

Getting Started with the TWS C++ API Guide for Advisors

Download the software to your PC and the desktop version allows you to access your account through an internet browser, and is always available. The desktop version uses less memory and may run faster, but it offers more new features. To download to your PC, click on the Desktop version.

Interactive Brokers
Select Trader Workstation

Click on the
username and



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

Getting Started with the TWS C++ API
September 2014
Supports TWS API Release 9.71

© 2014 Interactive Brokers LLC. All rights reserved.

Sun, Sun Microsystems, the Sun Logo and Java are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. Excel, Windows and Visual Basic (VB) are trademarks or registered trademarks of the Microsoft Corporation in the United States and/or in other countries. TWS Javahelp version 013, March 25, 2008.

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

Contents

1 Introduction	7
How to Use this Book	8
Organization	8
Chapter 1: Introduction.....	8
Chapter 2: Introducing the TWS C++ API.....	8
Chapter 3: Preparing to Use the TWS C++ API.....	9
Chapter 4: Financial Advisors.....	9
Chapter 5: Allocation Methods and Profiles.....	9
Chapter 6: Where to Go from Here	9
Footnotes and References	9
Icons	10
Document Conventions.....	11
2 TWS and the C++ API.....	13
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
Why Use the TWS C++ API?	18
TWS and the API	18
Available API Technologies	19
An Example	19
3 Preparing to Use the C++ API	21
Install an IDE.....	22
Download the API Software.....	23
Connect to the C++ Sample Application.....	27
Running the C++ API Sample Application.....	27
Running the C++ API Sample Application from Visual Studio 2008	28

In case of errors:	29
What's Next	30
4 The C++ API for Financial Advisors	31
Viewing Account Data for Managed Accounts	32
Viewing Account Data for a Managed Account	32
OnReqAccountUpdate()	34
The reqAccountUpdates() Method	35
C++ EWrapper Functions that Return Account Data	35
The updatePortfolio() function returns the portfolio data for the specified account code. 36	
Viewing a List of Managed Accounts	38
Viewing a List of Managed Accounts	38
OnReqAccts()	39
The reqManagedAccts() Method.....	39
C++ EWrapper Functions that Return a List of Managed Accounts.....	39
Viewing and Modifying FA Configuration Information.....	40
What Happens When I Click the Financial Advisor Button?	40
OnFinancialAdvisor().....	41
The requestFA() Method.....	41
C++ EWrapper Functions that Return Financial Advisor Configuration Information	41
The replaceFA() Method	42
Placing Orders Using FA Accounts	43
Allocation Methods for Account Groups	45
EqualQuantity Method.....	45
NetLiq Method	45
AvailableEquity Method	45
PctChange Method.....	45
Allocation Profiles	46
Percentages.....	46
Ratios	46
Shares	46
In Summary	47
Allocation Methods for Account Groups	50
EqualQuantity Method	50
NetLiq Method	50

AvailableEquity Method..... 50
 PctChange Method 51
 Allocation Profiles 52
 Percentages 52
 Ratios 52
 Shares 52

5 Where To Go From Here 53

Linking to TWS using the TWS C++ API..... 54
 Additional Resources 59
 Help with Microsoft Visual Studio and C++ Programming 59
 Help with the TWS C++ API 59
 The API Reference Guide 59
 The API Beta and API Production Release Notes..... 59
 The TWS API Webinars..... 60
 API Customer Forums 60
 IB Customer Service 60
 IB Features Poll..... 60

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 C++ 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.



This book assumes that you are a Financial Advisor. To learn how to get started using the basic trading features of the TWS ActiveX API, see the [Getting Started with the TWS ActiveX API guide](#).

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

How to Use this Book

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

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

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



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



Before you read any further, we need to tell you that this book focuses on the TWS side of the C++ API - we don't really help you to learn C++. If you aren't a fairly proficient C++ 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 C++ programming book, and come back to us when you're comfortable with the language.

Organization

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

Chapter 1: Introduction

This chapter (the one you're currently reading) tells you what you'll find in this guide and how to use it.

Chapter 2: Introducing the TWS C++ API

This chapter helps 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 C++ API have to offer me?" and even "What other API choices do I have?"

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

Chapter 3: Preparing to Use the TWS C++ API

Chapter 3 walks you through the different things you'll need to do before your API application can effectively communicate with TWS. We'll help you download and install the API software, configure TWS, and get the sample application up and running. A lot of this information is very important when you first get started, but once it's done, well, it's done, and you most likely won't need much from this section once you've completed it.

Chapter 4: Financial Advisors

This chapter describes how to use the C++ API sample application to view account data for managed accounts, view a list of managed accounts, and view and modify the FA groups, FA methods and allocation profiles that make up your FA configuration.

Chapter 5: Allocation Methods and Profiles

This chapter provides a handy reference for advisors who need to refresh their memory about the different allocation methods and profiles that can be set up in Trader Workstation.

Chapter 6: Where to Go from Here

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

Footnotes and References

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

Icons



TWS-Related

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.



C++ Tip

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



Important!

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



Take a Peek!

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



Go Outside!

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

Document Conventions

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

Convention	Description	Examples
Bold	Indicates: <ul style="list-style-type: none"> • menus • screens • windows • dialogs • buttons • tabs • keys you press • names of classes and methods 	When you click the Req Mkt Data button... Press Ctrl+C to copy...
<i>Italics</i>	Indicates: <ul style="list-style-type: none"> • commands in a menu • objects on the screen, such as text fields, check boxes, and drop-down lists 	To access the users' guide, under the Software menu, select <i>Trader Workstation</i> , then click <i>Users' Guide</i> .
Code samples	Code samples appear gray boxes throughout the book.	See below.
<pre style="border: 1px solid gray; padding: 5px; background-color: #f0f0f0;">m_pClient->reqMktData(m_dlgOrder->m_ m_dlgOrder->m_genericTicks, m_</pre>		



Introduction

How to Use this Book

TWS and the C++ 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 C++ API can do for you!

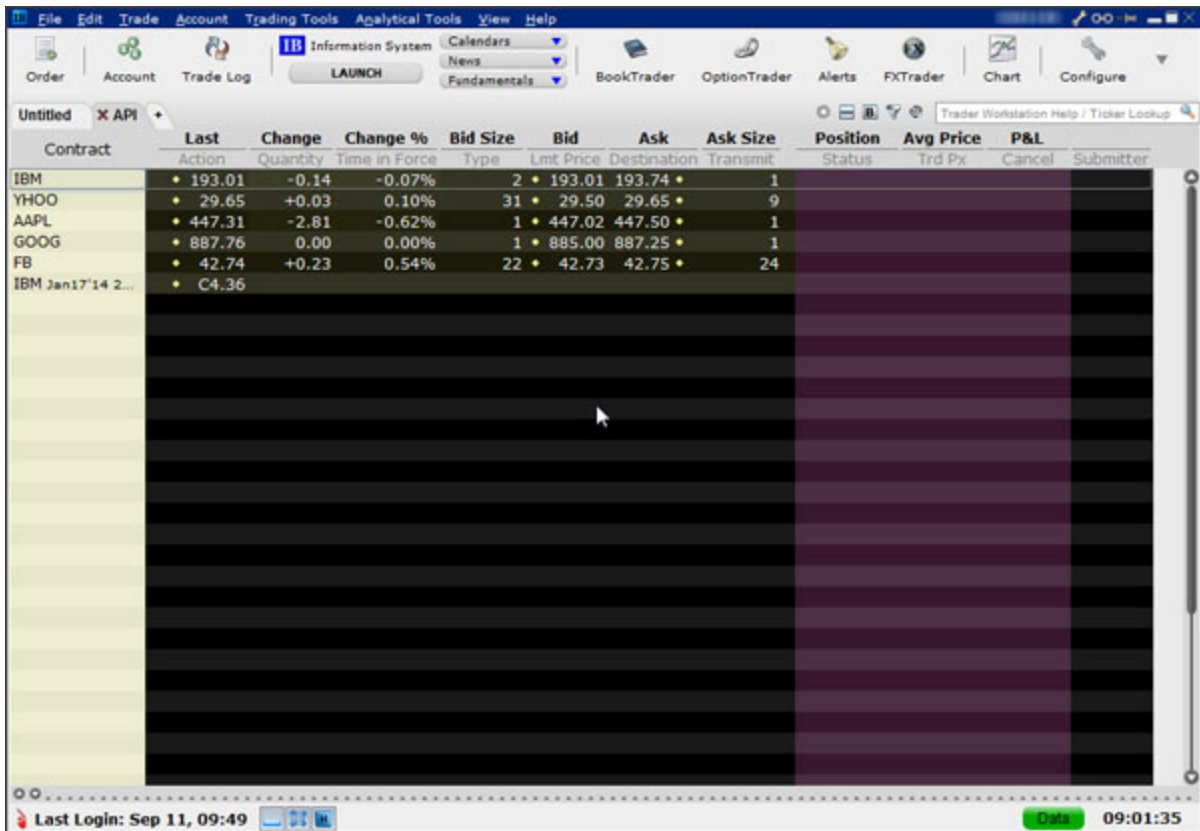
Here's what you'll find in this chapter:

