order dialog.

The screenshot shows a window titled 'Contract' with several tabs: Contract, Order, Misc, Advisor, Volatility, Scale, and IB Algo. The 'Contract' tab is selected. It contains the following fields and values:

Field	Value
Symbol	IBM
Sec type	STK
Expiry	
Strike	
Put/call	None
Multiplier	
Exchange	SMART
Currency	USD
Local symbol	
Trading class	

At the bottom of the window are three buttons: Transmit Order, Check Margin, and Close.

The Order dialog contains several tabs, each of which contains entry fields and selections that pertain to orders:

- **Contract** - This is where you define the contract you want to trade.
- **Order** - This is where you define the basic order parameters, including action, quantity and order type.
- **Misc** - This tab contains a variety of optional settings that are collectively called "extended order attributes."
- **Advisor** - This tab is for Advisors, who use the entry fields to create orders and allocate shares among multiple clients. We talk about this in the [Java API Getting Started Guide for Advisors](#).
- **Volatility** - This tab contains the fields you need to place a volatility order. These order attributes are also considered to be "extended order attributes." Volatility orders are beyond the scope of this guide, so we won't be discussing them in this chapter.
- **Scale** - This tab contains the fields you need to place a scale order. These order attributes are also considered to be "extended order attributes." Scale orders are beyond the scope of this guide, so we won't be discussing them.
- **IB Algo** - This tab contains a variety of entry fields that are used to place IB Algo orders. These orders are very advanced and so we won't be discussing them.

In addition to the aforementioned tabs, there are also links to Transmit Order, Check Margin and Close (which simply closes the Order dialog).

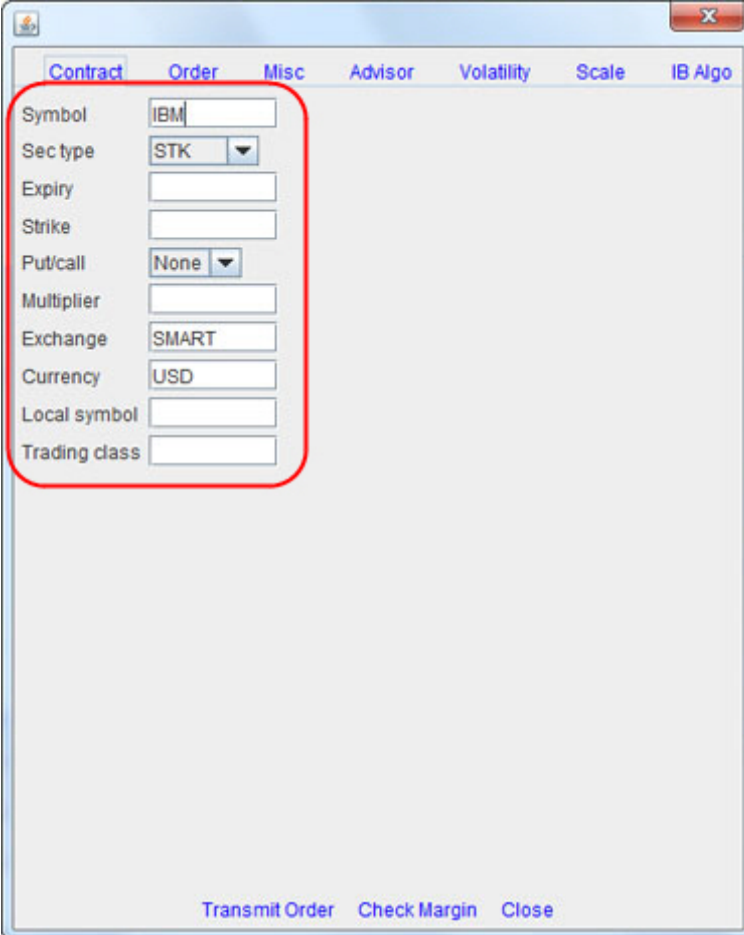
By the way, if you're curious about which part of the Java API contains the code for the Order dialog, it's the **TicketDlg** java class, which is part of the **apidemo** package.

What Happens When I Place an Order?

In this section, we'll show you the methods and parameters responsible for communicating your order to TWS.

Of course, the first thing you have to do when you want to place a new order is click the **Place New Order** link located on the right side of the Order tab in the sample application. The Order dialog opens to the Contract tab.

The image below highlights the fields on the Contract tab that you use to identify the contract you want to trade.



The screenshot shows the 'Contract' tab of the Order dialog. A red box highlights the following fields:

- Symbol: IBM
- Sec type: STK
- Expiry: (empty)
- Strike: (empty)
- Put/call: None
- Multiplier: (empty)
- Exchange: SMART
- Currency: USD
- Local symbol: (empty)
- Trading class: (empty)

At the bottom of the dialog, there are three buttons: Transmit Order, Check Margin, and Close.

The Order tab is where you define the action of the trade (BUY or SELL), the quantity to trade, and the order type, among other things. The image below highlights the fields on the Order tab as well as the **Transmit Order** link.

The screenshot shows the TWS Order tab interface. The 'Order' tab is selected, and a red box highlights the following fields: Account (dropdown), Action (BUY dropdown), Quantity (100 text box), Display size (empty text box), Order type (LMT dropdown), Limit price (1.0 text box), Aux price (empty text box), and Time-in-force (DAY dropdown). At the bottom of the window, a red box highlights the 'Transmit Order' button, with 'Check Margin' and 'Close' buttons to its right.

For simple orders, you're probably only going to fill in the fields on the Contract and Order tabs. So to place an order, first define the contract on the Contract tab, then define the order on the Order tab, then click **Transmit Order**. Simple!

