**Trading API & Market Data API**

**Specifications**

**Version: 1.00**

**Document Release Date: 27-3-2017**

|  |
| --- |
| This document is written in both English and Chinese. In case of discrepancy, the Chinese version shall prevail. |



1. **Revision History**

**Revision History**

|  |  |  |
| --- | --- | --- |
| **Version Number** | **Revision Date** | **Brief Description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| 1.00 | 3/27/2017 | First version by SFIT. |

**Verification:**

|  |  |  |
| --- | --- | --- |
| **Verifier Name** | **Department** | **Verification Date** |
|  |  |  |
|  |  |  |

**Approval:**

|  |  |  |
| --- | --- | --- |
| **Approver** | **Department** | **Approval Date** |
|  |  |  |
|  |  |  |

**Maintained by: Shanghai Futures Information Technology Corporation (SFIT )**

Contents

[Part I. Introduction to Trading System Interfaces 8](#_Toc477959502)

[1. Introduction 9](#_Toc477959503)

[1.1. Background 9](#_Toc477959504)

[1.2. TraderAPI Overview 9](#_Toc477959505)

[1.3. MduserAPI Overview 10](#_Toc477959506)

[1.4. Platforms supported by current TraderAPI/MduserAPI 10](#_Toc477959507)

[1.5. Version Amendment History 11](#_Toc477959508)

[1.5.1. Version 1.00 11](#_Toc477959509)

[2. FTD Architecture 11](#_Toc477959510)

[2.1. Communication Mode 11](#_Toc477959511)

[2.2. Data Stream 12](#_Toc477959512)

[3. Interface Type 14](#_Toc477959513)

[3.1. TraderAPI Interface 14](#_Toc477959514)

[3.1.1. Dialog Stream and Query Stream Programming Interface 14](#_Toc477959515)

[3.1.2. Private Stream Programming Interface 14](#_Toc477959516)

[3.1.3. Public Stream Programming Interface 15](#_Toc477959517)

[3.2. MduserAPI Interface 15](#_Toc477959518)

[3.2.1. Dialog Stream Programming Interface 15](#_Toc477959519)

[3.2.2. Market Data Stream Programming Interface 16](#_Toc477959520)

[4. Working Mechanism 17](#_Toc477959521)

[4.1. Work Flow 17](#_Toc477959522)

[4.1.1. Initialization Phase 17](#_Toc477959523)

[4.1.2. Function Invocation Phase 17](#_Toc477959524)

[4.2. Working Thread 17](#_Toc477959525)

[4.3. Interaction Between the Member System and the Exchange’s trading system via TraderAPI 18](#_Toc477959526)

[4.4. Connection to the Exchange’s trading system Gateway 22](#_Toc477959527)

[4.5. Local Files 22](#_Toc477959528)

[4.6. Request/Reply Log Files 22](#_Toc477959529)

[4.7. Subscription Methods for Reliable Data Stream 23](#_Toc477959530)

[4.7.1. Message Replay by Maintaining Sequence Number in API 23](#_Toc477959531)

[4.7.2. Message Replay By Maintaining Sequence Number in the Member System 24](#_Toc477959532)

[4.8. Heartbeat Mechanism (Heartbeat) 25](#_Toc477959533)

[4.9. Gateway List 26](#_Toc477959534)

[4.10. Disaster Recovery Interface 28](#_Toc477959535)

[Part II. TraderAPI Reference Manual 29](#_Toc477959536)

[1. Categories of TraderAPI Interfaces 30](#_Toc477959537)

[1.1. Admin Interfaces 30](#_Toc477959538)

[1.2. Business Interfaces 30](#_Toc477959539)

[1.3. Business Interface Not currently Supported 32](#_Toc477959540)

[2. TraderAPI Reference Manual 33](#_Toc477959541)

[2.1. CShfeFtdcTraderSpi Interface 33](#_Toc477959542)

[2.1.1. OnFrontConnected Method 34](#_Toc477959543)

[2.1.2. OnFrontDisconnected Method 34](#_Toc477959544)

[2.1.3. OnHeartBeatWarning Method 34](#_Toc477959545)

[2.1.4. OnPackageStart Method 34](#_Toc477959546)

[2.1.5. OnPackageEndMethod 35](#_Toc477959547)

[2.1.6. OnRspUserLogin Method 35](#_Toc477959548)

[2.1.7. OnRspUserLogout Method 36](#_Toc477959549)

[2.1.8. OnRspUserPasswordUpdate Method 37](#_Toc477959550)

[2.1.9. OnRspSubscribeTopic Method 38](#_Toc477959551)

[2.1.10. OnRspQryTopic Method 39](#_Toc477959552)

[2.1.11. OnRspError Method 39](#_Toc477959553)

[2.1.12. OnRspOrderInsert Method 40](#_Toc477959554)

[2.1.13. OnRspOrderAction Method 42](#_Toc477959555)

[2.1.14. OnRspQuoteInsert Method 44](#_Toc477959556)

[2.1.15. OnRspQuoteAction Method 46](#_Toc477959557)

[2.1.16. OnRspExecOrderInsert Method 48](#_Toc477959558)

[2.1.17. OnRspExecOrderAction Method 49](#_Toc477959559)

[2.1.18. OnRspAdminOrderInsert Method 51](#_Toc477959560)

[2.1.19. OnRspQryPartAccount Method 52](#_Toc477959561)

[2.1.20. OnRspQryOrder Method 53](#_Toc477959562)

[2.1.21. OnRspQryQuote Method 55](#_Toc477959563)

[2.1.22. OnRspQryTrade Method 57](#_Toc477959564)

[2.1.23. OnRspQryClient Method 58](#_Toc477959565)

[2.1.24. OnRspQryPartPosition Method 59](#_Toc477959566)

[2.1.25. OnRspQryClientPosition Method 60](#_Toc477959567)

[2.1.26. OnRspQryInstrument Method 61](#_Toc477959568)

[2.1.27. OnRspQryInstrumentStatus Method 63](#_Toc477959569)

[2.1.28. OnRspQryBulletin Method 63](#_Toc477959570)

[2.1.29. OnRspQryMarketData Method 64](#_Toc477959571)

[2.1.30. OnRspQryMBLMarketData Method 66](#_Toc477959572)

[2.1.31. OnRspQryHedgeVolume Method 66](#_Toc477959573)

[2.1.32. OnRtnTrade Method 67](#_Toc477959574)

[2.1.33. OnRtnOrder Method 68](#_Toc477959575)

[2.1.34. OnRtnQuote Method 70](#_Toc477959576)

[2.1.35. OnRtnExecOrder Method 71](#_Toc477959577)

[2.1.36. OnRtnInstrumentStatus Method 72](#_Toc477959578)

[2.1.37. OnRtnInsInstrument Method 73](#_Toc477959579)

[2.1.38. OnRtnDelInstrument Method 74](#_Toc477959580)

[2.1.39. OnRtnInsCombinationLeg Method 74](#_Toc477959581)

[2.1.40. OnRtnDelCombinationLeg Method 75](#_Toc477959582)

[2.1.41. OnRtnBulletin Method 76](#_Toc477959583)

[2.1.42. OnRtnAliasDefine Method 76](#_Toc477959584)

[2.1.43. OnRtnFlowMessageCancel Method 77](#_Toc477959585)

[2.1.44. OnErrRtnOrderInsert Method 77](#_Toc477959586)

[2.1.45. OnErrRtnOrderAction Method 78](#_Toc477959587)

[2.1.46. OnErrRtnQuoteInsert Method 79](#_Toc477959588)

[2.1.47. OnErrRtnQuoteAction Method 80](#_Toc477959589)

[2.1.48. OnErrRtnExecOrderInsert Method 81](#_Toc477959590)

[2.1.49. OnErrRtnExecOrderAction Method 82](#_Toc477959591)

[2.1.50. OnRspCombOrderInsert Method 83](#_Toc477959592)

[2.1.51. OnRspQryCombOrder Method 84](#_Toc477959593)

[2.1.52. OnRtnCombOrder Method 86](#_Toc477959594)

[2.1.53. OnErrRtnCombOrderInsert Method 88](#_Toc477959595)

[2.1.54. OnRspQryExecOrder Method 89](#_Toc477959596)

[2.1.55. OnRspQryExchangeRate Method 90](#_Toc477959597)

[2.1.56. OnRspQryInformation Method 91](#_Toc477959598)

[2.1.57. OnRspAbandonExecOrderInsert Method 92](#_Toc477959599)

[2.1.58. OnRspAbandonExecOrderAction Method 93](#_Toc477959600)

[2.1.59. OnRspQryAbandonExecOrder Method 94](#_Toc477959601)

[2.1.60. OnRtnAbandonExecOrder method 95](#_Toc477959602)

[2.1.61. OnErrRtnAbandonExecOrderInsert method 96](#_Toc477959603)

[2.1.62. OnErrRtnAbandonExecOrderAction method 97](#_Toc477959604)

[2.1.63. OnRspQuoteDemand method 98](#_Toc477959605)

[2.1.64. OnRtnQuoteDemandNotify method 99](#_Toc477959606)

[2.2. CShfeFtdcTraderApi Interfaces 99](#_Toc477959607)

[2.2.1. CreateFtdcTraderApi Method 99](#_Toc477959608)

[2.2.2. GetVersion Method 100](#_Toc477959609)

[2.2.3. Release Method 100](#_Toc477959610)

[2.2.4. Init Method 100](#_Toc477959611)

[2.2.5. Join Method 100](#_Toc477959612)

[2.2.6. GetTradingDay Method 101](#_Toc477959613)

[2.2.7. RegisterSpi Method 101](#_Toc477959614)

[2.2.8. RegisterFront Method 101](#_Toc477959615)

[2.2.9. RegisterNameServer Method 101](#_Toc477959616)

[2.2.10. RegisterCryptAlgorithm Method 102](#_Toc477959617)

[2.2.11. RegisterCertificateFile Method 102](#_Toc477959618)

[2.2.12. SetHeartbeatTimeout Method 103](#_Toc477959619)

[2.2.13. OpenRequestLog Method 103](#_Toc477959620)

[2.2.14. OpenResponseLog Method 103](#_Toc477959621)

[2.2.15. SubscribePrivateTopic Method 104](#_Toc477959622)

[2.2.16. SubscribePublicTopic Method 104](#_Toc477959623)

[2.2.17. SubscribeUserTopic Method 104](#_Toc477959624)

[2.2.18. ReqUserLogin Method 105](#_Toc477959625)

[2.2.19. ReqUserLogout Method 106](#_Toc477959626)

[2.2.20. ReqUserPasswordUpdate Method 106](#_Toc477959627)

[2.2.21. ReqSubscribeTopic Method 107](#_Toc477959628)

[2.2.22. ReqQryTopic Method 107](#_Toc477959629)

[2.2.23. ReqOrderInsert Method 108](#_Toc477959630)

[2.2.24. ReqOrderAction Method 109](#_Toc477959631)

[2.2.25. ReqQuoteInsert Method 110](#_Toc477959632)

[2.2.26. ReqQuoteAction Method 111](#_Toc477959633)

[2.2.27. ReqExecOrderInsert Method 112](#_Toc477959634)

[2.2.28. ReqExecOrderAction Method 113](#_Toc477959635)

[2.2.29. ReqAdminOrderInsert method 114](#_Toc477959636)

[2.2.30. ReqQryPartAccount Method 115](#_Toc477959637)

[2.2.31. ReqQryOrder Method 115](#_Toc477959638)

[2.2.32. ReqQryQuote Method 116](#_Toc477959639)

[2.2.33. ReqQryTrade Method 117](#_Toc477959640)

[2.2.34. ReqQryClient Method 117](#_Toc477959641)

[2.2.35. ReqQryPartPosition Method 118](#_Toc477959642)

[2.2.36. ReqQryClientPosition Method 119](#_Toc477959643)

[2.2.37. ReqQryInstrument Method 119](#_Toc477959644)

[2.2.38. ReqQryInstrumentStatus Method 120](#_Toc477959645)

[2.2.39. ReqQryMarketData Method 121](#_Toc477959646)

[2.2.40. ReqQryBulletin Method 121](#_Toc477959647)

[2.2.41. ReqQryMBLMarketData Method 122](#_Toc477959648)

[2.2.42. ReqQryHedgeVolume Method 122](#_Toc477959649)