- [What is Trader Workstation?](#)
- [Why Use the TWS C++ API?](#)

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.




The screenshot displays the TWS interface with a spreadsheet view of market data. The table includes columns for Contract, Last, Change, Change %, Bid Size, Bid, Ask, Ask Size, Position, Avg Price, and P&L. The data is as follows:

Contract	Last	Change	Change %	Bid Size	Bid	Ask	Ask Size	Position	Avg Price	P&L
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.

Why Use the TWS C++ 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 "HmMMM, 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 C++ version. Why would you use C++ over the other API technologies? The main reason might be that you are a C++ expert. If you don't know C++ 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 C++, 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 [Application Programming Interfaces](#) page on our web site.

An Example

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

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

The screenshot shows the 'Account' window in TWS, displaying several sections of account information:

- Balances:** A table with columns for Parameter, Total, IB-US Securi..., IB-US Comm..., and IB-UK Regul... All values are 0 USD.
- Margin Requirements:** A table with columns for Parameter, Total, IB-US Securi..., IB-US Comm..., and IB-UK Regul... Values are 0 USD, with some checkboxes.
- Available for Trading:** A table with columns for Parameter, Total, IB-US Securi..., IB-US Comm..., and IB-UK Regul... Values are 0 USD, with some checkboxes and a leverage value of 0.00.
- Market Value - Real FX Balance:** A table with columns for Currency, Total Cash, FX Cash, Stock, Options, Futures, FOPs, Nt Lqdn VI, Unrlzsd P&L, and Rlzd P&L. All values are 0.
- FX Portfolio - Virtual FX Position:** A section with a filter input and a 'More options' button.
- Portfolio:** A section with a filter input and a 'More options' button.

At the bottom, it says 'Last updated at 09:05'.

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

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

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

Preparing to Use the C++ 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 chapter:

- [Install an IDE](#)
- [Download the API Software](#)
- [Connect to the C++ Sample Application](#)



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

Install an IDE

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



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

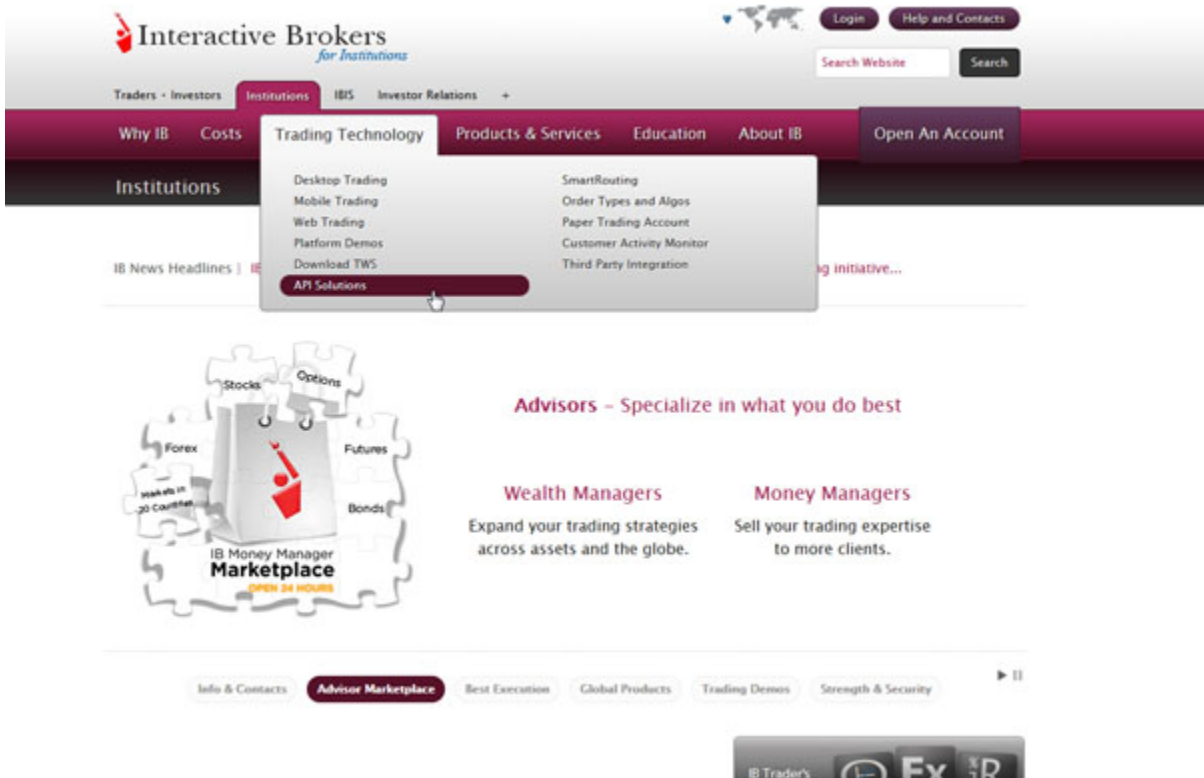
Once you have your C++ development environment installed, you can go to the IB website and download the TWS API software.

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.

Our proprietary API solutions let you create your own automated rule-based trading system that takes advantage of our high-speed order routing and broad market depth.