When you click that link, we call the EClientSocket method `placeOrder()`, which sends your entries in all those fields to TWS as its parameters.

The `placeOrder()` Method

The EClientSocket **`placeOrder()`** method header is shown below.

```
public synchronized void placeOrder(NewContract contract, NewOrder
order)
```

Once again, you'll notice that we're using a couple of those Java SocketClient Properties to send contract- and order-related attributes to TWS.

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

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

The *contract* and *order* classes contain parameters that correspond to the Contract and Order tabs in the Order dialog that you filled in. We're already familiar with *contract* but what kinds of things are included in the *order* class? Well, it contains attributes such as the String *m_action*, which is set to BUY, SELL or SSHORT depending on what kind of trade you want to execute; and another String *m_orderType* which, as you might expect, defines the order type (Market, Limit, Stop, and so on). There are many other attributes in the order class, and you can see the full list [here](#). The attributes very neatly correspond to the fields on the Order tab.

In the Java Test Client, the Trading tab displays each of your live, open orders in a table at the top of the screen, and your completed orders in a Trade Log table, as shown below.

Perm ID	Client ID	Order ID	Account	Action	Quantity	Contract	Status
1746377309	0	6	DU167128	BUY	100	IBM STK SMART	Submitted
1746377310	0	7	DU167128	BUY	100	FB STK SMART	Submitted

Date/Time	Account	Action	Quantity	Description	Price	Commission
20130917 13:11:07	DU167128	BOT	100	IBM STK ARCA	192.29	1.0

5 202 Order Canceled - reason:

10000751 101 Max number of tickers has been reached

10000624 101 Max number of tickers has been reached

10000751 300 Can't find Eld with tickerid:10000751

The `orderStatus()` Method

After you place your order, TWS responds by sending back the EWrapper `orderStatus()` method, which is shown below.

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

You probably notice that some of the columns in the Live Orders table in the sample application correspond to the parameters in `orderStatus()`. These include the Order ID, which matches the order status to the correct order; and the Perm ID, which is an internal ID used by TWS to identify the order. `orderStatus()` also returns some other fields including some related to price where applicable, and the current status of your order.

Order statuses include:

- Submitted - This means that your order has been accepted at the order destination and is working in the system.
- Cancelled - This means that the balance of your order has been confirmed canceled by the IB system.
- Filled - This means that your order has been filled.

There are other statuses of course, and you can see these in the [API Reference Guide](#).

You might also have noticed that the other columns in the Live Orders table correspond to a few of the attributes in the *contract* and *order* classes.

`orderStatus()` is also sent by TWS whenever the status of your order changes. So when your order is filled, for example, you'll get another `orderStatus()` from TWS and the Status of that order in the Live Orders table will update dynamically.

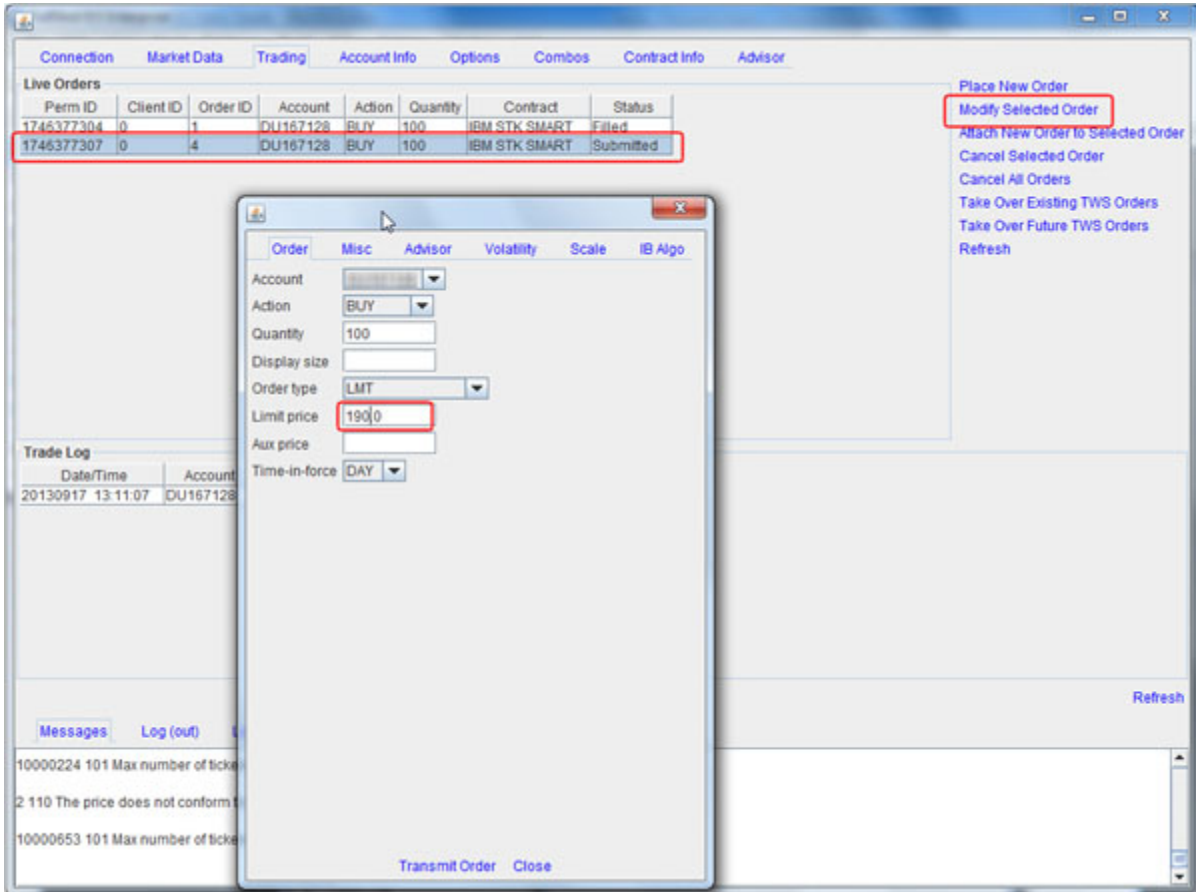
Order IDs

One very important thing to remember when you're placing orders from your own API application is that the order ID number **must be greater than the previously used numbers**. For example, if you place an order with an Order ID of 11, the next order you place should have an Order ID of at least 12. If you forget to do this, your order won't go through.