[2.2.43. ReqCombOrderInsertMethod 123](#_Toc477959650)

[2.2.44. ReqQryCombOrder Method 124](#_Toc477959651)

[2.2.45. ReqQryExecOrder Method 125](#_Toc477959652)

[2.2.46. ReqQryExchangeRate Method 126](#_Toc477959653)

[2.2.47. ReqQryInformation Method 127](#_Toc477959654)

[2.2.48. ReqAbandonExecOrderInsert method 127](#_Toc477959655)

[2.2.49. ReqAbandonExecOrderAction method 128](#_Toc477959656)

[2.2.50. ReqQryAbandonExecOrder method 129](#_Toc477959657)

[2.2.51. ReqQuoteDemand method 130](#_Toc477959658)

[3. TraderAPI—A Development Example 131](#_Toc477959659)

[Part III. MduserAPI Reference Manual 134](#_Toc477959660)

[1. Categories of MduserAPI Interfaces 135](#_Toc477959661)

[1.1. Admin Interfaces 135](#_Toc477959662)

[1.2. Business Interfaces 135](#_Toc477959663)

[2. MduserAPI Reference Manual 136](#_Toc477959664)

[2.1. CShfeFtdcMduserSpi Interface 136](#_Toc477959665)

[2.1.1. OnFrontConnected Method 136](#_Toc477959666)

[2.1.2. OnFrontDisconnected Method 136](#_Toc477959667)

[2.1.3. OnHeartBeatWarning Method 136](#_Toc477959668)

[2.1.4. OnPackageStart Method 137](#_Toc477959669)

[2.1.5. OnPackageEnd Method 137](#_Toc477959670)

[2.1.6. OnRspUserLogin Method 137](#_Toc477959671)

[2.1.7. OnRspUserLogout Method 138](#_Toc477959672)

[2.1.8. OnRspSubscribeTopic Method 139](#_Toc477959673)

[2.1.9. OnRspQryTopic Method 139](#_Toc477959674)

[2.1.10. OnRspError Method 140](#_Toc477959675)

[2.1.11. RegisterCertificateFile Method 140](#_Toc477959676)

[2.1.12. OnRtnDepthMarketData Method 141](#_Toc477959677)

[2.1.13. OnRtnFlowMessageCancel method 142](#_Toc477959678)

[2.2. CShfeFtdcMduserApi Interfaces 143](#_Toc477959679)

[2.2.1. CreateFtdcMduserApi Method 143](#_Toc477959680)

[2.2.2. GetVersion Method 143](#_Toc477959681)

[2.2.3. Release Method 144](#_Toc477959682)

[2.2.4. Init Method 144](#_Toc477959683)

[2.2.5. Join Method 144](#_Toc477959684)

[2.2.6. GetTradingDay Method 144](#_Toc477959685)

[2.2.7. RegisterSpi Method 144](#_Toc477959686)

[2.2.8. RegisterFront Method 144](#_Toc477959687)

[2.2.9. RegisterNameServer Method 145](#_Toc477959688)

[2.2.10. RegisterCertificateFile Method 145](#_Toc477959689)

[2.2.11. RegisterCryptAlgorithm 146](#_Toc477959690)

[2.2.12. SetHeartbeatTimeout Method 146](#_Toc477959691)

[2.2.13. SubscribeMarketDataTopic Method 146](#_Toc477959692)

[2.2.14. ReqUserLogin Method 147](#_Toc477959693)

[2.2.15. ReqUserLogout Method 147](#_Toc477959694)

[2.2.16. ReqSubscribeTopic Method 148](#_Toc477959695)

[2.2.17. ReqQryTopic Method 149](#_Toc477959696)

[3. MduserAPI—A Development Example 150](#_Toc477959697)

[Part IV. Appendix 153](#_Toc477959698)

[1. Error Code 153](#_Toc477959699)

[2. Enumeration Value List—Translated 156](#_Toc477959700)

[3. Data Type List—Translated 159](#_Toc477959701)

# Introduction to Trading System Interfaces

This part introduces the interfaces of the Trading System.

Chapter 1introduces two of the APIs in the Trading System: TraderAPI and MduserAPI. ***TraderAPI*** is designed for member systems to send requests for trading, controlling, and inquiry, and to receive results from the private stream (for execution reports of new orders, order updates and trade etc.), public stream (for market control notices), acknowledgement stream (for responses to new orders etc.) and inquiry stream (for returns from inquiries); ***MduserAPI*** is designed for member systems and Market Data Distributor acquisition systems to receive the market data stream.

Chapter 2 introduces the **FTD (Futures Trading Data)** Exchange Protocol underlying the two APIs, and specifies various data streams.

Chapter 3 illustrates the primary interfaces of two APIs applied to different data stream.

Chapter 4 specifies the function mechanism of two APIs, including the communication between threads, heartbeat mechanism, and how the communication reliability in the private data stream is achieved.

Users of TraderAPI and MduserAPI should read part 1 carefully, as it introduces the fundamentals of Trading System interfaces.

# Introduction

## Background

Shanghai Futures Exchange, Zhengzhou Commodities Exchange, and Dalian Commodities Exchange collaborate on the Futures Trading Data Exchange Protocol (***FTD or FTD Protocol***) under the leadership of China Securities Regulatory Commission (CSRC). CSRC officially released the FTD Protocol on March 25th,2005 (JR/T 0016-2004), and the protocol has since been used as the industry standard. INE follows it.

FTD is the fundamental built-in protocol trading system for member systems to access exchange platforms. However, the Exchange does not recommend that member systems interface with the FTD Protocol directly due to its complexity. Instead, the Exchange has developed TraderAPI and MduserAPI, which wrap FTD, to simplify the development on the client side and improve the reliability of the trading system.

The membermember system connects to the Trading System with TraderAPI, which also contains callbacks for the member system to handle execution reports returned from the Exchange. The member system or the market data distributor system (collectively referred to as” market data receiving terminal”) connects to the Trading System with MduserAPI, which will send callbacks to the market data receiving terminal after receiving market data from the Exchange.

TraderAPI encapsulates the complex protocol conversion, data synchronization, and network communication that occur during trading activities. TraderAPI connects to the trading gateway by establishing a TCP-based virtual link, through which messages related to trading activity (such as new orders) are transmitted. The virtual link created by TraderAPI has characteristics of multi-address registration, automatic reconnection, and message replay.

Like TraderAPI, MduserAPI connects to the market data gateway by establishing a TCP-based virtual link to achieve the purpose of market data acquisition.

## TraderAPI Overview

**TraderAPI** is a C++ based class library. It uses and extends the interfaces provided by the class library to implement all trading related functions, including new orders and quote entries, orders and quote cancellations, orders and quote suspensions, orders and quote reactivations, orders and quote modifications, orders and quote queries, trade queries, client/member info queries, position queries at the member level, position queries at the client level, instrument queries, instrument trading status queries, and exchange advisory queries. The following table lists 5 files included in the class library:

|  |  |  |  |
| --- | --- | --- | --- |
| **File Name** | **Version** | **File Size** | **File Description** |
| FtdcTraderApi.h | V1.0 |  | Header file for the trading interface |
| FtdcUserApiStruct.h | V1.0 |  | Header file defining a series of data types used by the UserAPI |
| FtdcUserApiDataType.h | V1.0 |  | Header file defines a series of business-related data structures |
| traderapi.dll | V1.0 |  | Dynamic-link library (DLL) binary file |
| traderapi.lib | V1.0 |  | Imported library(.lib) file |
| libtraderapi.so | V1.0 |  | Linux Version Dynamic-link library |

TraderAPI runs on MS VC 6.0，MS VC.NET 2003~MS VC.NET 2010, and GCC compilers. The option for multi-thread compilation/MT must be selected.

Note: The Trading System supports several new order types (such as market orders, spreads etc.) and trading types (such as options and quotes on options). However, these new functions are not yet open to the public under the restriction of current exchange trading rules. During the development cycle, the client side developer should carefully read the function spec and note these unavailable functions.

## MduserAPI Overview

**MduserAPI** is also a C++ based class library. Client programs use and extend the interface provided by this class library to implement market data subscription and receiving functions. This class library includes 5 files illustrated in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **File Name** | **Version** | **File Size** | **File Description** |
| FtdcMduserApi.h | V1.0 |  | Header file for Quotation Interface |
| FtdcUserApiStruct.h | V1.0 |  | Header file defining a series of data types required by UserAPI |
| FtdcUserApiDataType.h | V1.0 |  | Header file defining a series of business-related data structures |
| mduserapi.dll | V1.0 |  | Dynamic-link library(.dll) binary file |
| mduapi.lib | V1.0 |  | Library(.lib) file |
| libmduserapi.so | V1.0 |  | Linux Version Dynamic-link library |

MduserAPI runs on MS VC 6.0MS VC.NET 2003~MS VC.NET 2010 and GCCcompilers. The option of multi-threading compilation/MT must be turned on.

## Platforms Supported by Current TraderAPI/MduserAPI

Current TraderAPI and MduserAPI support the following operating systems along with their versions:

* **Intel X86/WindowsXP and later versions**: including **.h** files, **.dll** files and **.lib** files
* **Intel Linux**: including **.h** files and **.so** files; complied in **RedHat kernel 2.6.9-42**

If another OS or version is needed, please contact:

|  |
| --- |
| **+86-21-20617046** |

## Version Amendment History

### Version 1.00

This is the first version of API specifications published by INE, mainly including two parts:

* This version provides trading API specifications:
  + TraderAPI provides multiple interfaces for trading and sending back execution reports.
* This version provides trading API specifications:
  + MdUserAPI provides interfaces to Members and market data distributors to access market data.

# FTD Architecture

The member system does not communicate directly with the core servers. For the purposes of security and risk isolation, the member system is not even aware of the existence of the match engine and market data server. Instead, the member system connects to the proxy servers of the core systems. These proxy servers are the trading gateway and market data gateway. The trading gateway takes care of order routing, and the market data gateway is responsible for market data distribution.

TraderAPI communicates with the order routing gateway via the FTD Protocol built on top of TCP.

MduserAPI communicates with the market data gateway via the FTD Protocol built on top of TCP. The market data gateway receives subscription requests from the market data distributor, and streams the data back to the market data distributor’s acquisition system.



## Communication Mode

All communications within the FTD Protocol are based on a set of communication modes. Each communication mode depicts a way two parties collaborate with each other.

FTD provides the following three communication modes:

* Dialog Communication Mode
* Private Communication Mode
* Broadcast Communication Mode

**Dialog Communication Mode:** The member system initiates a request to the Exchange, such as a new order or order query; the Exchange system processes the request and returns the execution report to the member system. This mode is similar to the client/server communication.

**Private Communication Mode**: The Exchange system takes the initiative to send information such as a fill notification to a specific member or a specific trader under the member.

**Broadcast Communication Mode:** The Exchange system broadcasts the same information to all members participating in trading on the platform. This information includes advisories and public market notices.

Each of above communication modes does not necessarily map to a specific physical link method. In other words, one type of network link method may support multiple communication modes, and one communication mode can facilitate message transmission in multiple network link methods.

Disregarding the difference in communication modes, the communication procedure stays the same among the communication modes, as depicted in Chart 1:

Member System

Exchange

----------

----------

Connection Request

Connection Confirmation

Identification Authentication Request

Identification Authentication Response

Send Request (in dialog mode)

Send Response (in dialog mode)

Send Private Message (in private mode)

Send Market Announcement (in public mode)

Disconnection Request

Disconnection Confirmation

**Chart 1: Workflow for All Communication Modes**

## Data Stream

The Exchange trading gateway supports the dialogue, private, and broadcast communication modes. The Exchange market data gateway supports the dialogue and broadcast communication modes.

**1) Dialog Communication Mode**

The Dialog Communication Mode supports the **dialog data stream** and **query data stream**.

The **dialog data stream** is a bidirectional data flow through which the member system sends trading requests and the Exchange system responds with acknowledgement. The Exchange system does not maintain its status during the dialog data stream. In the event of system failure, the dialog data stream will be reset, which can cause the transmitted data to be lost.

The **query data stream** is a bidirectional data flow through which the member system sends query requests and the trading system returns query results. The Exchange system does not maintain the status of the query stream. In the event of a system failure, the query data stream will be reset, which could result in transmitted data being lost.