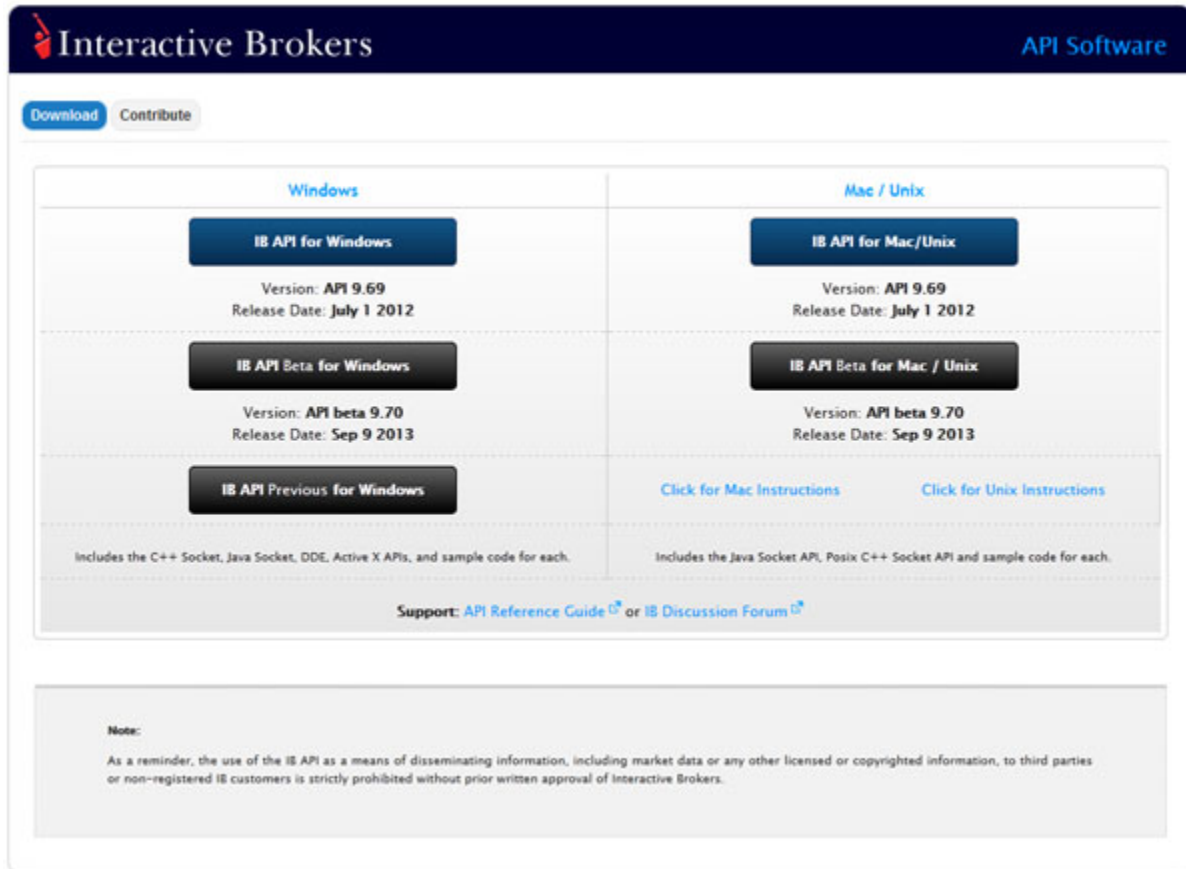
The screenshot shows a navigation menu on the left with four buttons: 'API Software' (highlighted with a red box), 'IB Gateway Software', 'Guides', and 'Release Notes'. In the center, a blue double-headed arrow labeled 'API' connects the menu to a screenshot of the TWS Order Management window. The TWS window displays a table of orders with columns for Underlying, Exchange, Description, Bid Price, Ask Price, Last Price, and Change. Below the TWS window is the Gateway logo.

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.




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 C++ sample application, connect to TWS, and get started using the sample application!

Connect to the C++ Sample Application

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

Although the API provides great flexibility in implementing your automated trading ideas, all of its functionality runs through TWS. This means that you must have a TWS account with IB, and you must have TWS running in order for the API to work. This section describes how to enable TWS to connect to the C++ 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 \$100,000 in phantom cash to play with.

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!

Running the C++ API Sample Application

Here's how to connect to the Visual Basic sample application:

Step 1: Log into TWS.

OK, log into TWS, or run the Demo available on the [Demo](#) tab of the Trader Workstation page on our website.

Step 2: Enable TWS to support the C++ API.

Now look up at the top of the trading window, and you'll see the menu bar. Click the Edit menu, and then click *Global Configuration*. In the Configuration window, click *API* in the left pane, then click *Settings*, which reveals several options on the right side of the window. Check the *Enable ActiveX and Socket Clients* check box and click OK.

Step 3: Run the C++ Sample Application.

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

Running the C++ API Sample Application from Visual Studio 2008

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

Step 1: Log into TWS.

This step is the same for the VB.NET sample as it was for the Visual Basic sample; we're including it here so you don't have to turn the page.

OK, log into TWS, or run the Demo available on the **Demo** tab of the Trader Workstation page on our website.

Step 2: Enable TWS to support the C++ API.

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

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

Step 3: Run the C++ API Sample Application.



These instructions assume that you are using Microsoft Visual Studio 2008. Our C++ API sample application code was developed as a C++6 project. If you are using a different C++ development environment, adjust the instructions below accordingly.

Here's how to do this:

- 1** Create the folder where all project related files will be located. For example, *C:\SampleSocketClient*.
- 2** Copy the folders *TestSocketClient*, *Shared* and *SocketClient* from the TWS API installation folder into the folder you created in Step 2. (For example, *C:\SampleSocketClient*.)
- 3** Open Microsoft Visual Studio, then select **New > Project From Existing Code** from the File menu.
- 4** Select **Visual C++** as the project type.
- 5** In the Create New Project from Existing Code Files dialog:
 - a** For the Project File location, select the folder you created in Step 2. (For example, *C:\SampleSocketClient*.)
 - b** For the Project Name, type **SampleSocketClient**.
 - c** Click **Finish**.
- 6** Right-click the project in the Solution Explorer and select **Properties**. In the Property Pages dialog:

- a On left side of the dialog, select **Configuration Parameters > C/C++ > General**. In the Additional Include Directories field on the right side of the dialog, type:

./Shared;./SocketClient/src

- b On left side of the dialog, select **Configuration Parameters > C/C++ > Code Generation**. In the Runtime Library field on the right side of the dialog, select **Multi-threaded (/MT)**.

7 Click **OK** to save your changes to the project properties.

8 Build the project.

In case of errors:

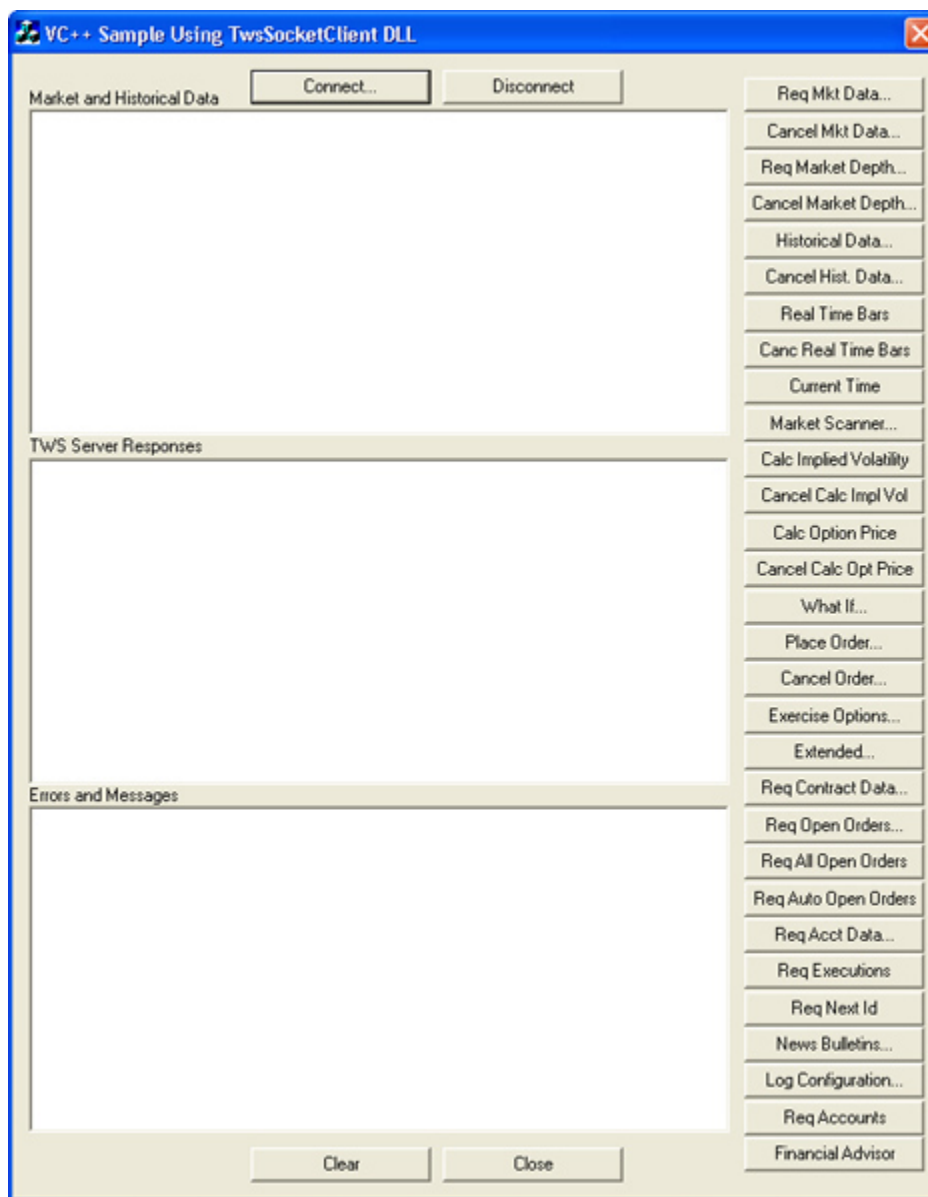
If the C++ sample application won't build or run due to errors, try the following steps:

- 1 In the Visual Studio 2008 Solution Explorer, remove the following files from the project (select the item from the list and press **Delete**):
 - Remove *DlgShareAllocation.h* from the list of Header Files.
 - Remove *DlgShareAllocation.cpp* from the list of Source Files.
 - Remove *TwsSocketClient.lib* from the project.
- 2 In the Solution Explorer, add the following existing items from the *TestSocketClient* folder in your TWS API installation folder (right-click the Header Files/Source Files folder, then select **Add > Existing Item**):
 - Add *DlgUnderComp.h* to the list of Header Files.
 - Add *DlgUnderComp.cpp* to the list of Source Files.



*If you are running TWS API Version 9.6 or higher, you must also add *DlgAlgoParams.h* to the list of Header Files and *DlgAlgoParams.cpp* to the list of Source Files in the project.*

- 3 In the Solution Explorer, add the following existing items from the *SocketClient/src* folder in your TWS API installation folder to the list of Source Files (right-click the Source Files folder, then select **Add > Existing Item**):
 - *EClientSocket.cpp*
 - *MySocket.cpp*
- 4 Press **F5** to run the sample client, which is pictured on the next page.



What's Next

You're now ready to start looking at how our C++ sample application uses our TWS C++ API. Part 3 focuses on market data-related trading tasks defined by the action buttons in the sample application. We'll take a quick, general look at what's going on behind the GUI, then we'll walk through the basics of requesting market data using the TWS C++ API.

The C++ API for Financial Advisors

Much of the content in this book applies to both individual traders and financial advisors, but there are some specific areas of interest that apply only to advisors who handle multiple clients. This chapter focuses on the features in the C++ sample application that are only available to multi-client account advisors. We discuss the methods, events and parameters used by the C++ API and sample application for Financial Advisors (or FA as we like to call it) accounts.

This section includes the following chapters:

- [Viewing Account Data for Managed Accounts](#)
- [Viewing a List of Managed Accounts](#)
- [Viewing and Modifying FA Configuration Information](#)



Before you go any further, we want to remind you that you should be a professional financial advisor who routinely handles multiple clients in TWS, and is familiar with the financial advisor features in TWS. If you'd like to learn more about TWS's Financial Advisor features, see the [TWS Users' Guide](#), available on our web site. Or, if you want to try out these advisor-only trading tasks in the C++ API sample application yourself but don't have an active Advisor account with TWS, you can run the Advisor Demo available on our website, follow the instructions in the TWS Users' Guide on how to set up an advisor configuration, then run our C++ API sample application.

Viewing Account Data for Managed Accounts

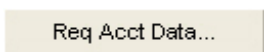
There are three buttons on the C++ sample application that apply to FA accounts:

- **Req Acct Data:** This button lets you get account information for one of the accounts that you manage.
- **Req Accounts:** This button displays a list of account codes for all of the accounts that you manage.
- **Financial Advisor:** This button lets you view/manage your FA account groups, allocation profiles, and account aliases.

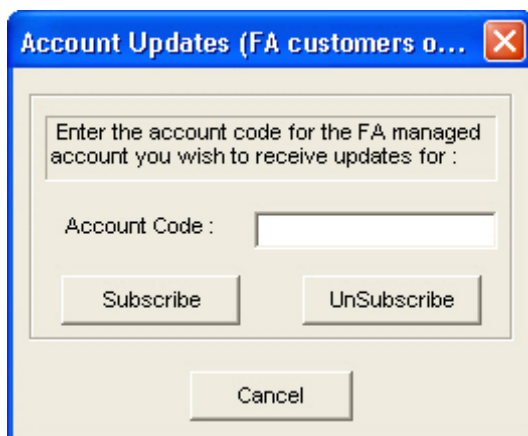


All of the trading tasks described in this section require that you first manually configure Financial Advisor account groups, methods, percentages, allocation profiles and account aliases in TWS before you can use them in the API sample application.

Viewing Account Data for a Managed Account

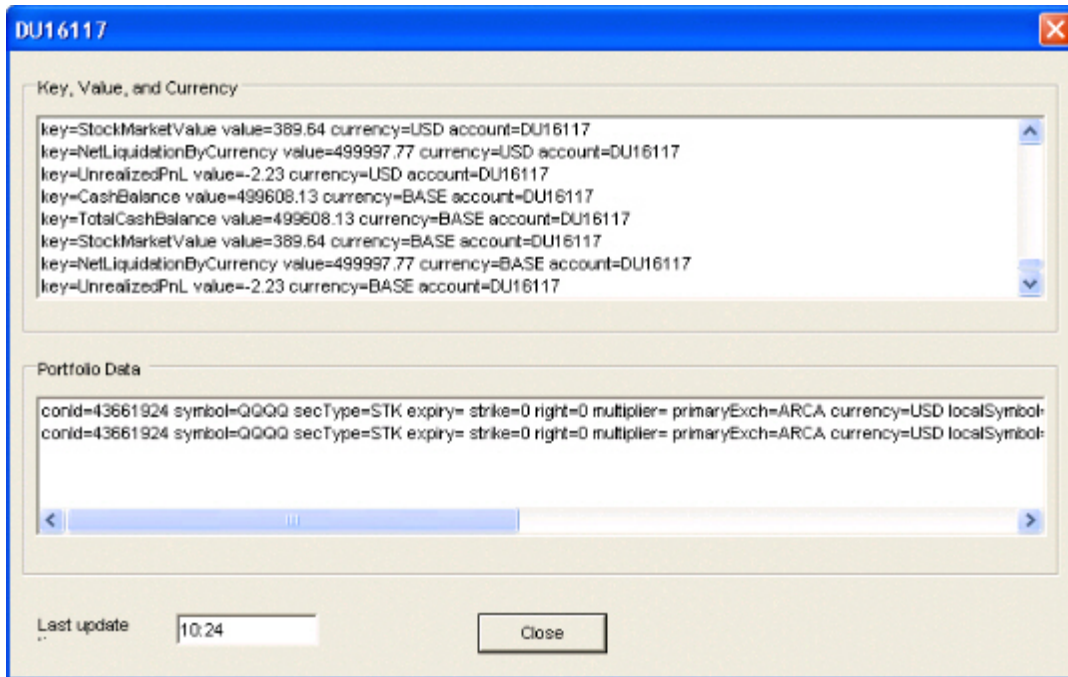


The **Req Acct Data** button in the sample application lets you view account information for one of your managed accounts. If you are a financial advisor and you want to view information about one of your managed accounts (and this is the same kind of information that appears in the Account Window in TWS), first click the **Req Acct Data** button in the sample application. The Account Updates dialog appears.