Modifying an Order

In the Java Test Client, you can modify an existing open order by clicking on an open order displayed in the Live Orders table, then clicking the **Modify Selected Order** link.

As you can see in the image below, when you click that link, the Order dialog opens to the Order tab, which means you can modify any of the available settings on that tab, then transmit the modified order. By the way, in the Java API, you're simply resending the **placeOrder()** method to TWS, which will respond by sending back **orderStatus()**.

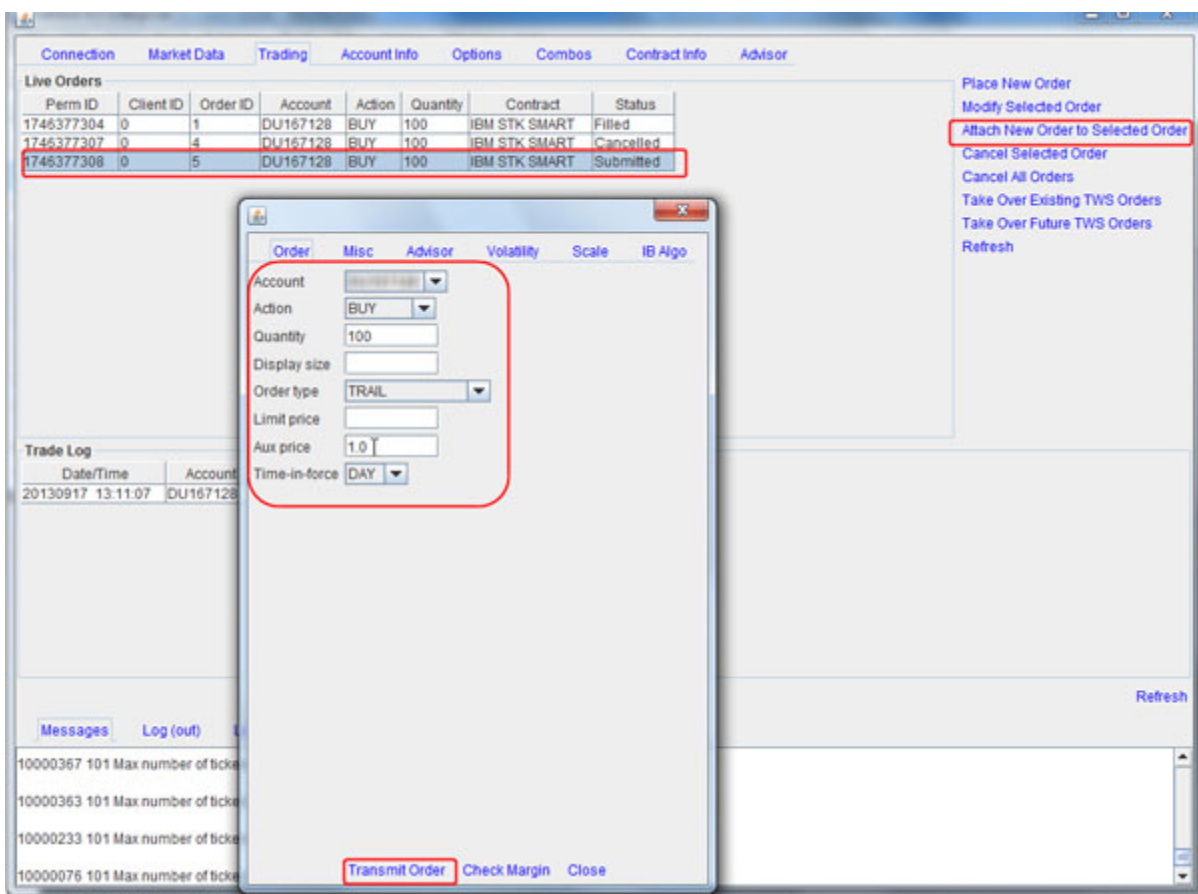


Attaching an Order to an Existing Order

In the Java Test Client, you can modify an existing open order by clicking on an open order displayed in the Live Orders table, then clicking the **Attach New Order to Selected Order** link.