**2) Private Communication Mode**

In the Private Communication Mode, data steaming is reliable. Within a trading day, if the member system disconnects and then reconnects, it can request the message to replay starting from a specific sequence number for all privately transmitted data. The private data stream carries messages such as order status reports and fill notifications, and can be further divided into a **member’s private stream** and a **trader’s private stream**.

The Exchange system maintains a private data stream for each participating member system. All messages relating to a specific client, such as execution reports for orders or trade, are transmitted in this private data stream. A trader needs to have enough privilege allotted before subscribing to the client’s private stream.

The trader’s private stream is similar to the member client’s private stream except that it carries responses initiated by the specific trader only. Every trader has the right to subscribe to a personal private stream.

**3) Broadcast Communication Mode**

The broadcast Communication Mode supports the public data stream.

The **public data stream** is a one-way data stream by which the Exchange’s trading system or market data system distributes public market information to the client’s system. The public data stream is reliable because the Exchange system maintains all public data streams. Within a trading day, if a member system disconnects from and then reconnects to the Exchange, it can request to replay a previously transmitted public message starting from a specified sequence number.

For example, the market data stream is a public data stream. It carries market data from the Exchange market data distribution system to the client’s system. The market data stream is reliable, because the Exchange system maintains all market data streams. Within a trading day, if subscriber’s data acquisition system disconnects from and then reconnects to the Exchange, it can request the Exchange system to replay the market data starting from a specific sequence number from the public data stream.

Market data is organized by topics. Each topic gathers market data from the contracts covered in a specific product group, and includes market data contents, publishing methods, market depth, distribution frequency, and delays. The Exchange publishes a list of topics, and configures the topics to which a client subscribes. Each topic maps to a market data stream.

The client’s market data acquisition system connects to the Exchange market data gateway, and then requests subscribing market data from one or more topics.

# Interface Types



## TraderAPI Interface

**TraderAPI** provides two interfaces, namely **CShfeFtdcTraderApi** and **CShfeFtdcTraderSpi**. These two interfaces encapsulate the FTD Protocol.

The member system sends operation requests with **CshfeFtdcTraderApi**, and handles the responses byimplemented callbacks from **CShfeFtdcTraderSpi**.

### Dialog Stream and Query Stream Programming Interface

The programming interface for communication through the dialog stream typically looks like this:

**////Request：**

**int CShfeFtdcTraderApi::ReqXXX (CShfeFtdcXXXField \*pReqXXX,**

**int nRequestID)**

**////Response：**

**Void CShfeFtdcTraderSpi::OnRspXXX(CShfeFtdcXXXField \*pRspXXX,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)**

The first parameter of the request interface is the requested content, and it cannot be empty. This parameter is an object instantiated from a specific class based on the request requirement. Please refer to the appendix “**Enumeration Value List**” and “**Data Type List**” for the available class types and permitted values for the member variables.

The 2nd parameter of the request interface is requests the ID, which should be maintained uniquely in the member system. When the member system receives a response from the Exchange system, requests the ID is wrapped in the returns, thus helping the member system to map the response to the specific request.

The callback in **CShfeFtdcTraderSpi** will be invoked upon receiving a response from the Exchange system. If there are multiple responses, the callback will be invoked multiple times.

The callback function contains four parameters:

The first parameter is the actual data in the response. It is NULL if an error occurs or if no result is returned.

The second parameter is the transaction process status, indicating whether or not the request was processed successfully. If there are multiple returns, starting from the second onwards, the value in this parameter is NULL.

The third parameter is requests the ID designated from the original request.

The fourth parameter is a flag indicating whether this is the last call for the current response.

### Private Stream Programming Interface

The private stream contains data dedicated to a specific member client or trader, such as an execution report from an order entry request, trade, quote entry request, or exercise entry request.

The interfaces to receive message from private stream look like this:

**void CShfeFtdcTraderSpi::OnRtnXXX(CShfeFtdcXXXField \*pXXX)；**

**////or**

**void CShfeFtdcTraderSpi::OnErrRtnXXX(CShfeFtdcXXXField \*pXXX,**

**CShfeFtdcRspInfoField \*pRspInfo);**

The callback in **CShfeFtdcTraderSpi** will be invoked upon receiving messages from the Exchange system in the private data stream. The parameter of the callback function carries the returned content.

### Public Stream Programming Interface

The public stream carries public information broadcasted from the Exchange. This information includes but is not limited to contracts and announcements.

The programming interface to receive a public message from the public stream typically looks like this:

**void CShfeFtdcTraderSpi::OnRtnXXX(CShfeFtdcXXXField \*pXXX)；**

The callback in **CShfeFtdcTraderSpi** is invoked upon receiving data from the Exchange system in the public data stream. The parameter of the callback carries the content of the return.

## MduserAPI Interface

Similar to TraderAPI, the **MduserAPI** also provides two interfaces**CShfeFtdcMduserApi** and **CShfeFtdcMduserSpi**. These two interfaces encapsulate the FTD Protocol.

The market data distributor’s data acquisition system sends a request via **CShfeFtdcMduserApi,** and receives and processes the response by implementing **CShfeFtdcMduserSpi** and overloading the callback functions.

### Dialog Stream Programming Interface

The programming interface for communicating through the dialog stream typically looks like this:

**////Request：**

**int CShfeFtdcMduserApi::ReqXXX(CShfeFtdcXXXField \*pReqXXX,**

**int nRequestID)**

**////Response：**

**void CShfeFtdcMduserSpi::OnRspXXX(CShfeFtdcXXXField \*pRspXXX,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)**

The first parameter of the request interface is the requested content, and cannot be left empty. The second parameter of the request interface is requests the ID. Requests the IDs are maintained by the member system and should be unique. Requests the ID from the original request will be sent to the member system together with the response from the Exchange, so that the client application can match a specific request to its response.

The callback from **CShfeFtdcMduserSpi** will be invoked upon receiving a response from the Exchange system. If there are multiple responses, the callback will be invoked multiple times.

The callback function has four parameters:

The first parameter is the actual response. If an error occurs or no result is returned, this field is set to NULL.

The second parameter is the processing status, indicating whether or not the request was successfully processed. If multiple callbacks occur, the value for this parameter starting from the second onwards will be NULL.

The third parameter is requests the ID from the original request.

The fourth parameter is the flag indicating whether this is the last call for the current response.

### Market data Stream Programming Interface

The market data stream carries market data distributed from the Exchange system.

The interface for receiving market data stream typically looks like this:

**void CShfeFtdcMduserSpi::OnRtnXXX(CShfeFtdcXXXField \*pXXX)；**

The callback in **CShfeFtdcMduserSpi** will be invoked upon receiving market data from the Exchange system via the market data stream. The parameter of the callback contains the received data.

# 

# Working Mechanism



## Work Flow

The interaction between a member system (or a market data distributor’s data acquisition system) and the Exchange system is segmented into two phases: the initialization phase and the function invocation phase.

### Initialization Phase

In the initialization phase, the member system (or vendor’s data acquisition system) must complete following steps (a code example is available in the TraderAPI/MduserAPI Reference Manual).

|  |  |  |
| --- | --- | --- |
| **Steps** | **Member System** | **Market Data Receiving System** |
| 1 | Create an instance of **CShfeFtdcTraderApi** | Create an instance of **ShfeFtdcMduserApi** |
| 2 | Create an event handler instance | Create an event handler instance |
| 3 | Register an event handler instance | Register an event handler instance |
| 4 | Subscribe to the private stream;  Subscribe to the public stream; | Subscribe to the market data stream |
| 5 | Set the IP address of the **NameServer** 1 to look up the trading gateways | Set the IP address of the NameServer to look up the market data gateways |
| 6 | Initialization | Initialization |

1This version of API continues providing interfaces to register the trading gateway and market data gateway for backward compatibility. However, registering directly to these gateways is not recommended, and these interfaces will be removed in the next version. Please refer to Section 4.9formore details about the **NameServer**.

### Function Invocation Phase

In the function invocation phase, the member system can invoke the request methods from TraderAPI**,** e.g. ReqUserLogin and ReqOrderInsert, and retrieve the returns from the callbacks. Please note:

1. Input parameters for the API request function cannot be NULL.
2. Within the output parameter returned from the request, 0 stands for success, and other numbers indicate an error. Please refer to the **Appendix** “**Error Code List**” for more details.

## Working Thread

The client application of a member system or market data acquisition system consists of a minimum of two threads: one is the main thread, and the other is the API working thread (TraderAPI or MduserAPI). The communication between the client application and the Exchange trading gateway or market data gateway is driven by the API working thread.

The interfaces provided by CShfeFtdcTraderApi and CShfeFtdcMduserApi are thread-safe, and therefore multiple client applications are permitted to send requests simultaneously.

The callbacks provided by CShfeFtdcTraderSpi are triggered by the TraderAPI working thread. By implementing the SPI interface, the client application is able to retrieve the returned data from the Exchange trading gateway.

Similarly, the callbacks provided by CShfeFtdcMduserSpi are driven by the MduserAPI working thread. The client application collects the returned data from the market data gateway by implementing the SPI interface.

If any overloaded callback becomes stuck, the TraderAPI or MduserAPI working thread will be impacted and will also become stuck. This will cause the termination of communication between the API and the Exchange gateway. When this happens, the client application should immediately return from the callback. In the callback of the extended CShfeFtdcTraderSpi or CShfeFtdcMduserSpi class, a quick return can be achieved by storing data into the buffer or through the Windows messaging mechanism.

SPI Object

Replies to API requests,

e

.

g

.

,

transaction information

,

client error information

,

contract modification

information

,

etc

.

Exchange trading gateway

API Object

API requests

,

e

.

g

.

,

login request

,

order request

,

query request

,

etc

.

Member System

SPI object

Quotation

Exchange market data gateway

API Object

Login request

Subscription request

Market Data Acquisition System

Chart 2. TradeAPI working thread Chart 3. MduserAPI working thread

## Interaction Between the Member System and the Exchange’s Trading System via TraderAPI

A member system interacts with the Exchange’s trading system through TraderAPI. Requests from the member system are sent to the Exchange’s trading system through TraderAPI; the execution reports from the Exchange’s trading system are sent to the member system through TraderAPI as well.

The trading interface and private stream interface of TraderAPI are interrelated. For instance, a trader submits an order entry in **ReqOrderInsert**, and instantly receives the order acknowledgement in **OnRspOrderInsert,** indicating that the Exchange’s trading system has received the order. After the order is accepted into the order book, or whenever the status of the order changes, the Exchange’s trading system will send the order execution report in **OnRtnOrder.** If the order is matched (including a partial match), the Exchange’s trading system will send a fill notification in **OnRtnTrade**. One trader’s order/trade execution report is visible to the other traders under the same member, unless this trader chooses to receive returns from his/her personal stream only.

To illustrate the interaction with a daily trading example, assuming that there are two member systems (A and B), and the following events occur:

**1). Trader A places an order on contract cu0711, buy, 20 lots, 64000RMB**

* **CShfeFtdcTraderApi::ReqOrderInsert:** The main client application thread invokes this function to add a new order into the Exchange’s trading system through the dialog stream.
* **Trading System Order Processing:** The Exchange’s trading system allocates the Exchange sequence number “1” to the new order. Because there is no matching counterparty in the order book at that moment, the order status is set to “**Not Traded and Still Queuing**”. The Exchange’s trading system gateway sends an order acknowledgement to trader A via the dialog stream, and sends an order execution report to Trader A and Trader A’s member system via the private data stream. The contents of the acknowledgement and execution report can be loaded from Spi in the TraderAPI working thread.
* **CShfeFtdcTraderSpi::OnRsqOrderInsert:** Upon the member system receiving a response from the Exchange’s trading system, the TraderAPI working thread invokes this callback to parse the acknowledgement for the new order entry request: Request is successfully accepted; the local sequence number is “1”, and the Exchange sequence number is “1”.
* **CShfeFtdcTraderSpi::OnRtnOrder:** The trading system gateway sends an order execution report to Trader A and his/her member system via the private stream. The execution report includes all information about the order, including its status. The TraderAPI working thread invokes this callback to parse the returned contents. If Member A has other traders logged into the trading platform at that moment, and is listening to the member’s private stream, these traders will receive the execution report as well.