When you first log into our C++ sample application as a Financial Advisor, the TWS Server Responses text panel displays a list of all of your managed accounts.

As it says in the dialog, you simply type the account code for the managed account of interest in the *Account Code* box, then click **Subscribe**. The information is displayed in a separate window, as shown below. This information dialog displays the account and portfolio data for the specified managed account, and also displays the time of the last update. When you are done viewing this information, you can click **Close** to close the dialog.



Of course, you can unsubscribe to these updates at any time by clicking **Req Acct Data** in the sample application main window, then clicking **Unsubscribe** in the Account Updates dialog.

Now lets see what happens in the code when you click the **Req Acct Data** button.

OnReqAccountUpdate()

When you click the **Req Acct Data** button in the sample application, the **OnReqAccountUpdate()** defined in *client2Dlg.cpp* runs:

```
void CClient2Dlg::OnReqAccountUpdate()
{
    CAcctUpdatesDlg dlg;

    if ( dlg.DoModal() != IDOK) return;

    if ( dlg.getSubscribe() ) {
        s_accountDlg.accountDownloadBegin(dlg.getAcctCode());
    }

    m_pClient->reqAccountUpdates(dlg.getSubscribe(), dlg.getAcctCode());

    // Show the account details dialog if we are subscribing.
    if ( dlg.getSubscribe() )
    {
        s_accountDlg.DoModal();
    }
}
```

OnReqAccountUpdate() does the following:

- Displays the Account Updates dialog (*AcctUpdatesDlg* in the code).
- Calls the C++ method **reqAccountUpdates()** when the **Subscribe** button in the Account Updates dialog is clicked.
- Displays the results dialog, called *AcctUpdatesDlg* in the code, which shows the account details for the managed account number you entered in the Account Updates dialog.

The **Subscribe** button (and the **Unsubscribe** button) in the Account Updates dialog has its own On method as well. This method (NOT part of our C++ API code, but part of the code required by the sample application) basically sets the *subscribe* parameter to TRUE or FALSE, depending on which button you click. If you click the **Subscribe** button in the Account Updates dialog, then the *subscribe* parameter is set to true and the **reqAccountUpdates()** method is called. If you click the **Unsubscribe** button dialog, then the *subscribe* parameter is set to FALSE and the **reqAccountUpdates()** method is not called.



The reqAccountUpdates() Method

This method is called only if you click the **Subscribe** button in the Account Updates dialog.

```
m_pClient->reqAccountUpdates(dlg.getSubscribe(), dlg.getAcctCode());
```

reqAccountUpdates() has two parameters which correspond to your actions in the dialog:

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

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

C++ EWrapper Functions that Return Account Data

The requested account information is received from TWS via the following EWrapper functions:

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

updateAccountValue() returns the actual account data for the specified account code. The account data is returned in a series of key-value pairs, as shown in the table of parameters below.

```
void CClient2Dlg::updateAccountValue(const CString &key, const CString &val,
                                     const CString &currency, const CString &accountName)
{
    s_accountDlg.updateAccountValue(key, val, currency, accountName);
}
```

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
value	The value associated with the key.

Parameter	Description
currency	The currency type, in case the <i>value</i> is a currency type.
account	The account to which the message applies. Useful for Financial Advisor sub-account messages.

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

The **updatePortfolio()** function returns the portfolio data for the specified account code.

```
void CClient2Dlg::updatePortfolio( const Contract& contract, int position,
    double marketPrice, double marketValue,
    double averageCost, double unrealizedPNL,
    double realizedPNL, const CString &accountName)
{
    s_accountDlg.updatePortfolio(contract, position, marketPrice,
    marketValue, averageCost, unrealizedPNL, realizedPNL, accountName);
}
```

The parameters that are returned via this method are listed in the following table.

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

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

The *contract* object used here is the same structure that is used to get market data (and many other trading tasks!). For a complete list of the properties in the *contract* structure, see the [API Reference Guide](#).

updateAccountTime() returns the returns the last update time for the specified account. The last update time is the time at which the account values and portfolio market prices were last calculated. This event has just one parameter, *timeStamp*, which indicates the last update time of the account information. This time is displayed at the bottom of the account information window pictured on the previous page.

```
void CClient2Dlg::accountDownloadEnd(const CString &accountName)
{
    s_accountDlg.accountDownloadEnd( accountName);

    CString str;
    str.Format("Account Download End: %s", accountName);
    int i = m_orderStatus.AddString(str);

    // bring into view
    int top = i - N < 0 ? 0 : i - N;
    m_orderStatus.SetTopIndex(top);
}
```

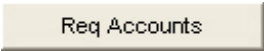
There is one additional EWrapper function that is called, and that is **accountDownloadEnd()**. This function is called when a complete snapshot of an account state is sent to a client as a response to **reqAccountUpdates()**. This function has a single parameter, *accountName*, which is simply the name of the account, and is invoked once after a batch of **updateAccountValue()** and **updatePortfolio()** calls are sent. **accountDownloadEnd()** notification will not be sent for subsequent account updates.

Viewing a List of Managed Accounts

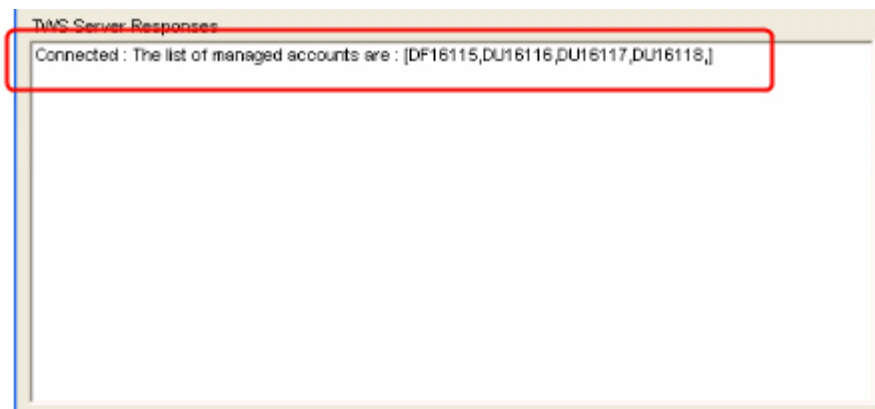
Financial Advisors can use the C++ API sample application to view a list of all managed accounts.

Viewing a List of Managed Accounts

If you are a Financial Advisor, you can use the **Req Accounts** button to view a list of account codes for all of the accounts that you manage.

A rectangular button with a light beige background and a thin border, containing the text "Req Accounts".

When you click this button (and you are logged in with your Advisor account), a list of the account codes for all your managed accounts is displayed in the *TWS Server Responses* text panel of the sample application, as shown below.



Pretty simple, right? Now let's look at the code behind this process.

OnReqAccts()

When you click the **Req Accounts** button, the **OnReqAccts()** method defined in *client2Dlg.cpp* runs:

```
void CClient2Dlg::OnReqAccts()
{
    // request the list of managed accounts
    m_pClient->reqManagedAccts();
}
```

The **OnReqAccts()** does one thing - it calls the **reqManagedAccts()** EClient Socket method. That's it, that's all it does!

The reqManagedAccts() Method

```
m_pClient->reqManagedAccts();
```

The **reqManagedAccts()** method requests a list of managed accounts from TWS. It has no parameters.