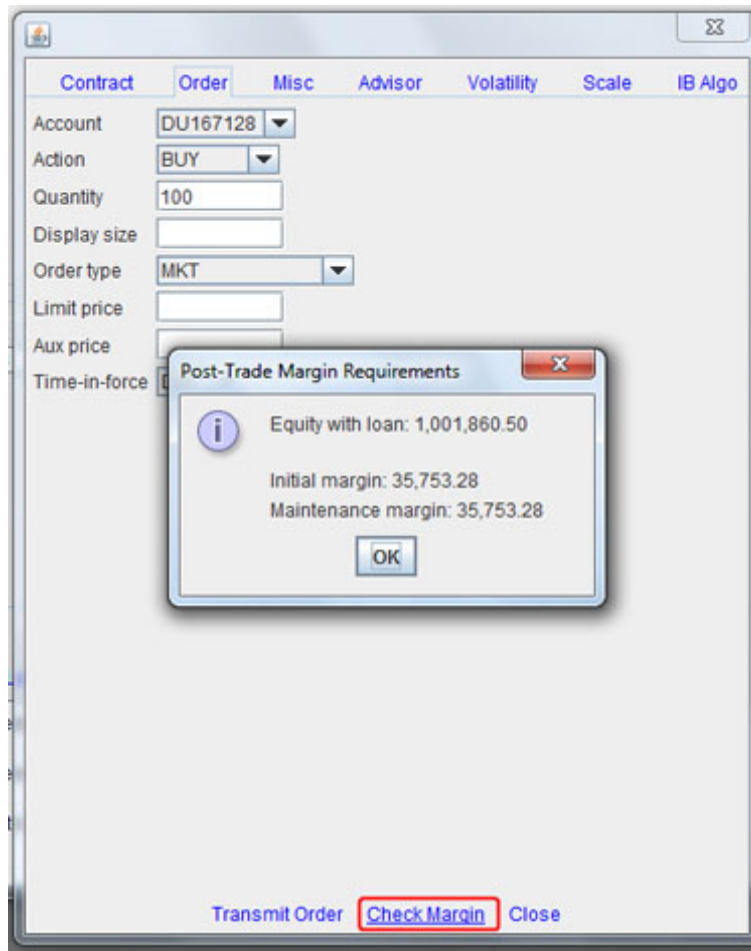
As you can see in the image below when you click that link, the Order dialog opens to the Order tab, which means you can modify any of the available settings on that tab, then transmit the order you want to attach to the selected open order. For example, you might want to attach a Trailing Stop order to an existing order by selecting TRAIL as the order type.

By the way, in the Java API, you're simply sending another **placeOrder()** method to TWS with fresh order parameters, and TWS responds, as always, by sending back **orderStatus()**.



Checking Post-Trade Margin Requirements

There's another interesting feature that we've included in the Java Test Client: Check Margin. If you enter your order parameters in the Order dialog, you'll notice a Check Margin link at the bottom of the dialog. Click this link AFTER you enter all of your order parameters (including contract and order settings on those respective tabs), and you will get a popup that gives you a preview of what your margin requirements will be AFTER your trade is executed. You can see what your Equity with Loan value, initial and maintenance margin values will be if your trade goes through.



When you click the **Check Margin** link, you're actually sending the **placeOrder()** method to TWS with one of the attributes of the order class, the boolean *m_whatIf*, set to true. This attribute tells TWS to return pre-trade margin requirements.

And it's the EWrapper method, **openOrder()**, that returns the margin requirements in one of ITS parameters, another SocketClient Property called *orderState*.

Canceling an Order

You can cancel any order that has not yet been filled. In the Java Test Client, click an open order to select it, then click the **Cancel Selected Order** link.



When you click this link, the API calls the EClientSocket **cancelOrder()** method, and the order associated with the specified ID is canceled. `cancelOrder()` has a single parameter, *id*, which matches the cancel instruction to the correct open order.

The header for the EClientSocket method **cancelOrder()** is shown below.

```
public synchronized void cancelOrder( int id)
```

Once you cancel your order, the status of the order displayed in the Live Orders table on the Trading tab of the sample application changes to Cancelled. This is because your order status changed when you canceled it, and the EWrapper method **orderStatus()** is called whenever your order status changes.

Chapter 16: 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 Java API are actually included in the *order* java class. For ease of use, we have created five separate tabs in the Order dialog in which you can assign values to the extended order attributes. These are shown in the sections that follow.

The important thing to remember is that the entries and selections you make in these tabs are all attributes of the order class, which is sent to TWS as a parameter of **placeOrder()** then you transmit an order.

Misc

This tab contains a variety of optional settings, such as Good after and Good until, Percent Offset, Fill Outside RTH, All-or-none, and many others.

Contract	Order	Misc	Advisor	Volatility	Scale	IB Algo
Order ref						
Min Qty						
Good after						
Good until						
Rule 80A		None				
Trigger method		Default				
Percent Offset						
Trail order stop price						
Trailing percent						
Discretionary amount						
NBBO price cap						
OCA group and type			None			
Hedge type and param		None				
Not held	<input type="checkbox"/>		Override constraints	<input type="checkbox"/>		
Block order	<input type="checkbox"/>		E-trade only	<input type="checkbox"/>		
Sweep-to-fill	<input type="checkbox"/>		Firm quote only	<input type="checkbox"/>		
Hidden	<input type="checkbox"/>		Opt out SMART routing	<input type="checkbox"/>		
Fill outside RTH	<input type="checkbox"/>		Transmit	<input checked="" type="checkbox"/>		
All-or-none	<input type="checkbox"/>					

Transmit Order Check Margin Close

Advisor

This tab is for Advisors, who use the entry fields to create orders and allocate shares among multiple clients. We talk about this in the [Java API Getting Started Guide for Advisors](#).

Volatility

This tab contains the fields you need to place a volatility order. These order attributes are also considered to be “extended order attributes.” Volatility orders are beyond the scope of this guide, so we won’t be discussing them in this chapter except to point out that the parameters on this tab can be found in the *order* class. For example, the Hedge order type drop-down pictured below allows you to select an order type to instruct TWS to submit a delta neutral trade on full or partial execution of the VOL order. This field corresponds to a String attribute *m_deltaNeutralOrderType* attribute in the *order* class.