**2). Trader B places an order on contract cu0711, sell, 10 lots, 64000 RMB**

* **CShfeFtdcTraderApi::ReqOrderInsert:** New order entry request.
* **Order Processing of the Trading System:** The Exchange’s trading system allocates the Exchange sequence number “2” to the new order and set the status to “Not Filled and Still Queuing”.
* **Trading System Matching Orders:** The new order is matched with an existing order on the order book, and its status is set to “All Filled”. The trading system gateway sends:
* an order acknowledgement to TraderB’s dialog stream;
* an order execution report to Trader B and his/her member’s private stream;
* an order execution report to Trader A and his/her member’s private stream, with an order status of “**Partially Filled and Still Queuing**” and a remaining quantity of 10;
* a trade execution report to Trader B and his/her member’s private stream;
* a trade execution report to Trader A and his/her member’s private stream

trading system to ensure that:

* + the order execution report is delivered ahead of the trade execution report (fill notification)
  + The remaining quantity in the execution report is updated to reflect the most recent value, thus avoiding adjustment on the client side with the traded volume from the trade execution report.
* **CShfeFtdcTraderSpi::OnRsqOrderInsert:** The Exchange’s trading system responds to a new order request with content: request is successfully accepted; the local sequence number is “1” and system sequence number is “2”.
* **CShfeFtdcTraderSpi::OnRtnOrder:** The Exchange trading gateway sends an order execution report to Trader B and his/her member’s system with the order status of “**All Filled**”.
* **CShfeFtdcTraderSpi::OnRtnOrder:** The Exchange trading gateway sends an order execution report to Trader A and his/her member’s private stream with the status of “**Partially Filled and Still Queuing**” and a remaining unmatched quantity of 10
* **CShfeFtdcTraderSpi::OnRtnTrade:** The Exchange trading gateway sends a trade execution report (fill notification) to Trader A and his/her member’s private stream
* **CShfeFtdcTraderSpi::OnRtnTrade:** The Exchange trading gateway sends a trade execution report (fill notification) to Trader B and his/her member’s private stream.

**3). Trader A cancels the order**

The following chart describes the UML interaction among the member system, TraderAPI, and the Exchange’s trading system.

Chart 4: Interaction between the member systems and the Exchange’s trading system

## Connection to the Exchange’s Trading System Gateway

The member system uses TraderAPI and MduserAPI to communicate with the Exchange trading and market data gateway. The API leverages the FTD that is built on top of the TCP. TraderAPI uses the **CShfeFtdcTraderApi::RegisterFront** method to register the IP address of the Exchange trading gateways. The **MduserAPI** uses the **CShfeFtdcMduserApi::RegisterFront** to register the IP address of the Exchange market data gateways.

The exchange sets multiple trading and market data gateways to achieve the purposes of load balancing and fault tolerance, thus improving performance and reliability of the trading system. TraderAPI and MduserAPI may register multiple gateways. After initialization, the API chooses one of the registered gateways, and attempts to establish a network connection to it. If the attempt fails, it will try to connect to other registered gateways one by one until a connection is successfully established. If a network failure occurs during trading, the API will attempt to connect to the other gateways in the same fashion.

The exchange publishes IP addresses for at least two gateways located in the Shanghai Futures Exchange Tower and Zhangjiang respectively. Each member system should register the IP addresses of at least two gateways to prevent single point failure that will likely result from a network breakdown. Due to the fact that the FTD requires more bandwidth than OFPV1 or OFPV2, it is highly recommended that all members use a dedicated DDN link with a bandwidth of 128K and above, or a dedicated SDH digital link with a bandwidth of 2M. Trading members who trade in both INE and CFFEX can share the same link access to both exchanges. Therefore, technically a trading member can take the link to INE as a backup link to access CFFEX if his/her direct link to CFFEX is broken, and vice versa.

## Local Files

TraderAPI will write some data into local files during the running process. The **CreateFtdcTraderApi** function provides an input parameter that specifies the path pointing to these local files. The file path must be created before TraderAPI is initialized. The local files must have the extension “.con”.

**MduserAPI** works in similar manner but with differently functioning **CreateFtdcMduserApi**.

## Request/Reply Log Files

TraderAPI offers two log-relevant interfaces to record communication logs. **OpenRequestLog** is to open the log for requests, while **OpenResponseLog** is to open the log for replies. After the logs are opened, the client application writes all requests into the request log, and all acknowledgements and execution reports into the reply log. For the purposes of confidentiality and saving storage space, a login request/reply or query request/reply should not be written in the log,

Log Format for Request:

Date time, request name, request result, [request parameter name, request parameter content]

Log Format for Acknowledgement:

Date time, reply name, response ID, response content, [reply parameter name, reply parameter content]

Log Format for Execution Report

Date time, return name, [return parameter name, return parameter content]

## Subscription Methods for Reliable Data Stream

In the FTD protocol, the private stream, public stream, and market data stream are invoked reliable data streams because they can transmit data from one end to the other in a reliable and orderly manner. Reliable data streams are critical to ensuring the correctness and completeness of the data in the member systems. For example, the member system can obtain sufficient information from various execution reports received in the private streams to help it complete business operations. In order to guarantee the correctness of business operations, the member system must receive messages from the private streams in a reliable manner to ensure that the received data is sequenced and unique.

Reliable data stream transmission requires a mechanism for message replay. The client side is responsible for keeping the sequence number that is received from the data stream. In case of interrupted transmission, the client side can request message replay from a specific sequence number, in order to maintain message completeness and data integrity on its end.

The dialog stream and query stream do not support replay, and therefore are unreliable streams.

TraderAPI offers two methods of managing reliable data streams: Maintaining the message sequence number in API for message replay, or maintaining the message sequence number in the member system for message replay.

### Message Replay by Maintaining the Sequence Number in API

When API receives an execution report from a reliable data stream, it (a) firstly invokes the callback function in SPI to inform the member systemsystem, then (b) records the message sequence number in the local file (with file extension “.con”). If the member system logs in again and requests message replay, it can leverage this sequence number from the local file to ask for data retransmission.

**SubscribePrivateTopic, SubscribePublicTopic**, and **SubscribeUserTopic** from CShfeFtdcTraderApi and **SubscribeMarketDataTopic** from CShfeFtdcMduserApi are used to subscribe reliable data streams.

Message replay can be performed in three styles, depending on the value passed in the interface parameter: RESTART, RESUME, or QUICK (snapshot).

* **RESTART** means replaying from the first message in the stream. The message sequence number from the local file is ignored.
* **RESUME** means replaying from the last sequence number recorded in the local file. For market data replay, a snapshot of each contract covered in the specific topic is firstly transmitted from the current moment, followed by a market data message starting from the specified sequence number (so that the member system can make up the lost messages while still receiving the new stream). The exchange recommends “RESUME” for replaying data from the member’s or trader’s private stream, to ensure the data’s integrity.
* **QUICK** means starting the replay from the largest sequence number (most recent message) at the moment of request. For market data, the most recent snapshot for each contract covered in the subscribed topic will be taken and transmitted to the member systemsystem. QUICK can be used when the urgency is more attached to restarting trading or receiving data rather than maintaining the data’s completeness and integrity. The exchange does not recommend QUICK if it is to request replay from a trader’s or member’s private stream.

Message replay by API has the potential risk of losing data consistency, for example if (a) the invoking callback is finished while (b) the recording message sequence number is not, then the replay can result in duplicate messages being loaded into the member system; or, if the local file storing the sequence number is damaged, the message replay has to start from the very beginning, affecting the efficiency and performance of the member system.

If maintaining a message sequence number in API, API needs to keep the value returned from the previous login reply in the fields “TradingDay” and “DataCenterID” into the file “resume.con”. On re-login, API should overwrite the values in these two fields filled by the member system with values saved in this file.

### Message Replay by Maintaining the Sequence Number in the Member System

When API receives an execution report from a reliable data stream, it (a) first invokes **OnPackageStart** to inform the member system of the arrival of the execution report, then (b) invokes the callback in SPI to parse the returned business data, and (c) finally invokes the **OnPackageEnd** to declare the completion of the callback. From the interfaces **OnPackageStart** and **OnPackageEnd**, the member system can retrieve the sequence number of the execution report in the callback and log it. Later on, this sequence number can be passed to the parameter in **CShfeFtdcTraderApi::ReqSubscribeTopic** for message replay (similar to the RESUME mode).

With the **CShfeFtdcTraderApi::ReqSubscribeTopic** function, the member system can designate the sequence number from which the replay should start. If the designated sequence number is 0, the entire data stream will be replayed (similar to RESTART); if the designated sequence number is -1, the replay will start from the largest sequence number (or latest) at this moment (similar to the QUICK mode).

If the subscribed stream is the market data stream and the designated sequence number is not 0, a market data snapshot on each subscribed contract before the designated sequence number is transmitted, then followed by the subsequent market data. During the transmission of the market data snapshots, the value of parameter **nSequenceNo** in the callback **OnPackageStart** and **OnPackageEnd** is 0.

Maintaining the sequence number in the member system for message replay achieves better consistency and reliability than maintaining it in API. Cmember systems with a high demand for the integrity of data should adopt this method.

Notably, at re-login, the **TradingDay** and **DataCenterID** fields should be filled with the returned value from the previous login. For the first-time login or re-login without message replay, the **TradingDay** field can be set to an empty string, and the **DataCenterID** can be filled with either 0 or the primary data center ID published by the Exchange.

## Heartbeat Mechanism (Heartbeat)

The member system communicates with the Exchange trading platform via a TCP virtual link. If this virtual link fails at a moment when there is no business communication between the member system and the Exchange (more specifically, if **Socket recv()** and **Socket send()**are not invoked on both ends), then it is hard to identify the network breakdown until the socket itself times out. Since the default timeout is set relatively long, this poses difficulty for both ends in detecting the abnormal status, and thus could delay the response and recovery.

Adding a heartbeat is a simple and cost-effective way to solve this issue: when business data is under transmission, both ends rely on communication to monitor the health of the link; otherwise, both ends can send a heartbeat to each other in the absence of business communication. This does not add pressure to the bandwidth because no business data is transmitted when the heartbeat message is transmitted. However, along with the increase in the quantity of the server(gateway), it does add linear operation costs, as each server must be polled every second to ascertain whether or not to send a heartbeat and also maintain the connection table.

Adding a heartbeat helps detect the health of a link. If one end does not receive a message before timeout, it deems that the TCP virtual link has failed and should disconnect. If one end does not send out any business data within a certain interval, it should send a heartbeat to the other end so as to maintain the normal working conditions of the virtual link. Typically, the **timeout** is set to three times the **interval**.

API provides a **SetHeartbeatTimeout (unsigned int timeout)** method for the member system to set the timeout period to monitor the validity of the TCP virtual link: During an idle period, the Exchange’s trading system sends a heartbeat to API every (**timeout-1**) / 3 seconds. If API does not receive the message from the Exchange’s trading system within **timeout**/2 seconds, it triggers the callback function **CShfeFtdcTraderApi::OnHeartBeatWarning ()**; if API does not receive a message from the Exchange’s trading system after **timeout**, it disconnects and triggers callback **CShfeFtdcTraderApi:: OnFrontDisconnected ()**.

For instance, assuming that the client side sets the heartbeat timeout to 16 seconds, then the Exchange’s trading system will send one heartbeat to the API every 5 seconds during an idle time. If API does not receive a message from the Trading System in 8 seconds, it invokes a callback **CShfeFtdcTraderApi::OnHeartBeatWarning ()**. If API does not receive a message in 16 seconds, it disconnects and triggers callback**CShfeFtdcTraderApi::OnFrontDisconnected ()**. When disconnectionhappens, the member system may choose the backup link to connect back to the Exchange gateway.

The Exchange gateway also uses a heartbeat mechanism to monitor the TCP connection to the member system. If the member system does not set the heartbeat timeout with the **SetHeartbeatTimeout** method, the default value is 10 seconds. Otherwise, the Exchange gateway will synchronize with the value set through **SetHeartbeatTimeout** to monitor the member system. This mechanism is very helpful. When the link to the member system is broken, the Exchange gateway will automatically disconnect the TCP connection within timeout + 5 seconds. This helps the member system to recognize the ineffectiveness of the TCP connection so that it allows the backup link (with a different IP address) to reconnect.

Note:

The minimum timeout is 4 seconds. If the **timeout** is set too high, it could take the member system a much longer time to switch to the alternative link in the event of disrupted connection. If the **timeout** is set too low, unexpected switching might occur. Therefore, setting the timeout correctly requires a combined consideration on the member system’s operation and network status.