C++ EWrapper Functions that Return a List of Managed Accounts

The list of managed accounts is returned by the **managedAccounts()** EWrapper function, which is shown below.

```
void CClient2Dlg::managedAccounts(const CString& accountsList)
{
    m_financialAdvisor = true;
    m_managedAccounts = accountsList;

    CString displayString;
    displayString.Format("Connected : The list of managed accounts are :
[%s]", accountsList);
    m_orderStatus.AddString( displayString);
}
```

The **managedAccounts()** function has one parameter, *accountsList*, which returns a comma-delimited list of managed accounts to the sample application. Note that this event is also called when a successful connection is made to a TWS FA account from the C++ sample application.

Viewing and Modifying FA Configuration Information

The **Financial Advisor** button lets you view and modify your FA configuration information in the C++ sample application, including:

- FA Account Groups - These are groups of accounts and an associated allocation method that you create in TWS.
- FA Allocation Profiles - These are profiles you create in TWS that distribute shares to each account based on a percentage, ratio or absolute number.
- FA Account Aliases - These are recognizable names that you assign to accounts in TWS.

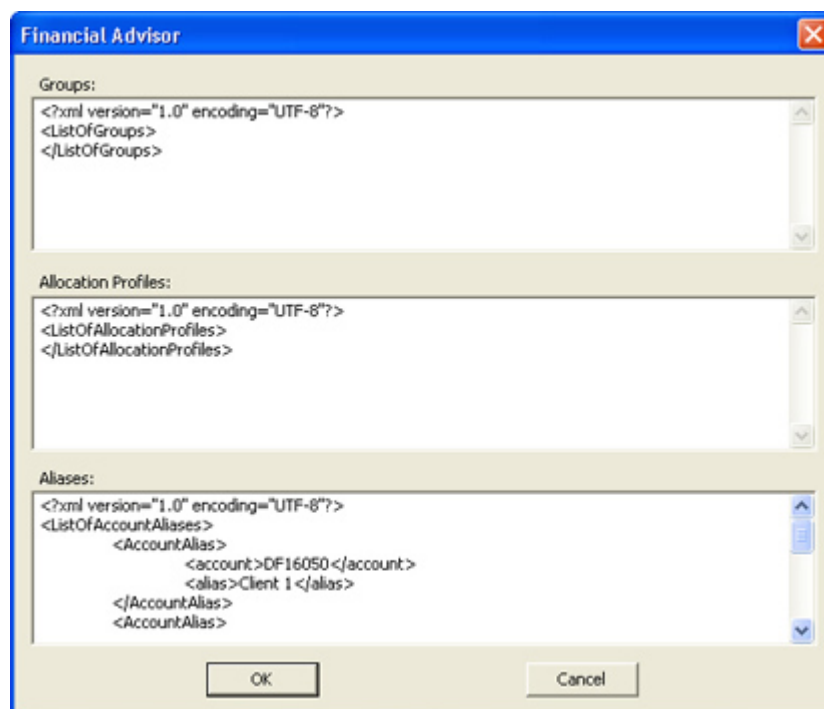


We want to take a moment to remind you again that the trading tasks described in this chapter require you to first manually configure your Financial Advisor account groups, methods, percentages, allocation profiles and account aliases in TWS before trying to use them in the C++ API sample application.

What Happens When I Click the Financial Advisor Button?

Financial Advisor

So, what happens when you, a Financial Advisor, clicks the **Financial Advisor** button? Simply put, you're just requesting your FA configuration information (account groups, allocation profiles and account aliases) from TWS for viewing. The information is displayed as three separate XML files in the Financial Advisor dialog, which is pictured below.



Let's at the code behind this process.

OnFinancialAdvisor()

The On method associated with the **Financial Advisor** button and defined in *client2Dlg.cpp* is **OnFinancialAdvisor**.

```
void CClient2Dlg::OnFinancialAdvisor()  
{  
    faGroupsXML = faProfilesXML = faAliasesXML = "" ;  
    faError = false ;  
    m_pClient->requestFA(GROUPS) ;  
    m_pClient->requestFA(PROFILES) ;  
    m_pClient->requestFA(ALIASES) ;  
}
```

When you click the button, **OnFinancialAdvisor** calls the C++ method **requestFA()**. Actually, if you look in the code sample above, there are three calls to **requestFA()**, one for each type of FA Configuration data requested (aliases, groups and profiles).

The requestFA() Method

```
m_pClient->requestFA(GROUPS) ;  
m_pClient->requestFA(PROFILES) ;  
m_pClient->requestFA(ALIASES) ;
```

This method requests FA configuration information from TWS. It has one parameter, *faDataType*, which identifies the type of FA data to return. Valid values for *faDataType* include:

- 1 = GROUPS
- 2 = PROFILE
- 3 = ACCOUNT ALIASES

As you can see, the three values for this parameter correspond to the three types of FA configuration information requested and ultimately displayed in the Financial Advisor dialog. So when the **OnFinancialAdvisor()** method calls **requestFA()** three times, it's actually calling for each of the three types of FA configuration information.

C++ EWrapper Functions that Return Financial Advisor Configuration Information

The actual FA configuration information is returned as XML strings from the **receiveFA()** EWrapper function. The **receiveFA()** function has two parameters, *faDataType*, which specifies the type of Financial Advisor configuration information to receive (again, 1 = GROUPS, 2 = PROFILE, 3 = ACCOUNT ALIASES), and *xml*, which is the XML string containing the FA configuration information saved on the TWS server.

```
void CClient2Dlg::receiveFA(faDataType pFaDataType, const CString& xml)
```

Keep in mind, all this action occurs behind the scenes and the dialog appears immediately when you click the **Financial Advisor** button.

The replaceFA() Method

There is another EClient Socket method used for Financial Advisor configuration information, **replaceFA()**. You can use the **replaceFA()** method to modify the configuration information from the API, which you can do in the C++ sample application in the Financial Advisor dialog. This information is saved on the TWS server. So, whether or not you actually make changes in the Financial Advisor dialog, the FA configuration information displayed as XML strings is sent to the TWS server via **replaceFA()**. Whenever you view this information in the C++ sample application, the XML strings are returned via the **receiveFA()** event. Thus, the XML strings are being "updated" via **replaceFA()** every time you click **OK** in the Financial Advisor dialog.

Here is the **replaceFA()** method as it is being used in the code for our sample application. These lines of code are actually part of the **receiveFA()** method in *client2Dlg.cpp*

```
.
.
.
if (!faError && faGroupsXML != "" && faProfilesXML != "" && faAliasesXML !=
"" ) {
    CDlgFinancialAdvisor dlg;
    dlg.receiveInitialXML(faGroupsXML, faProfilesXML, faAliasesXML);

    if (dlg.DoModal() != IDOK) return;

    dlg.extractXML(faGroupsXML, faProfilesXML, faAliasesXML);
    m_pClient->replaceFA( GROUPS, faGroupsXML );
    m_pClient->replaceFA( PROFILES, faProfilesXML );
    m_pClient->replaceFA( ALIASES, faAliasesXML );
.
.
.
```

The **replaceFA()** method, shown above, has the same two parameters as the **requestFA()** method, *faDataType*, which specifies the type of Financial Advisor configuration information to receive (1 = GROUPS, 2 = PROFILE, 3 = ACCOUNT ALIASES), and *cxml*, which is the XML string containing the FA configuration information.



If you know how to write XML, you can actually create your FA Groups, Allocation Profiles and Account Aliases directly in the C++ sample application's Financial Advisor dialog. However, it is much easier to set up your FA configuration in TWS first.

Placing Orders Using FA Accounts

In this section, we take a look at how to set up an order for an advisor using a single account group, an account group or an allocation profile. When placing an order as an advisor, you'll use the same order fields in the same Order dialog pictured below and described earlier in this book. If you are an advisor however, there are a couple of extra steps you take in order to enter your allocation profile or account group information.