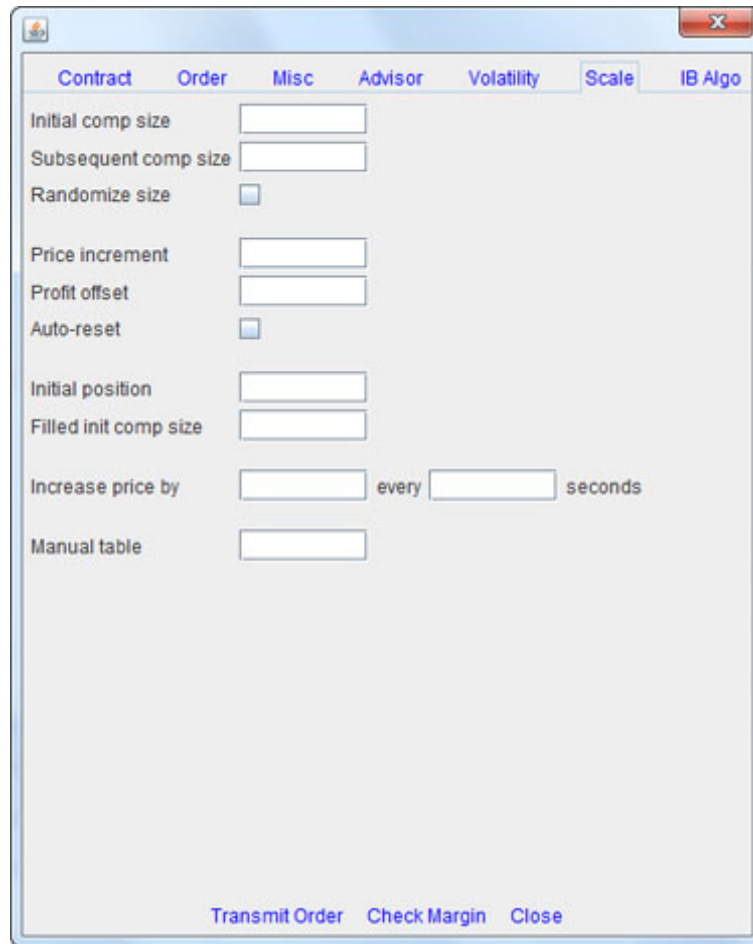
The screenshot shows a window titled "Volatility" with several tabs: Contract, Order, Misc, Advisor, Volatility (selected), Scale, and IB Algo. The Volatility tab contains the following fields:

- Volatility: A text input field followed by a dropdown menu showing "None".
- Continuously update price: A checkbox.
- Option reference price: A dropdown menu showing "None".
- Hedge order type: A dropdown menu showing "None".
- Hedge order aux price: A text input field.
- Hedge contract conid: A text input field.
- Stock range - upper: A text input field.
- Stock range - lower: A text input field.

At the bottom of the window, there are three buttons: "Transmit Order", "Check Margin", and "Close".

Scale

This tab contains the fields you need to place a scale order. These order attributes are also considered to be “extended order attributes.” Scale orders are beyond the scope of this guide, so we won’t be discussing them except to point out that the parameters on this tab can be found in the *order* class. For example, the Auto-reset check box pictured below corresponds to a boolean attribute *m_scaleAutoReset* in the *order* class.



The screenshot shows a window titled "Scale" with several tabs: Contract, Order, Misc, Advisor, Volatility, Scale (selected), and IB Algo. The Scale tab contains the following fields and controls:

- Initial comp size:
- Subsequent comp size:
- Randomize size: ☐
- Price increment:
- Profit offset:
- Auto-reset: ☐
- Initial position:
- Filled init comp size:
- Increase price by: every seconds
- Manual table:

At the bottom of the window, there are three buttons: Transmit Order, Check Margin, and Close.



For a complete list of ALL of the attributes in the order class, see the [API Reference Guide](#).

IB Algo

This tab contains a variety of entry fields that are used to place IB Algo orders. These orders are very advanced and so we won't be discussing them in too much detail. However, we would like to introduce you briefly to two attributes that are important to IB Algo orders.

If you look at all of the attributes of the *order* class, you will notice that many of the IBAIgo parameters included on the IB Algo tab pictured below are *not* included as specifically named attributes. There are two String attributes in the *order* class that support IB Algo orders:

- *m_algoStrategy*, which contains the actual name of the IB Algo order type (called a "strategy," some of these include AD for Accumulate/Distribute, PctVol for Percentage of Volume, and so on).
- *m_algoParams*, which is essentially a container for IB Algo order parameters and their values in the form:

```
m_algoParams.add( new TagValue("maxPctVol","0.01") );
```

So the entry fields on the IB Algo tab in the Order dialog that you can't find in the order class are created using the *m_algoParams* attribute in the *order* class.

The screenshot shows a window titled "Order" with several tabs: "Contract", "Order", "Misc", "Advisor", "Volatility", "Scale", and "IB Algo". The "IB Algo" tab is selected. It contains a list of fields for configuring an IB Algo order. The "Algo strategy" field is a dropdown menu currently set to "None". The other fields are text input boxes. At the bottom of the window, there are three buttons: "Transmit Order", "Check Margin", and "Close".

Field	Value
Algo strategy	None
startTime	
endTime	
allowPastEndTime	
maxPctVol	
pctVol	
strategyType	
noTakeLiq	
riskAversion	
forceCompletion	
displaySize	
getDone	
noTradeAhead	
useOddLots	
componentSize	
timeBetweenOrders	
randomizeTime20	
randomizeSize55	
giveUp	
catchUp	
waitForFill	



You can read more about programming IB Algo orders, including a list of accepted values for the `m_algoParams` in the [API Reference Guide](#).

This concludes our brief introduction to extended order attributes. In the next chapter, you'll learn how to retrieve all your account and portfolio information from TWS.

Account and Portfolio Information

This chapter describes how to view all of your account and portfolio information in the Java Test Client.