The exchange suggests setting the timeout at value of 10-30 seconds.

## Gateway List

A member system connects to the trading system through the Exchange gateways. For the purpose of load balancing and fault tolerance, both primary and backup data centers set a cluster of gateways. The member system selects one of gateways from the Exchange’s published list of gateway IP addresses and connects to it. If the connected gateway breaks down, the member system should reconnect with a different gateway from the list. The member system can only connect to one gateway at a time.

The way for a member system to obtain the gateway address is described below:

Retrieve the gateway IP address from NameServer, which stores the network addresses of all gateways. The member system firstly registers the Exchange’s published NameServer IP address with the API **RegisterNameServer** method, and then retrieves a gateway address from NameServer.

The advantages of employing NameServer include:

* Enhanced deployment flexibility. Based on business requirements and load balancing needs, adding a new gateway does not cause any changes on the client end.
* NameServer provides a better way of switching between the primary exchange system and the disaster recovery system.
* NameServer can be deployed flexibly, attributing to its singular functionality, simple structure, and low usage. The exchange provides two NameServers running in a live-live fashion at each data center.

The flow-charts below show how API connects to the Exchange gateway:

Connected

successfully?



Randomly choose one



Establish a

session with the

gateway

Have all gateways in the gateway list been tried?

Yes

No

End

Yes

Start

No



Randomly choose a

NameServer from

the NameServer list

and connect to it

Yes

Connected

successfully

?



Obtain the

gateway list from

the NameServer



Register the

gateway list



Register NameServer



Register the gateway list

Have all the

No

Yes

Yes

No

gateway from the gateway

list and connect to it

NameServers in the

NameServer list been tried?

Chart 5. Using API to connect to the Exchange gateway

## Disaster Recovery Interface

The Exchange has two data centers, located at the SHFE Tower and Zhangjiang respectively. The two centers are connected by high-speed optical fiber. The trading system runs in both: the primary data center takes care of business process, while the backup data center receives data from the primary data center and synchronizes business transactions.

If the primary data center encounters a disaster, all trading activities will switch over to the backup data center. The backup data center becomes the primary data center, and continues business processing. It is likely that this will result in data loss. If this happens, the member system must acquire information about rolled-back transactions through the API.

1). In the login request, the user must fill the “Data Center ID” field with the value that was used in the last login. When the Exchange responds to the login request, it fills in the data center ID that is currently being used. The member system should keep this returned data center ID and use it in its next login request.

2). After the member system logs in again and submits a subscription request, it receives a range of sequence numbers from **OnRtnFlowMessageCancel** for the rolled-back transaction by the Exchange due to the switchover. Based on the sequence number, the member system can find the corresponding execution reports from **OnPackageStart** and **OnPackageEnd**.

# TraderAPI Reference Manual

Software engineers can refer to this part to develop an interface to the Exchange trading platform.

Chapter 1 lists the interfaces and methods in **TraderAPI** from the perspective of system management and business operation, together with the public availability of each interface.

Chapter 2 is a reference manual for **TraderAPI**.

Chapter 3 presents a programming example of **TraderAPI**.

Error codes, enumeration values, and data typesarelisted in Part IV. Appendix.

# Categories of TraderAPI Interfaces

## Admin Interfaces

**TraderAPI**admin interfaces control the life cycle and operation parameters.

| **Interface Type** | **Interface Name** | **Explanation** |
| --- | --- | --- |
| Lifecycle Management Interfaces | CShfeFtdcTraderApi ::CreateFtdcTraderApi | Create a **TraderApi** instance |
| CShfeFtdcTraderApi ::GetVersion | Get API version |
| CShfeFtdcTraderApi ::Release | Delete the instance of interface |
| CShfeFtdcTraderApi ::Init | Initialization |
| CShfeFtdcTraderApi ::Join | Wait for Interface thread to terminate |
| Parameter Management Interfaces | CShfeFtdcTraderApi ::RegisterSpi | Register callback interface |
| CShfeFtdcTraderApi ::RegisterFront | Register the IP address of gateway |
| CShfeFtdcTraderApi ::RegisterNameServer | Register the IP address of **NameServer** |
| CShfeFtdcTraderApi ::RegisterCertificateFile | Load certificate |
| CShfeFtdcTraderApi:: RegisterCryptAlgorithm | Register the encryption algorithm for user certification |
| CShfeFtdcTraderApi ::SetHeartbeatTimeout | Set the timeout for heartbeat |
| Subscription Interfaces | CShfeFtdcTraderApi ::SubscribePrivateTopic | Subscribe to private stream |
| CShfeFtdcTraderApi ::SubscribePublicTopic | Subscribe to the public stream |
| CShfeFtdcTraderApi ::SubscribeUserTopic | Subscribe to trader’s stream |
| Audit Log Interfaces | CShfeFtdcTraderApi ::OpenRequestLog | Open the request log file |
| CShfeFtdcTraderApi ::OpenResponseLog | Open the response log file |
| Communication Status Interfaces | CShfeFtdcTraderSpi ::OnFrontConnected | This method is invoked when connection to the Exchange’s trading system is established (but before login) |
| CShfeFtdcTraderSpi ::OnFrontDisconnected | This method is invoked when the member system disconnects from the Exchange’s trading system |
| CShfeFtdcTraderSpi ::OnHeartBeatWarning | This method is invoked when no heartbeat is received for a preset period of time |
| CShfeFtdcTraderSpi ::OnPackageStart | This method is invoked to announce the arrival of a response package |
| CShfeFtdcTraderSpi ::OnPackageEnd | This method is invoked to announce the end of a response package |
| Disaster Recovery Interfaces | CShfeFtdcTraderSpi ::OnRtnFlowMessageCancel | This method is invoked to retrieve cancelled executions after the data center switches to the backup |

## Business Interfaces

| **Business Type** | **Business Name** | **Request Interface / Response Interface** | **Data Stream** |
| --- | --- | --- | --- |
| Login/logout | Login | CShfeFtdcTraderApi :: ReqUserLogin  CShfeFtdcTraderSpi :: OnRspUserLogin | N / A |
| Logout | CShfeFtdcTraderApi :: ReqUserLogout  CShfeFtdcTraderSpi :: OnRspUserLogout | Dialogue Stream |
| Update User Password | CShfeFtdcTraderApi ::ReqUserPasswordUpdate  CShfeFtdcTraderSpi ::OnRspUserPasswordUpdate | Dialogue Stream |
| Subscription | Subscribing Subject | CShfeFtdcTraderApi :: ReqSubscribeTopic  CShfeFtdcTraderSpi :: OnRspSubscribeTopic | Dialogue Stream |
| Querying a Topic | CShfeFtdcMduserApi :: ReqQryTopic  CShfeFtdcMduserSpi :: OnRspQryTopic | Query Stream |
| Trading | Adding a New Order | CShfeFtdcTraderApi :: ReqOrderInsert  CShfeFtdcTraderSpi :: OnRspOrderInsert | Dialogue Stream |
| Acting on an Existing Order (modifying, suspending, reactivating) | CShfeFtdcTraderApi :: ReqOrderAction  CShfeFtdcTraderSpi :: OnRspOrderAction | Dialogue Stream |
| Adding a New Spread Order | CShfeFtdcTraderApi :: ReqCombOrderInsert  CShfeFtdcTraderSpi :: OnRspCombOrderInsert | Dialogue Stream |
| Adding a New Quote | CShfeFtdcTraderApi :: ReqQuoteInsert  CShfeFtdcTraderSpi :: OnRspQuoteInsert | Dialogue Stream |
| Acting on an Existing Quote | CShfeFtdcTraderApi :: ReqQuoteAction  CShfeFtdcTraderSpi :: OnRspQuoteAction | Dialogue Stream |
| Adding a New Order of (option) Exercising | CShfeFtdcTraderApi :: ReqExecOrderInsert  CShfeFtdcTraderSpi :: OnRspExecOrderInsert | Dialogue Stream |
| Acting on an Existing Order to (option) Exercise | CShfeFtdcTraderApi :: ReqExecOrderAction  CShfeFtdcTraderSpi :: OnRspExecOrderAction | Dialogue Stream |
| Adding a New Exercise Abandonment | CShfeFtdcTraderApi:: ReqAbandonExecOrderInsert  CShfeFtdcTraderSpi:: OnRspAbandonExecOrderInsert | Dialogue Stream |
| Acting on an Existing Order to Abandon an Exercise | CShfeFtdcTraderApi:: ReqAbandonExecOrderAction  CShfeFtdcTraderSpi:: OnRspAbandonExecOrderAction | Dialogue Stream |
| Adding a New Quote Demand | CShfeFtdcTraderApi:: ReqQuoteDemand  CShfeFtdcTraderSpi:: OnRspQuoteDemand  CShfeFtdcTraderSpi::OnRtnQuoteDemandNotify | Dialogue Stream |
| Private Return | Trade Execution Report | CShfeFtdcTraderSpi :: OnRtnTrade | Private Stream |
| Order Execution Report | CShfeFtdcTraderSpi :: OnRtnOrder | Private Stream |
| Spread  Order Execution Report | CShfeFtdcTraderSpi :: OnRtnCombOrder | Private Stream |
| Price Quotation Return | CShfeFtdcTraderSpi :: OnRtnQuote | Private Stream |
| Exercise Execution Report | CShfeFtdcTraderSpi :: OnRtnExecOrder | Private Stream |
| Order Entry Error Return | CShfeFtdcTraderSpi :: OnErrRtnOrderInsert | Private Stream |
| Order Action Error Return | CShfeFtdcTraderSpi :: OnErrRtnOrderAction | Private Stream |
| Spread Order Entry Error Return | CShfeFtdcTraderSpi ::OnErrRtnCombOrderInsert | Private Stream |
| RFQEntry Error Return | CShfeFtdcTraderSpi :: OnErrRtnQuoteInsert | Private Stream |
| RFQ Action  Error Return | CShfeFtdcTraderSpi :: OnErrRtnQuoteAction | Private Stream |
| Exercise Entry Error Return | CShfeFtdcTraderSpi :: OnErrRtnExecOrderInsert | Private Stream |
| Exercise Action Error Return | CShfeFtdcTraderSpi ::OnErrRtnExecOrderAction | Private Stream |
| Public Notice | Instrument Status | CShfeFtdcTraderSpi :: OnRtnInstrumentStatus | Public Stream |
| New Instrument | **CShfeFtdcTraderSpi :: OnRtnInsInstrument** | Public Stream |
| Instrument Deletion | **CShfeFtdcTraderSpi :: OnRtnDelInstrument** | Public Stream |
| New Spread Instrument | **CShfeFtdcTraderSpi :: OnRtnInsCombinationLeg** | Public Stream |
| Spread Instrument Deletion | **CShfeFtdcTraderSpi :: OnRtnDelCombinationLeg** | Public Stream |
| Alias Definition Notification | **CShfeFtdcTraderSpi :: OnRtnAliasDefine** | Public Stream |
| Advisory Notice | **CShfeFtdcTraderSpi :: OnRtnBulletin** | Public Stream |
| Query | Member Cash Fund Query | **CShfeFtdcTraderApi :: ReqQryPartAccount**  **CShfeFtdcTraderSpi :: OnRspQryPartAccount** | Query Stream |
| Order Query | **CShfeFtdcTraderApi :: ReqQryOrder**  **CShfeFtdcTraderSpi :: OnRspQryOrder** | Query Stream |
| Spread Order Query | **CShfeFtdcTraderApi :: ReqQryCombOrder**  **CShfeFtdcTraderSpi :: OnRspQryCombOrder** | Query Stream |
| RFQ Query | **CShfeFtdcTraderApi :: ReqQryQuote**  **CShfeFtdcTraderSpi :: OnRspQryQuote** | Query Stream |
| Trade Query | **CShfeFtdcTraderApi :: ReqQryTrade**  **CShfeFtdcTraderSpi :: OnRspQryTrade** | Query Stream |
| Client Query | **CShfeFtdcTraderApi :: ReqQryClient**  **CShfeFtdcTraderSpi :: OnRspQryClient** | Query Stream |
| Member’s Position Query | **CShfeFtdcTraderApi :: ReqQryPartPosition**  **CShfeFtdcTraderSpi :: OnRspQryPartPosition** | Query Stream |
| Client’s Position Query | **CShfeFtdcTraderApi :: ReqQryClientPosition**  **CShfeFtdcTraderSpi :: OnRspQryClientPosition** | Query Stream |
| Instrument Query | **CShfeFtdcTraderApi :: ReqQryInstrument**  **CShfeFtdcTraderSpi :: OnRspQryInstrument** | Query Stream |
| Instrument Status Query | **CShfeFtdcTraderApi :: ReqQryInstrumentStatus**  **CShfeFtdcTraderSpi ::OnRspQryInstrumentStatus** | Query Stream |
| Hedge Quota Query | **CShfeFtdcTraderApi :: ReqQryHedgeVolume**  **CShfeFtdcTraderSpi :: OnRspQryHedgeVolume** | Query Stream |
| Market Data Query | **CShfeFtdcTraderApi :: ReqQryMarketData CShfeFtdcTraderSpi :: OnRspQryMarketData** | Query Stream |
| Advisory Query | **CShfeFtdcTraderApi :: ReqQryBulletin**  **CShfeFtdcTraderSpi :: OnRspQryBulletin** | Query Stream |
| Order Book Query | **CShfeFtdcTraderApi :: ReqQryMBLMarketData**  **CShfeFtdcTraderSpi ::OnRspQryMBLMarketData** | Query Stream |
| (Option) Exercise Query | **CShfeFtdcTraderApi::ReqQryExecOrder**  **CShfeFtdcTraderSpi::OnRspQryExecOrder** | Query Stream |
| Exchange Rate Query | **CShfeFtdcTraderApi::ReqQryExchangeRate**  **CShfeFtdcTraderSpi::OnRspQryExchangeRate** | Query Stream |
| Information Query | **CShfeFtdcTraderApi::ReqQryInformation**  **CShfeFtdcTraderSpi::OnRspQryInformation** | Query Stream |
| Abandoned Exercise Query | CShfeFtdcTraderApi:: ReqQryAbandonExecOrder  CShfeFtdcTraderSpi:: OnRspQryAbandonExecOrder | Query Stream |