To place an order using a single account, an account group, and an allocation profile in the C++ sample application, first click the **Place Order** button. In the Place Order dialog, enter the contract information in the *Ticker Description* section and the order information in the *Order Description* section, as shown below. Yes, these are the same fields that you use for any other order placed from the sample application.

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

Section	Field	Value
Contract Description	Id	21
	Contract Id	0
	Symbol	QQQQ
	Type	STK
	Expiry	
	Strike	0
	Right	
	Multiplier	
	Exchange	SMART
	Primary Exchange	ISLAND
	Currency	USD
	Local Symbol	
	Include Expired	0
Order Description	Action	BUY
	Total Order Size	10
	Order type	LMT
	Price	40
	Aux Price	0
	Good After Time	
	Good Till Date	
Buttons		FA Alloc, Combo Legs, Delta Neutral, Algo Params
Market Data	Generic Tick Tags	100,101,10
Market Data	Snapshot	<input type="checkbox"/>
Market Depth	Max Number of Rows	20
Exercise Options	Action (1 or 2)	1
	Number of Contracts	1
	Override (0 or 1)	0
Historical Data	End Date/Time	20090112 07:43:28
	Duration	1 M
	Bar Size Setting	1 day
	What To Show	TRADES
Historical Data	Regular Trading Hours (1 or 0)	1
Historical Data	Date Format Style (1 or 2)	1
Bottom Buttons		OK, Cancel

This is where placing orders as an advisor is different from placing orders for a normal account. Next, click the **FA Alloc** button at the bottom of the Place Order dialog. The FA Allocation Info dialog appears. This is where you will enter your allocation profile or account group information.

The screenshot shows a dialog box titled "FA Allocation Information". It is divided into two sections. The "Group" section contains three input fields: "Group", "Method", and "Percentage". The "Profile" section contains one input field labeled "Profile" with the text "Allocation1" entered. At the bottom of the dialog are two buttons: "OK" and "Cancel".

In the *Profile* box, type the name of the allocation profile you want to use for this order.

- If you enter an allocation profile, you don't have to enter the group information in the *Group* section of the dialog.
- If, however, you want to place the order for a specific account group, type the name of the account group, method and percentage if required by the method in the appropriate boxes. In the sample screen above, the group "NetLiqGroup" was entered in the *Group* field, and the method "NetLiq" was entered in the *Method* field. No percentage is required for this particular method so the *Percentage* field is left blank.
- If you enter a *Profile*, you don't have to enter the *Group* information; conversely, if you fill in the *Group* fields, you don't have to enter a *Profile*. When you have filled in the correct boxes, click **OK**.

Click **OK** to close the FA Allocation Info dialog and place your order. You can always check on the progress of your open orders and executions in the sample application by clicking the appropriate button.

The code behind this process is the same piece of code used when you place a non-FA order. The only difference here is that when the Place Order dialog is initialized and you're logged on with an FA account, the **FA Alloc** button is enabled.

The next couple of sections contains some basic background information about allocation methods and profiles.

Allocation Methods for Account Groups

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

EqualQuantity Method

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

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

NetLiq Method

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

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

AvailableEquity Method

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

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

PctChange Method

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

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

Example 2: You want to close out all long positions for three of the five accounts in a group. You create a sell order and enter "-100" in the Percentage field. The system calculates 100%

of each position for every account in the group that holds a position, and sells all shares to close the positions.

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

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

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

Allocation Profiles

Percentages

This method will split the total number of shares in the order between listed accounts based on the percentages you indicate.

Example: An order for 1000 shares using a profile with four accounts at 25% each would allocate 250 shares to each listed account in the profile.

Ratios

This method calculates the allocation of shares to the listed accounts based on the ratios you indicate.

Example: An order for 1000 shares using a profile with four accounts set to a ratio of 4, 2, 1, 1 would allocate 500, 250, 125 and 125 shares to the listed accounts, respectively.

Shares

This method allocates an absolute number of shares to each account listed. If you use this method, you don't need to enter an order quantity. The order size is determined by adding together the number of shares allocated to each account in the profile.

That's all folks! Of course, you can always look at our [TWS Users' Guide](#) for more details and examples on allocation methods and profiles.

In Summary

Well, you've done it. You've made your way through the C++ sample application and lived to tell about it! Along the way, hopefully you've begun to think about how to create your own Visual Basic or VB.NET application using our C++ API.

Allocation Methods and Profiles

This chapter describes the allocation methods and allocation profiles available to Financial Advisors in Trader Workstation and the API.

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

- [Allocation Methods for Account Groups](#)
- [Allocation Profiles](#)

Allocation Methods for Account Groups

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

EqualQuantity Method

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

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

NetLiq Method

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

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

AvailableEquity Method

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

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

PctChange Method

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

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

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

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

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

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

Allocation Profiles

Percentages

This method will split the total number of shares in the order between listed accounts based on the percentages you indicate.

Example: An order for 1000 shares using a profile with four accounts at 25% each would allocate 250 shares to each listed account in the profile.

Ratios

This method calculates the allocation of shares to the listed accounts based on the ratios you indicate.

Example: An order for 1000 shares using a profile with four accounts set to a ratio of 4, 2, 1, 1 would allocate 500, 250, 125 and 125 shares to the listed accounts, respectively.

Shares

This method allocates an absolute number of shares to each account listed. If you use this method, you don't need to enter an order quantity. The order size is determined by adding together the number of shares allocated to each account in the profile.

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 C++ API can do, and how to make it do some of the things you want. Now we give you a bit more information about how to link to TWS with our C++ API, and we suggest some helpful outside resources you can use to help you move forward.

This section contains the following chapters:

- [Linking to TWS using the TWS C++ API](#)
- [Additional Resources](#)

Linking to TWS using the TWS C++ API

If you have the skill and confidence to handle C++ on your own, you can build your own C++ API application to link to TWS using the following steps as a guide.

To link to TWS using the `TwsSocketClient.dll`

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

Ewrapper Function	Description
<code>tickPrice()</code>	Handles market data.
<code>tickSize()]</code>	
<code>tickOptionComputation()</code>	
<code>tickGeneric()</code>	
<code>tickString()</code>	
<code>tickEFP()</code>	
<code>orderStatus()</code>	Receives order status.
<code>openOrder()</code>	Receives open orders.
<code>error()</code>	Receives error information.
<code>connectionClosed()</code>	Notifies when TWS terminates the connection.
<code>updateAccountValue()</code>	Receives current account values.
<code>updateAccountTime()</code>	Receives the last time account information was updated.
<code>updatePortfolio()</code>	Receives current portfolio information.
<code>nextValidId()</code>	Receives the next valid order ID upon connection.
<code>contractDetails()</code>	Receives contract information.
<code>contractDetailsEnd()</code>	Identifies the end of a given contract details request.
<code>bondContractDetails()</code>	Receives bond contract information.

Ewrapper Function	Description
execDetails()	Receives execution report information.
updateMktDepth()	Receives market depth information.
updateMktDepthL2()	Receives Level II market depth information.
updateNewsBulletin()	Receives IB news bulletins.
managedAccounts()	Receives a list of Financial Advisor (FA) managed accounts.
receiveFA()	Receives FA configuration information.
historicalData()	Receives historical data results.
scannerParameters()	Receives an XML document that describes the valid parameters of a scanner subscription.
scannerData()	Receives market scanner results.
realTimeBar()	Receives real-time bars.
currentTime()	Receives the current system time on the server.
fundamentalData()	Receives Reuters global fundamental market data.