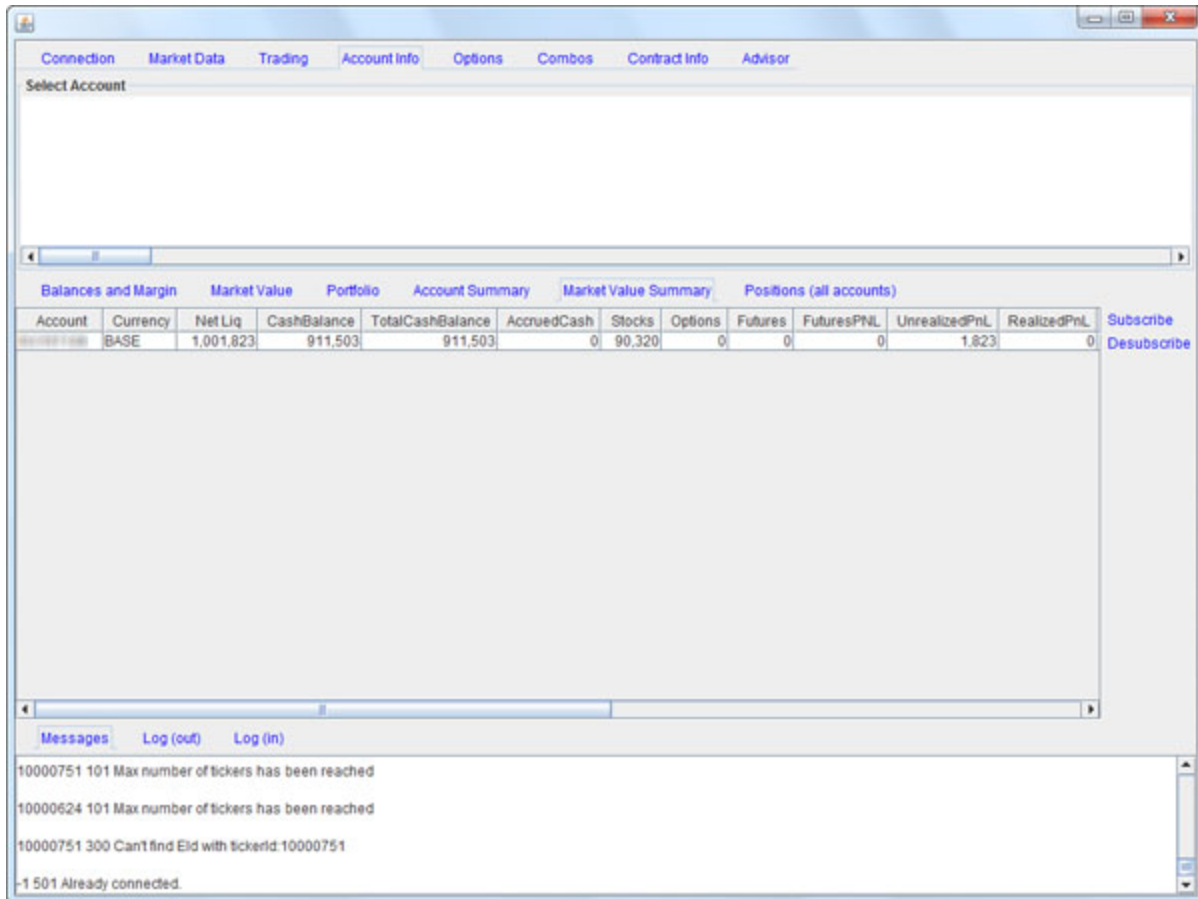
- [Chapter 17: Retrieving Account and Portfolio Information](#)



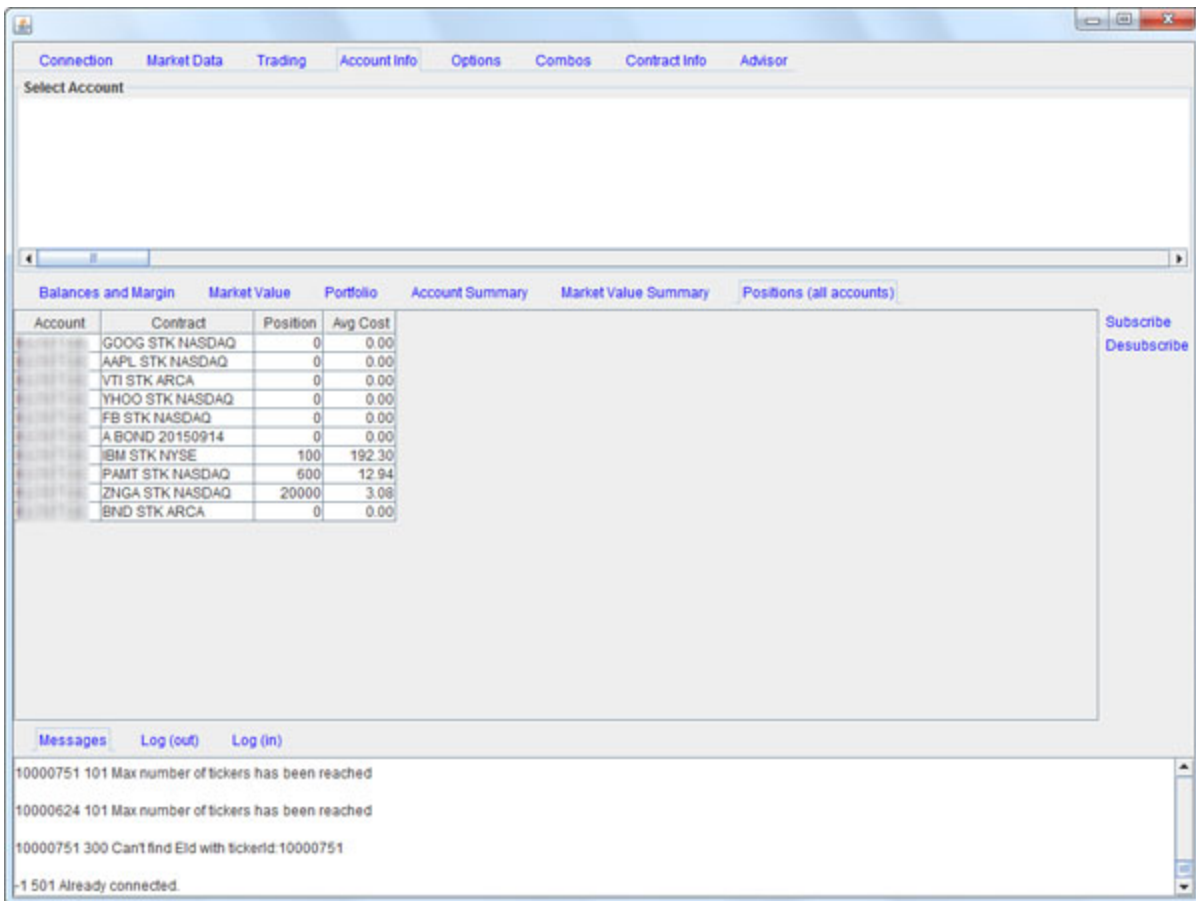
In addition to the tasks described in this chapter, the Java API sample application also includes a few more advanced functions, including the ability to place spread, EFP and Delta-Neutral orders. For more information on these and other advanced capabilities of the Java API, see our [API Reference Guide](#), available from the Documentation page in our Traders' University.