## Business Interfaces Not Currently Supported

| **Business Type** | **Business Name** | **Request Interface / Response Interface** | **Opening Status** |
| --- | --- | --- | --- |
| Trading | New Order | **CShfeFtdcTraderApi ::ReqOrderInsert**  **CShfeFtdcTraderSpi ::OnRspOrderInsert** | Partially open |
| Order Action | **CShfeFtdcTraderApi ::ReqOrderAction**  **CShfeFtdcTraderSpi ::OnRspOrderAction** | Partially open |
| Spread Order Entry | **CShfeFtdcTraderApi ::ReqCombOrderInsert**  **CShfeFtdcTraderSpi ::OnRspCombOrderInsert** | Not open |
| RFQ Entry | **CShfeFtdcTraderApi ::ReqQuoteInsert**  **CShfeFtdcTraderSpi ::OnRspQuoteInsert** | Not open |
| RFQ Action | **CShfeFtdcTraderApi ::ReqQuoteAction**  **CShfeFtdcTraderSpi ::OnRspQuoteAction** | Not open |
| (Option) Exercise Entry | **CShfeFtdcTraderApi ::ReqExecOrderInsert**  **CShfeFtdcTraderSpi ::OnRspExecOrderInsert** | Not open |
| (Option) Exercise Action | **CShfeFtdcTraderApi ::ReqExecOrderAction**  **CShfeFtdcTraderSpi ::OnRspExecOrderAction** | Not open |
| Abandon Exercise | CShfeFtdcTraderApi::ReqAbandonExecOrderInsert  CShfeFtdcTraderSpi::OnRspAbandonExecOrderInsert | Not open |
| Abandon Exercise Action | CShfeFtdcTraderApi::ReqAbandonExecOrderAction  CShfeFtdcTraderSpi::OnRspAbandonExecOrderAction | Not open |
| Price Quote??? | CShfeFtdcTraderApi:: ReqQuoteDemand  CShfeFtdcTraderSpi:: OnRspQuoteDemand  CShfeFtdcTraderSpi::OnRtnQuoteDemandNotify | Not open |
| Return | Execution Report of Spread  Order | CShfeFtdcTraderSpi ::OnRtnCombOrder | Not open |
| Execution Report of RFQ | CShfeFtdcTraderSpi ::OnRtnQuote | Not open |
| Execution Report of Exercise | CShfeFtdcTraderSpi ::OnRtnExecOrder | Not open |
| Spread Order Error Return | CShfeFtdcTraderSpi ::OnErrRtnCombOrderInsert | Not open |
| RFQ Error Return | CShfeFtdcTraderSpi ::OnErrRtnQuoteInsert | Not open |
| RFQ Action  Error Return | CShfeFtdcTraderSpi ::OnErrRtnQuoteAction | Not open |
| (Option) ExerciseEntry Error Return | CShfeFtdcTraderSpi ::OnErrRtnExecOrderInsert | Not open |
| (Option) Exercise Action Error Return | CShfeFtdcTraderSpi ::OnErrRtnExecOrderAction | Not open |
| Public Notification | New Spread Instrument Notification | CShfeFtdcTraderSpi ::OnRtnInsCombinationLeg | Not open |
| Spread Instrument Deletion Notification | CShfeFtdcTraderSpi ::OnRtnDelCombinationLeg | Not open |
| Inquiry | Spread Order Query | CShfeFtdcTraderApi ::ReqQryCombOrder  CShfeFtdcTraderSpi ::OnRspQryCombOrder | Not open |
| Exercise Query | CShfeFtdcTraderApi::ReqQryExecOrder  CShfeFtdcTraderSpi::OnRspQryExecOrder | Not open |
| Exchange Rate Query | CShfeFtdcTraderApi::ReqQryExchangeRate  CShfeFtdcTraderApi::OnRspQryExchangeRate | Not open |
| Abandoned Exercise Query | CShfeFtdcTraderApi::ReqQryAbandonExecOrder  CShfeFtdcTraderSpi::OnRspQryAbandonExecOrder | Not open |

# TraderAPI Reference Manual



## CShfeFtdcTraderSpi Interface

**CShfeFtdcTraderSpi** is an event handler interface. Client applications must implement this interface to handle events of interest.

### OnFrontConnected Method

This callback is invoked to handle responses from the Exchange after the member system connects to the Exchange. This connection is managed automatically by API.

**Method Signature:**

**void OnFrontConnected()；**

Note: The invocation of **OnFrontConnected** indicates that a TCP connection has been established successfully, and login is necessary before carrying out any business operation.

### OnFrontDisconnected Method

This callback is invoked to handle TCP disconnection between the member system and the Exchange. When disconnection occurs, API will attempt to reconnect automatically without involvement from the member system. API may choose the IP address of the original registered gateway or the other network address. The gateway selection is automatically handled by API.

**Method Signature:**

**void OnFrontDisconnected (int nReason)；**

**Parameter：**nReason: disconnection reason

* 0x1001 network reading failure
* 0x1002 network writing failure
* 0x2001 receiving heartbeat times out
* 0x2002 sending heartbeat times out
* 0x2003 receiving error message

### OnHeartBeatWarning Method

This callback is invoked to handle warnings issued from the Exchange when a heartbeat is not received for a preset period of time. The default warning timeout value is 10 seconds. If a customized timeout value is set through **SetHeartbeatTimeout (unsigned int timeout),** then the warning timeout is **timeout**/2.

**Method Signature:**

**void OnHeartBeatWarning(int nTimeLapse)；**

**Parameter:** nTimeLapse: time elapsed since last message was received (in seconds)

### OnPackageStart Method

This callback is invoked to handle the notification of the arrival of a package returned from the Exchange after a request was performed. The member system should open and start processing a package that may contain multiple responses.

**Method Signature:**

**void OnPackageStart (int nTopicID, int nSequenceNo)；**

**Parameters:**

**nTopicID**: Topic ID(e.g. private stream, public stream, market data stream)

**nSequenceNo**: Message Sequence Number

### On Package End Method

This callback is invoked to handle the notification of completion of a transmitted package. It pairs with OnPackageStart, which marks the arrival of a package.

**Function Signature:**

**void OnPackageEnd (int nTopicID, int nSequenceNo)；**

**Parameters:**

**nTopicID**: Topic ID(e.g. private stream, public stream, market data stream)

**nSequenceNo**: Message Sequence Number

### OnRspUserLogin Method

This callback is invoked to handle responses from the Exchange regarding a client’s login request.

**Method Signature:**

**void OnRspUserLogin(**

**CShfeFtdcRspUserLoginField \*pRspUserLogin,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspUserLogin**: points to the object containing the client’s login request, with the structure shown below:

struct CShfeFtdcRspUserLoginField {

///trade date

TShfeFtdcDateType TradingDay;

///successful login time

TShfeFtdcTimeType LoginTime;

///Maximum order local ID

TShfeFtdcOrderLocalIDType MaxOrderLocalID;

///Trader ID

TShfeFtdcUserIDType UserID;

/// Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Member System Name

TShfeFtdcTradingSystemNameType TradingSystemName;

///Data Center ID

TShfeFtdcDataCenterIDType DataCenterID;

///current length of the Member’s private stream

TShfeFtdcSequenceNoType PrivateFlowSize;

///current length of the trader-specific private stream

TShfeFtdcSequenceNoType UserFlowSize;

///Transaction day

TShfeFtdcDateType ActionDay;

};

**Note**: if member system is to maintainthe sequence number for purpose of message replay, it should save the returned value in the field of TradingDay and DataCenterID so as to fill these values in its next login attempt.

**Note:** ActionDay is a newly added field containing the transaction date. Its value will is set empty before this field is opened to the public. Trading date should be retrieved from the field of TradingDay.

**pRspInfo**: points to the object containing the execution status indicating if a client’s login request was successfully executed.

**Special attention**: If there are multiple responses to a user’s login request, the first response always contains the status information, but the subsequent responses may contain NULL.

The structure is shown below::

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible cause |
| 3 | Member cannot be found | InvalidMember ID at login |
| 45 | Invalid Settlement Group initialization Status | Trading System initialization is not yet completed. Please try again later |
| 59 | Repetitive Login Request | The user has logged in already |
| 60 | Invalid user ID or password | User ID or password is invalid |
| 62 | User Account Locked | User’s account is locked by the Exchange |
| 64 | User does not belong to the Member | Member ID is invalid |
| 65 | Invalid IP Address | The IP address of the member system is not recognizable by the Exchange |
| 100 | Invalid User Type | Non-trading user attempted to log in the Exchange |

**nRequestID**: requests the IDspecified by the member system in its login request

**bIsLast**: indicates if this is the last response with respect to the nRequestID

### OnRspUserLogout Method

This callback is invoked to handle responses from the Exchange regarding a client’s logout request.

**Method Signature:**

**void OnRspUserLogout(**

**CShfeFtdcRspUserLogoutField \*pRspUserLogout,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspUserLogout**: a points to the object contains the user’s logout request, with the structure below:

struct CShfeFtdcRspUserLogoutField {

///User ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

};

**pRspInfo**: a points to the object containing a message indicating if the logout request was successfully executed, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 66 | User not logged in yet | User has not logged in yet |
| 67 | Mismatched login ID | User ID in the logout request does not match the one in the login request |
| 68 | MismatchedMember ID | Member ID in the logout request does not match the one in the login request |

**nRequestID**: requests the ID specified by the member systemsystem in its logout request

**bIsLast**: indicates if the current response is the last one with respect to the nRequestID

### OnRspUserPasswordUpdate Method

This callback is invoked to handle responses from the Exchange regarding a client’s password update request.

**Method Signature:**

**void OnRspUserPasswordUpdate(**

**CShfeFtdcUserPasswordUpdateField \*pUserPasswordUpdate,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pUserPasswordUpdate**: a points to the object containing a user’s password update request, with the structure below:

struct CShfeFtdcUserPasswordUpdateField {

///TraderID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Old password

TShfeFtdcPasswordType OldPassword;

///New password

TShfeFtdcPasswordType NewPassword;

};

**pRspInfo**: a pointer containing information indicating if a user’s password update request was successfully executed, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 58 | Mismatched user | The user requesting a password update does not match the user that is logged in |
| 60 | Invalid user name or password | Either the user name or password is wrong |
| 66 | User not logged in yet | User has not logged in yet |
| 68 | Mismatched Member | The member requesting a password update does not match the member that is logged in |

**nRequestID**: requests the ID specified by the member system in its password update request

**bIsLast**: indicates if the current response is the latest with respect to the nRequestID

### OnRspSubscribeTopicMethod

This callback is invoked to handle responses from the Exchange about a client’s request to subscribe to a topic.