7 Instantiate the EClientSocket class.

8 Call the following functions:

- a** Import `com.ib.client.*` into your source code file.
- b** Implement the EWrapper interface. This class will receive messages from the socket.
- c** Override the following functions:

Ewrapper Functions	Description
tickPrice()	Handles market data.
tickSize()	
tickOptionComputation()	
tickGeneric()	
tickString()	
tickEFP()	
orderStatus()	Receives order status.
openOrder()	Receives open orders.
error()	Receives error information.
connectionClosed()	Notifies when TWS terminates the connection.
updateAccountValue()	Receives current account values.

Where To Go From Here

Linking to TWS using the TWS C++ API

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

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

EClientSocket Functions	Description
eConnect()	Connects to TWS.
eDisconnect()	Disconnects from TWS.
reqMktData()	Requests market data.
cancelMktData()	Cancels market data.
reqMktDepth()	Requests market depth.
cancelMktDepth()	Cancels market depth.
reqContractDetails()	Requests contract details.

EClientSocket Functions	Description
placeOrder()	Places an order.
cancelOrder()	Cancels an order.
reqAccountUpdates()	Requests account values, portfolio, and account update time information.
reqExecutions()	Requests a list of the day's execution reports.
reqOpenOrders()	Requests a list of current open orders for the requesting client and associates TWS open orders with the client. The association only occurs if the requesting client has a Client ID of 0.
reqAllOpenOrders()	Requests a list of all open orders.
reqAutoOpenOrders()	Automatically associates a new TWS with the client. The association only occurs if the requesting client has a Client ID of 0.
reqNewsBulletin()	Requests IB news bulletins.
cancelNewsBulletins()	Cancels IB news bulletins.
setServerLogLevel()	Sets the level of API request and processing logging.
reqManagedAccts()	Requests a list of Financial Advisor (FA) managed account codes.
requestFA()	Requests FA configuration information from TWS.
replaceFA()	Modifies FA configuration information from the API.
reqScannerParameters()	Requests an XML document that describes the valid parameters of a scanner subscription.
reqScannerSubscription()	Requests market scanner results.
cancelScannerSubscription()	Cancels a scanner subscription.
reqHistoricalData()	Requests historical data.
cancelHistoricalData()	Cancels historical data.
reqRealTimeBars()	Requests real-time bars.
cancelRealTimeBars()	Cancels real-time bars.
exerciseOptions()	Exercises options.
reqCurrentTime()	Requests the current server time.
serverVersion()	Returns the version of the TWS instance to which the API application is connected.
TwsConnectionTime()	Returns the time the API application made a connection to TWS.

Where To Go From Here

Linking to TWS using the TWS C++ API

EClientSocket Functions	Description
reqFundamentalData()	Requests Reuters global fundamental data. There must be a subscription to Reuters Fundamental set up in Account Management before you can receive this data.
cancelFundamentalData()	Cancels Reuters global fundamental data.

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

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 Microsoft Visual Studio and C++ Programming

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

The best place to go to find additional help with C++ programming in Microsoft Visual Studio is in the Help menu in Visual Studio or on Microsoft's web site at <http://msdn.microsoft.com/en-us/vstudio/default.aspx>. This is the Visual Studio Developer Center, and from here you can access complete information about Visual Studio. You can also find information specific to Microsoft Visual C++ [here](#).

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

Help with the TWS C++ API

For help specific to the TWS C++ 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, the IB website looks as I'm describing. IB has a tendency to revamp the look and organization of their site every year or two, so have a little patience if it looks slightly different from what's described here.

The API Reference Guide

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

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

The API Beta and API Production Release Notes

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

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

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

The TWS API Webinars

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

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

API Customer Forums

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

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

IB Customer Service

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

api@interactivebrokers.com

IB Features Poll

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

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

Index

- A**
 - Account Updates dialog 4-32
 - accountDownloadEnd() 4-37
 - additional resources 6-59
 - advisors 4-31, 4-32, 4-35, 4-37, 4-38, 4-39, 4-40, 4-41, 4-42, 4-43, 4-44
 - allocation methods for account groups 4-45, 5-50
 - allocation profiles 4-46, 5-52
 - API
 - reasons for using 2-18
 - API beta notes 6-59
 - API Reference Guide 6-59
 - API release notes 6-59
 - API software
 - downloading 3-23
 - installing 3-26
 - API support email 6-60
 - API technologies 2-19
 - API webinars 6-60
 - AvailableEquity Method 4-45, 5-50
- C**
 - C++
 - running the sample application from Visual Studio 2088 3-28
 - C++ API
 - additional resources 6-59
 - installing an IDE 3-22
 - linking to TWS using 6-54
 - preparing to use 3-21
 - C++ API sample application
 - connecting to 3-27
 - C++ API, help with 6-59
 - C++ programming help 6-59
 - connecting to the sample application 3-27
 - customer forums 6-60
 - customer service 6-60
- D**
 - document conventions 1-11
 - downloading API software 3-23
- E**
 - EqualQuantity Method 4-45, 5-50
 - EWwrapper functions
 - account data 4-35
 - FA configuration 4-41
 - managed accounts 4-39
- F**
 - FA Account Aliases 4-40
 - FA Account Groups 4-40
 - FA accounts
 - placing orders for 4-43
 - FA Alloc button 4-43
 - FA Allocation Info dialog 4-44
 - FA Allocation Info... button 4-43
 - FA Allocation Profiles 4-40
 - FA configuration information 4-40, 4-41, 4-42
 - fadataType parameter 4-41
 - Features Poll 6-60
 - Financial Advisor button 4-40
 - Financial Advisor dialog 4-40
 - Financial Advisors 4-31
 - allocation methods for account groups 4-45, 5-50
 - allocation profiles 4-46, 5-52
 - viewing managed account information 4-32
 - viewing managed accounts 4-38
 - footnotes and references 1-9
- H**
 - how to use this book 1-8
- I**
 - IB bulletin boards 6-60
 - IB Customer Service 6-60
 - icons used in this book 1-10
 - IDE, installing 3-22
 - installing API software 3-26
 - integrated development environment 3-22
 - introduction 1-7
- L**
 - linking to TWS 6-54
 - list of managed accounts 4-38
- M**
 - managedAccounts() 4-39
 - Microsoft Visual Studio
 - additional resources for 6-59
- N**
 - NetLiq Method 4-45, 5-50
- O**
 - orders
 - for FA accounts 4-43
 - organization of this book 1-8
- P**
 - PctChange Method 4-45, 5-51
 - placing orders
 - for FA accounts 4-43
 - preparing to use the C++ API 3-21
- R**
 - real-time account monitoring, in TWS 2-17
 - reasons for using an API 2-18
 - receiveFA() 4-41
 - receiveFA() parameters 4-41
 - replaceFA() 4-42
 - replaceFA() parameters 4-42
 - Req Accounts button 4-38
 - Req Acct Data button 4-32
 - reqAccountUpdates() 4-35
 - reqAccountUpdates()
 - parameters 4-35
 - reqManagedAccts() 4-39
 - requestFA() 4-41
 - resources 6-53
 - resources, for C++ programming help 6-59
- S**
 - sample application
 - connecting to 3-27
 - running 3-27, 3-28
 - subscribe 4-35
- T**
 - Trader Workstation
 - overview 2-14
 - trading window 2-16
 - TWS
 - available API technologies 2-19
 - real-time account monitoring in 2-17
 - TWS and the API 2-18
 - TWS Order Ticket 2-16
 - TWS overview 2-14, 2-16
 - TWS Quote Monitor 2-16
 - TwsSocketClient.dll
 - linking to 6-54

U

- updateAccountTime() 4-37
- updateAccountValue() 4-35
- updateAccountValue() parameters 4-35
- updatePortfolio() 4-36
- updatePortfolio() parameters 4-36
- using this book 1-8
 - document conventions 1-11
 - icons 1-10

- organization 1-8

V

- viewing a list of managed accounts 4-38
- viewing managed account information 4-32

W

- where to go from here 6-53