Chapter 17: Retrieving Account and Portfolio Information

Retrieving your account information from TWS is easy in the Java Test Client. In fact, we've provided a dedicated tab for just that purpose! It's called Account Info. The Account Info tab contains several tabs, each of which displays a different set of data from your account. By the way, this is a lot of the same information that you can view in the Account Window in TWS.



To retrieve your Account Information, select your Account ID from the top part of the screen (and if you only have a single, individual account, you'll only see one Account ID listed), then click the **Subscribe** link located on the right side of the screen. The various tabs on the Account Info screen will fill up with your account and portfolio information from TWS. The following image shows the Portfolio



What Happens When I Retrieve My Account Information?

When you click the Subscribe link, the API sends the EClientSocket method **reqAccountUpdates()** to TWS.

The reqAccountUpdates() Method

```
public synchronized void reqAccountUpdates(boolean subscribe,  
String acctCode)
```

As you can see, there are only two parameters in this method. *subscribe* is a boolean that, when set to true, tells TWS to send the account information along. The other parameter sends the account code to TWS so that it knows the correct account.

The account information is provided in three different EWrapper methods:

- **updateAccountTime()**
- **updateAccountValue()**
- **updatePortfolio()**

updateAccountTime() has a single parameter, the String *timeStamp*, which indicates the last time the account information was updated.

updateAccountValue() sends the actual account values that you see on the different tabs of the Account Info screen in the sample application.

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

As you can see from the list of parameters below, the actual account values are sent by the *key*, a String that indicates each type of account value; and *value*, another String that assigns the correct value to the account value type.

Parameter	Description
key	A string that indicates one type of account value. There is a long list of possible keys that can be sent, here are just a few examples: <ul style="list-style-type: none"> • CashBalance - account cash balance • DayTradesRemaining - number of day trades left • EquityWithLoanValue - equity with Loan Value • InitMarginReq - current initial margin requirement • MaintMarginReq - current maintenance margin • NetLiquidation - net liquidation value
value	The value associated with the key.
currency	String that defines the currency type, in case the value is a currency type.
account	String that indicates the account.

updatePortfolio() is responsible for sending your current portfolio information from TWS.

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

As you can see from the method header shown above, there are several parameters in **updatePortfolio()**, including our old friend *contract*, which sends the information about each contract in your portfolio, as well as other numerical values.

Desubscribing

When you trade, your account information and portfolio will change of course, and as long as the Java Test Client is still connected to TWS and you are still “subscribed” to account information, your account info will update in the sample application.

If you no longer want to receive all that account and portfolio information, you can click the **Desubscribe** link located in the same place as the Subscribe link, and you won’t get any more account information updates from TWS.

This concludes our discussion of the Java Test Client sample application for individual accounts. The next chapter describes the Java API examples that you can download from our website.

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 Java API can do, and how to make it do some of the things you want. Now we give you a bit more information about where to find additional helpful outside resources you can use to help you move forward.

This section contains the following chapters:

- [Chapter 25 - Additional Resources](#)

Chapter 25 - 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 Java Programming

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

The best place to go to find additional help with all things Java is the Oracle web site. Just type <http://www.oracle.com/technetwork/java/index.html> in your browser's address line and check out the list of links on that page. There are many online resources available for Java programmers, including documentation, tutorials, and code samples.

If you simply want to look up information about the actual Java API (as opposed to our TWS Java API), you can go directly to Sun's [API Specification page](#). There you will find links to the javadocs for all of Sun's different API versions.

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

Help with the Java API

For help specific to the Java TWS 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 in your browser's address line. Now that you're there, let me tell you where you can go.



As of this writing in October 2013, 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 *Documentation* from the **Education** menu, then click the **Users' Guides** link in the menu on the left, then click the **Application Programming Interface (API)**. Click **Online API Reference Guide**, then click the Online button to open the online API Reference Guide.

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 *Documentation* from the **Education** menu, then click the **Application Programming Interface** link and click either the **Production Notes** or **Beta Notes** buttons to view those release notes.