**Method Signature:**

**void OnRspSubscribeTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pDissemination**: a points to an object containing the subscribed topic, with the structure below:

struct CShfeFtdcDisseminationField {

///topic ID

TShfeFtdcSequenceSeriesType SequenceSeries;

///starting sequence number

TShfeFtdcSequenceNoType SequenceNo;

};

**pRspInfo**: a points to an object containing a message indicating if the request was successfully executed, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 66 | User not logged in yet | User not logged in yet |

**nRequestID**: requests the ID specified by the member system in its topic subscription request

**bIsLast**: indicates if the current response is the latest with respect to the nRequestID

### OnRspQryTopicMethod

This callback is invoked to handle query results that have returned from the Exchange.

**Method Signature:**

**void OnRspQryTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pDissemination**：points to the object containing topic query request, with the structure below:

struct CShfeFtdcDisseminationField {

///topic id

TShfeFtdcSequenceSeriesType SequenceSeries;

///total number of messages in the returns

TShfeFtdcSequenceNoType SequenceNo;

};

**pRspInfo**：points to the object containing a message indicating whether or not a user’s request was successfully executed, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 66 | User not logged in yet | User not logged in yet |

**nRequestID**: requests the ID specified by the member system in its topic query request

**bIsLast**：indicates if the current response is the latest with respect to the nRequestID

### OnRspError Method

This callback is invoked to handle error response from the Exchange during the execution of any request.

**Method Signature:**

**void OnRspError(**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pRspInfo**：a container to the object containing the error message with the structure below:

struct CShfeFtdcRspInfoField {

///

TShfeFtdcErrorIDType ErrorID;

///

TShfeFtdcErrorMsgType ErrorMsg;

};

Possible Errors：

**Error Code Error Msg Possible Causes**

1 Not Login Not Login yet

Too High FTD Version Too High FTD Version

Invalid interface product info Invalid interface product info

Unrecognized ftd tid Invalid FTD package head

997 api authentication failure Illegal api usage

998 query frequency is too high query frequency is too high

999 the last query result is on way the last query result is not sent yet

**nRequestID**：requests the ID maintained by the member system in its request

**bIsLast**：indicates whether or not the current response is the latest with respect to the nRequestID

### OnRspOrderInsert Method

This callback is invoked to handle acknowledgements from the Exchange about a client’s request to add a new order entry.

**Method Signature:**

**void OnRspOrderInsert(**

**CShfeFtdcInputOrderField \*pInputOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameter:**

**pInputOrder**：points to the object containing information about the requested new order entry and the sequence number assigned to this order entry by the Exchange. Note that some fields may contain a null value, which is different from the value when this order entry was submitted. The object is structured as below:

struct CShfeFtdcInputOrderField {

///Sequence number assigned by the Exchange

TShfeFtdcOrderSysIDType OrderSysID;

///Trading Member Code, **not used[[1]](#footnote-1)**

TShfeFtdcParticipantIDType ParticipantID;

///Client ID, **not used**

TShfeFtdcClientIDType ClientID;

///Trader ID, **not used**

TShfeFtdcUserIDType UserID;

///Instrument ID,**not used**

TShfeFtdcInstrumentIDType InstrumentID;

///Order type, **not used**

TShfeFtdcOrderPriceTypeType OrderPriceType;

///Buy or sell, **not used**

TShfeFtdcDirectionType Direction;

///Open or close position, **not used**

TShfeFtdcCombOffsetFlagType CombOffsetFlag;

///Hedgeor speculation, **not used**

TShfeFtdcCombHedgeFlagType CombHedgeFlag;

///Limit Price,**not used**

TShfeFtdcPriceType LimitPrice;

///quantity,**not used**

TShfeFtdcVolumeType VolumeTotalOriginal;

///validity type, **not used**

TShfeFtdcTimeConditionType TimeCondition;

///GTD date,**not used**

TShfeFtdcDateType GTDDate;

///Match quantity type, **not used**

TShfeFtdcVolumeConditionType VolumeCondition;

///Minimum quantity, **not used**

TShfeFtdcVolumeType MinVolume;

///Trigger condition, **not used**

TShfeFtdcContingentConditionType ContingentCondition;

///Trigger price,**not used**

TShfeFtdcPriceType StopPrice;

///Enforced close reasons,**not used**

TShfeFtdcForceCloseReasonType ForceCloseReason;

///local order ID

TShfeFtdcOrderLocalIDType OrderLocalID;

///Whether or not to automatically suspend the order when unexpected disconnection occurs, **not used**

TShfeFtdcBoolType IsAutoSuspend;

///business unit, **not used**

TShfeFtdcBusinessUnitType BusinessUnit;

///local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the object containing the transaction status indicating if the request to add a new order was successful. The object is structured as below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 2 | Contract cannot be located | Unable to find the contract requested in the order |
| 3 | Member cannot be located | Unable to find the member requested in the order |
| 4 | Client cannot be located | Unable to find the client requested in the order |
| 6 | Invalid value to the field | Supplied Value is invalid to the field such as the value is out of permitted range, or reason of enforced offset is specified for a non-enforced-offset order. |
| 12 | Duplicate order | Local ID of the order is duplicated |
| 15 | Client has no account with the claimed member | The client has no account with the specified member |
| 16 | IOC order entered duringa non-continuous trading session | User is to add an IOC (immediately-matched-or-cancelled) order during a non-continuous trading session |
| 17 | GFA order entered during a non-auction session | GFA order is attempted at a non-auction session |
| 18 | Market order cannot be accepted into order book | The time condition for the entered market order is not IOC |
| 19 | Volume restriction is used with a non-IOC order | The entered order does not have value in IOC time conditions (meaning it is a non-IOC order) but specifies a value in volume restriction. |
| 20 | GTD order is expired | The GTD date in the GTD order has expired |
| 21 | The minimum volume is greater than the order volume | The order’s minimum quantity is larger than the order quantity |
| 22 | Exchange data is not synchronized | The Trading System is not completely initialized. Please try again later. |
| 23 | Settlement group data is not synchronized | Initialization of the Trading System is incomplete. Please try again later. |
| 26 | This operation is forbidden under the current status | Instrument status does not allow trading or auction |
| 31 | User is trying to offset a position that is insufficient | Client has attempted to offset a position that exceeds the position account |
| 32 | Client’s position exceeds his/her limit | Client has attempted to enter a new order (adding position), that exceeds the permitted speculation limit. |
| 34 | Member’s position exceeds his/her limit | Client has attempted to enter a new order (adding position) which exceeds the member’s permitted position limit |
| 35 | Unable to find the account | Unable to find the cash account claimed in the order |
| 36 | Insufficient funds | Insufficient margin |
| 37 | Illegal quantity | Order quantity is not an integral multiple of the minimum quantity, or exceeds the maximum order quantity |
| 48 | Price is not a multiple of the minimum unit | Order price is not an integral multiple of the tick |
| 49 | Price exceeds the upper limit | Order price exceeds the upper limit of the instrument |
| 50 | Price is below the lower limit | Order price is lower than the lower limit of the instrument |
| 51 | No trading permission | Member, client, or trader does not have permission to trade the selected instrument |
| 52 | Order ofclosingposition only | Member, client, or trader has permission to enter an order for close position only |
| 53 | Insufficient privilege | Member does not have privileges that are mapped to the role of the client |
| 57 | Cannot operate for other members | Trader attempts to operate for other members that he does not belong to |
| 58 | Unmatched user | Trader specified in the order does not match trader who logged in |
| 66 | User is not logged in | User has not logged in yet |
| 78 | GTD date not set in the GTD order | GTD order does not have GTD date specified |
| 79 | Order type not supported | Exchange does not support the specified order type |
| 83 | Stop order for continuous trading mode only | Stop order is entered in non-continuous trading session |
| 84 | Stop-loss order has to be IOC/GFD | Time condition is neither IOC nor GFD for the stop order |
| 95 | Stop order should specify trigger price | The stop order does not specify the trigger price |
| 96 | Insufficient hedge quota | Client does not have sufficient hedge quota left for the entered hedge order |
| 103 | Cannot close hedge position made on current trade date | Hedge position cannot be closed by ‘Close Current Day Order’ |
| 114 | Market order not accepted | The time condition for the entered market order is not IOC |

**nRequestID**：requests the ID specified by the member system in its order entry request

**bIsLast**：indicates whether the current response is the latest with respect to the nRequestID.

**Note：**

If **CShfeFtdcRspInfoField.ErrorID** contains zero, it means that the order has been accepted successfully by the Exchange. In **ShfeFtdcInputOrderField \*pInputOrder**, only the order sequence number (assigned by the Exchange) and local order sequence number are meaningful, indicating the interrelationship of this order between the Exchange and the member system. The details of the order execution report need to be retrieved from the private stream.

Please refer to **OnRtnOrder**for the description of each data field in **CshfeFtdcInputOrderField.**

### OnRspOrderAction Method

This callback is invoked to handle execution reports of order actions, including order cancellations, order suspensions, order activation, and order modification.

**The order modification function is not supported in the current version.**

**Function Signature：**

**void OnRspOrderAction(**

**CshfeFtdcOrderActionField \*pOrderAction,**

**CshfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters：**

**pOrderAction**：points to the object storing the order action request, and the order sequence number assigned by the Exchange system. Note that certain fields in the returned order action request may be empty, which is different from the value set in the original order action request. This object is structured as below:

struct CshfeFtdcOrderActionField {

///Order Number

TShfeFtdcOrderSysIDType OrderSysID;

///Local Order Sequence Number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag of Order action

TshfeFtdcActionFlagType ActionFlag;

///Member’s code,**not used**

TShfeFtdcParticipantIDType ParticipantID;

///Client’s code, **not used**

TShfeFtdcClientIDType ClientID;

///Trader code

TShfeFtdcUserIDType UserID;

///Price,**not used**

TshfeFtdcPriceType LimitPrice;

/// Quantity Change, **not used**

TshfeFtdcVolumeType VolumeChange;

/// Local sequence number for this Operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit,**not used**

TshfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TshfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the data containing the transaction status regarding whether or not the requested action was executed successfully:

|  |  |  |
| --- | --- | --- |
| struct CshfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TshfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 3 | Member cannot be found | Member cannot be found in the order operation |
| 4 | Client cannot be found | Client cannot be found in the order operation |
| 8 | Invalid value in field | Invalid value is specified in a certain field (not in the pre-defined enumeration values). |
| 15 | Client didn’t open an account with this member | Client didn’t open an account with the designated member |
| 22. | The Exchange’s data is not yet fully synchronized | Initialization is not completed in the trading system. Please try again later. |
| 23 | The settlement group data is not fully synchronized | Initialization of trading system is not completed. Please try again later. |
| 24. | Order cannot be found | Order to be operated cannot be found |
| 26. | This operation is prohibited at the current trading status | Order activation is permitted in the continuous trading status and auction mode (including new order entry for auction and opening price calculation); Other types of order operation are permitted in the continuous trading status and new order entry for auction |
| 28 | Order already filled | Order has already been filled |
| 29 | Order cancelled | Order has already been cancelled |
| 32 | Exceeds client's position limit | The modification will cause client to exceed his/her approved speculation position limit |
| 34 | Exceeds member's position limit | The modification will cause member to exceed his/herapprovedposition limit |
| 35. | Account not found | Margin account is not found |
| 36 | Insufficient funds | Insufficient funds in margin account |
| 37. | Invalid quantity | Order modification will result in invalid quantity, which is either not an integral multiple of the minimum quantity, or exceeds the maximum quantity |
| 48 | Price not integral multiples of the tick | The order modification will result in a price that is not an integral multiple of the tick |
| 49. | Price exceeds the upper limit | The order modification will cause the price to exceeds the order’s upper limit |
| 50 | Price drops below the lower limit | The order modification will cause the price to drop below the lower limit |
| 57 | Trader attempts to operate for a non-member | The trader attempts to perform an operation for a member s/he does not belong to |
| 58 | Unmatched user | The trader in the order operation does not match the one at the time of login |
| 66 | User hasn't logged in yet | The user hasn't logged in yet |
| 76 | Order already suspended | The order has already been suspended when the user requests order suspension. |
| 77 | Order already activated | The order has already been activated when the user requests order activation |
| 96 | Insufficient hedge quota | The client's hedge quota is insufficient after modifying the order |
| 97 | Repeated operation | The local operation number is duplicate |
| 99 | Trader to perform unprivileged operation | Trader A attempts to perform an unprivileged operation for Trader B, who belongs to the same member as A |

**nRequestID**：requests the IDspecified by the member system in its order action request

**bIsLast**：indicates whether or not this response is the latest with respect to the nRequestID

### OnRspQuoteInsert Method

**This method is not currently supported.**

This callback is invoked to handle responses from the trading system, after a user submits a request to add a new quote entry.