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 Bulletin Board

You can trade ideas and send out pleas for help via the IB customer base accessible through both the IB Bulletin Board. 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.

To view or participate in the IB Bulletin Board, go to the **Education** menu and click *Bulletin Boards*. Click the **Launch IB Discussion Forum** button to access all of our bulletin boards, including the TWS API bulletin board.

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 **Products & Services**, 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.

Index

A

- account information 7-94, 7-95
 - desubscribing from 7-96
- additional resources 8-100
- API
 - reasons for using 2-18
- API beta notes 8-101
- API connection to TWS
 - enabling 4-35
- API Reference Guide 8-100
- API release notes 8-101
- API software
 - downloading 3-23
 - installing 3-26
- API support email 8-101
- API webinars 8-101
- apidemo 4-33
- apidemo.util 4-33
- attaching orders to existing orders 6-85

C

- cancelHistoricalData() 4-57, 4-60
- canceling historical data 4-53, 4-57
- canceling market data 4-40, 4-47
- canceling market depth 4-49, 4-52
- canceling market scanner subscriptions 4-61
- canceling orders 6-78, 6-87
- canceling real time bars 4-58, 4-60
- cancelMktData() 4-48
- cancelMktDepth() 4-52
- cancelOrder() 6-87
- Check Margin 6-86
- com.ib.client 4-33
- Connect 4-36
- connecting the Java Test client to TWS 4-32
- connecting to TWS 4-36
- contract data 4-65
- contractDetails() 4-67
- customer forums 8-101
- customer service 8-101

D

- DDE for Excel API
 - additional resources 8-100
 - preparing to use 3-21
- deep market data 4-50
- Disconnect button 4-38
- disconnecting from TWS 4-38
- document conventions 1-11

- downloading API software 3-23

E

- EClientSocket constructor 1-11
- edisonconnect() 4-39
- enable API connection in TWS 4-35
- exercise an option 5-74
- exerciseOptions() 5-75
- exerciseOptions() parameters 5-75
- exercising options 5-74
- Extended order attributes 6-88

F

- Features Poll 8-101
- footnotes and references 1-9
- framework of Java Test Client 4-32
- frozen market data 4-46

H

- Historical Data 4-54
- historical data 4-53
- historicalData() 4-56
- how to use this book 1-8

I

- IB bulletin boards 8-101
- IB Customer Service 8-101
- icons used in this book 1-10
- installing API software 3-26
- introduction 1-7, 4-31, 5-69, 6-77, 7-93

J

- J2SE Development Kit and NetBeans IDE Bundle 3-22
- Java API, help with 8-100
- Java IDE, downloading 3-22
- Java JDK, downloading 3-22
- Java programming help 8-100
- Java Test Client
 - connecting to TWS 4-32
 - framework 4-32
- Java Test Client main window 4-32

L

- let an option lapse 5-74
- logging in to TWS 4-33

M

- margin requirements after a trade 6-86
- market data 4-40

- canceling 4-47
- EWrapper methods 4-44, 5-72
- frozen 4-46
- snapshot 4-47
- market data returned 4-44
- Market Depth 4-50
- market depth 4-49
- market scanners 4-61, 4-62
- mktDataType() 4-46
- modifying orders 6-84

N

- NetBeans 3-22

O

- option chains 5-70
- options 5-74
 - exercising 5-74
- Order dialog 6-78
- order IDs 6-83
- orders 6-78
 - attaching orders 6-85
 - modifying 6-84
- orderStatus() 6-83
- organization of this book 1-8
- override option 5-74

P

- packages in the Java API 4-33
- Place Order button 6-80
- placeOrder() 6-81
- placeOrder() parameters 6-82
- placing orders 6-78
- preparing to use the DDE for Excel API 3-21

R

- real time bars 4-58
- real-time account monitoring, in TWS 2-17
- realtimeBar() 4-60
- reasons for using an API 2-18
- reqAccountUpdates() 7-95
- reqContractDetails() 4-66
- reqHistoricalData() 4-54
- reqMktData() 4-42, 5-71
- reqMktData() parameters 4-42, 5-71, 7-96
- reqMktDepth() 4-50
- reqMktDepth() parameters 4-50, 4-54, 4-59, 4-62
- reqRealTimeBars() 4-59

- reqScannerParameters() 4-64
- reqScannerSubscription() 4-62
- request option chains 5-70
- Request real-time bars 4-59
- Request Top Market Data 4-41
- requesting contract data 4-65
- requesting historical data 4-53
- requesting market data 4-40, 4-41
- requesting market depth 4-49
- requesting real time bars 4-58
- resources, for Java programming help 8-100
- retrieving account information 7-95

S

- Sample dialog
 - market data fields 4-43
- scannerData() 4-63
- scannerDataEnd() 4-63
- snapshot 4-47
- subscribing to market scanner subscriptions 4-61
- subscribing to market scanners 4-62

T

- tickEFP() 4-45, 5-73
- tickGeneric() 4-45, 5-73
- tickOptionComputation() 4-45, 5-73
- tickPrice() 4-44, 5-72

- tickSize() 4-44, 5-72
- tickString() 4-45, 5-73
- Trader Workstation
 - overview 2-14
- trading window 2-16
- TWS
 - logging in 4-33
 - real-time account monitoring in 2-17
- TWS and the API 2-18
- TWS login box options 4-35
- TWS Order Ticket 2-16
- TWS overview 2-14, 2-16
- TWS Quote Monitor 2-16

U

- updateAccountTime() 7-95
- updateAccountValue() 7-96
- updateMktDepth() 4-52
- updateMktDepthL2() 4-52
- updatePortfolio() 7-96
- using this book 1-8
 - document conventions 1-11
 - icons 1-10
 - organization 1-8

V

- viewing option chains 5-70