**Function Signature：**

**void OnRspQuoteInsert(**

**CShfeFtdcInputQuoteField \*pInputQuote,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters：**

**pInputQuote**：points to a new quote entry, structured as below:

struct CShfeFtdcInputQuoteField {

///Quote Number

TShfeFtdcQuoteSysIDType QuoteSysID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Quantity

TShfeFtdcVolumeType Volume;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

/// Local quotes Number

TShfeFtdcQuoteLocalIDType QuoteLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Flag indicating whether to open or closepositionon buy-side

TShfeFtdcCombOffsetFlagType BidCombOffsetFlag;

///Flag indicating whether to hedge or speculationon buy-side

TShfeFtdcCombHedgeFlagType BidCombHedgeFlag;

///Bid price

TShfeFtdcPriceType BidPrice;

///Flag indicating whether to open or close position on sell-side

TShfeFtdcCombOffsetFlagType AskCombOffsetFlag;

///Flag indicating whether to hedge or speculation on sell-side

TShfeFtdcCombHedgeFlagType AskCombHedgeFlag;

///Offer price

TShfeFtdcPriceType AskPrice;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the object containing the message indicating if the request was successfully transacted. Its structure is shown below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 2 | Contract cannot be found | The specified contract in the quote request cannot be found. |
| 3 | Member cannot be found | The specified member in the quote request cannot be found. |
| 4 | Client cannot be found | The specified client in the quote request cannot be found. |
| 8 | Invalid value in field | Invalid value in the field (e.g. the value is out of range of permitted constants). |
| 13 | Duplicate quote | The local quote number is duplicate |
| 15 | Client didn't open an account with the member | The client specified in the quote does not have an account at the designated member |
| 22. | The trading data is not synchronized yet | Initialization of the trading system is not complete. Please try again later. |
| 23 | The settlement group's data is not synchronized yet | Initialization of the trading system is not completed. Please try again later. |
| 26. | This operation is prohibited in the current status | The contract is not in a continuous trading or auction status, therefore the operation is not permitted |
| 31. | Insufficient position to be closed | The client does not have sufficient position to be closed. |
| 32 | Exceeds client's position limit | This quote has caused the client’s speculation position to exceed his/her approved limit |
| 34 | Exceeds member's position limit | This quote has caused the member's position to exceed his/her approved limit |
| 35. | Account cannot be found | The margin account associated with the quote entry cannot be found |
| 36 | Inadequate funds | In sufficient funds in the margin account |
| 37. | Invalid quantity | The specified quantity in this quote entry is either not an integral multiple of the minimum quantity or exceeds the maximum quantity |
| 48 | Price not on tick | The quoted price is not an integral multiple of the contract's tick size |
| 49. | Price exceeds upper limit | The quoted price is higher than the contract's upper limit |
| 50 | Price below lower limit | The quoted price is lower than the contract's lower limit |
| 51 | Unauthorized action | Either the trader or client does not have authorization to perform the requested transaction or operation on the contract |
| 52 | Authorized to close a position only | The user or member has authorization to close position on the specified contract only |
| 53. | No such trading role | On the designated contract, the member does not have the trading role mapped to the client |
| 57 | Cannot perform for another member | The trader is to perform a request for a member that she/he does not belong to |
| 58 | Unmatched user | The trader in the quote does not match with trader at the time of login |
| 66 | User hasn't logged in yet | User hasn't logged in yet |
| 79 | Order type is not supported | The Exchange does not support this order type. |
| 96 | Insufficient hedge quota | The client's hedge quota is insufficient |
| 103. | Current-day hedge position cannot be closed out | The hedge positions cannot be closed by the current-day quote (Note: this validation is related to a configurable flag that permits whether or not the current-day position can be offset. This flag is set to TRUE in most conditions) |

**nRequestID**：requests the ID specified by the member system in its quote entry request

**bIsLast**：Indicates whether this response is the latest with respect to the nRequestID

### OnRspQuoteAction Method

**This function is not currently supported.**

This callback is invoked to handle responses for quote-related operations, including cancellation, suspension, activation, and modification.

**Function signature：**

**void OnRspQuoteAction(**

**CShfeFtdcQuoteActionField \*pQuoteAction,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pQuoteAction**：points to the object containing the data in the original quote action request, as well as the sequence number allocated by the trading system:

struct CShfeFtdcQuoteActionField {

///Quote number, assigned by the Exchange

TShfeFtdcQuoteSysIDType QuoteSysID;

///Local quote number, maintained by the member system

TShfeFtdcOrderLocalIDType QuoteLocalID;

///Flag indicatingaction type

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///ID of this action maintained by client’s system

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the object containing the transaction status of this request, indicating if this request was successfully executed by the trading system. Its structure is shown below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Possible reason |
| 3 | Member not found | Member cannot be found in the quote operation request. |
| 4 | Client not found | Client cannot be found in the quote operation request |
| 8 | Invalid value | Value filled in the request is invalid (not permitted for the field). |
| 15 | Client does not have an account with this member | Client has not opened an account with the designated member |
| 22. | The trading data not synchronized | Initialization of the trading system is not complete. Please try again later. |
| 23 | The settlement group's data not synchronized | Initialization of the trading system is not complete. Please try again later. |
| 25. | Quote not found | The quote cannot be found |
| 26. | This operation is prohibited at the current trading status | Request for activation is not permitted when the contract is in neither trading nor auction status. |
| 28 | The order has already been filled | The order derived from the quote has already been filled |
| 35 | Account not found | The margin account cannot be found |
| 36 | Inadequate funds | In sufficient funds in the margin account |
| 57 | Operation not permitted for other members | Trader conducts operation on behalf of member to whom s/he does not belong |
| 58 | Unmatched user | The trader in the quote operation does not match the trader at the time of login |
| 66 | User hasn't logged in yet | User hasn't logged in yet |
| 70 | Quote has already been cancelled | Quote has already been cancelled |
| 97 | Duplicate request | Local operation number in the quote request is duplicate |
| 99 | Operation not permitted for this trader | Trader is attempting to perform an unauthorized operation |

**nRequestID**：requests the ID maintained by the member system in its quote action request

**bIsLast**：Indicating whether this response is the latest with respect to the nRequestID

### OnRspExecOrderInsert Method

**This method is not currently supported.**

This callback is invoked to handle execution reports from the Exchange about adding a new exercise entry.

**Function Signature：**

**void OnRspExecOrderInsert(**

**CShfeFtdcInputExecOrderField \*pInputExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID, bool bIsLast)；**

**Parameters：**

**pInputExecOrder**：points to the object containing the exercise entry, with the structure below:

struct CShfeFtdcInputExecOrderField {

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

/// Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///TraderID

TShfeFtdcUserIDType UserID;

/// Local exercise sequence number

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Quantity

TShfeFtdcVolumeType Volume;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Opening or Closing (offset) position

TShfeFtdcOffsetFlagType OffsetFlag;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the object containing the transaction status, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  ///ErrorID  TShfeFtdcErrorIDType ErrorID;  ///Error Message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed as below: | | |
| Error ID | Error message | Cause |
| 2 | Contract not found | Specified contract cannot be found. |
| 3 | Member not found | Specified member cannot be found. |
| 4 | Client not found | Specified client cannot be found. |
| 15 | Client didn't open an account with this member | Client didn't open an account with the claimed member |
| 22 | The trade data not synchronized | Initialization of the trading system is not complete. Please try again later. |
| 23 | Settlement group data is not synchronized | Initialization of the trading system is not complete. Please try again later. |
| 26 | This operation is prohibited in the current status | The contract is in closed status |
| 51 | Not authorized to trade | Not authorized to trade the designated contract/Client is not authorized to trade the designated contract/Trader is not permitted to trade |
| 53 | No such trading role | The member does not have a trading role mapped to this client on the designated contract |
| 57 | Operation not permitted for other members | The trader is attempting to perform an operation for a member that s/he does not belong to |
| 58 | Unmatched user | The specified trader does not match with the trader at the time of login |
| 66 | User hasn't logged in yet | User hasn't logged in yet |
| 79 | Order type not supported | The specified order type is not supported |
| 89 | Invalid value in the field | Value in the field is not permitted |
| 91 | Duplicate Exercise | The local sequence number for the submitted exercise is duplicate |
| 94 | Exercise is for option only | The specified contract is not an option contract |

**nRequestID**：requests the ID specified by the member system in its exercise entry request

**bIsLast**：Indicates whether or not this response is the latest with respect to the nRequestID

### OnRspExecOrderAction Method

**This method is not currently supported by the business.**

This callback is invoked to handle execution reports returned from the Exchange regarding actions on existing exercises.

**Function Signature：**

**void OnRspExecOrderAction(**

**CShfeFtdcExecOrderActionField \*pExecOrderAction,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pInputExecAction**：points to the object containing an acknowledged exercise entry. The structure of this object is shown below:

struct CshfeFtdcExecOrderActionField {

///Exercisenumber

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Local ID of the exercise

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Flag indicating action type

TshfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///TraderID

TShfeFtdcUserIDType UserID;

///Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TshfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TshfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：an object containing the transaction status with respect to the request. Its structure is shown below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 2 | Contract not found | The contract in the specified execution cannot be found |
| 3 | Member not found | The specified member cannot be found |
| 4 | Client not found | The specified client cannot be found. |
| 15 | Client has not opened an account under this member | The specified client has not opened an account under the designated member |
| 22. | The trading data is not synchronized | Data initialization in the trading system is not complete. Please try again later. |
| 23 | The settlement group data is not synchronized | Data initialization in the trading system is not complete. Please try again later. |
| 26. | This operation is prohibited in the current status | The contract is closed |
| 51 | Not authorized to trade | Not authorized to trade in the designated contract, or the client /trader is not authorized to trade in the designated contract. |
| 53. | No such trading role | The member does not have the correct trading role to trade in the designated contract. |
| 57 | Trader is not permitted to perform this action for a non-member | Trader cannot request an operation for member s/he does not belong to. |
| 58 | Unmatched user | The specified trader does not match with the trader at the time of login |
| 66 | User not logged in yet | User hasn't logged in yet |
| 79 | Order type is not supported | The specified order type is not supported. |
| 90 | Invalid value for this field | The value in the field is not permitted |
| 92 | Exercise cancelled | The specified exercise has been cancelled |
| 93 | Exercise not found | The specified exercise cannot be found |
| 97 | Duplicate operation | The local sequence number for this action is duplicate |

**nRequestID**：requests the ID specified by the member system in its exercise action request

**bIsLast**：indicates whether this response is the latest with respect to the nRequestID.

### OnRspAdminOrderInsert Method

**The efficacy of the credit limit setting in this function will be determined by the Exchange with further notice.**

This callback is invoked to handle responses returned from the Exchange system for inserting administrative orders.

**Function Signature:**

**void OnRspAdminOrderInsert(**

**CShfeFtdcInputOrderField \*pInputAdminOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pInputAdminOrder**：points to the object containing administrative order, with the structure below:

struct CShfeFtdcInputAdminOrderField {

/// Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///administrative order command

TShfeFtdcAdminOrderCommandFlagType AdminOrderCommand;

/// clearing member (participant) ID

TShfeFtdcParticipantIDType ClearingPartID;

/// Member code

TShfeFtdcParticipantIDType ParticipantID;

/// amount

TShfeFtdcMoneyType Amount;

/// Settlement group code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

};

**pRspInfo**：points to the object containing response from the Exchange, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  Possible errors are listed below:  **Error Code Error Message Possible Cause** | | |
| 3 | Member not found | The specified member cannot be found. |
| 23 | Clearing data not in sync | The data initialization in the trading system is not complete. Please try again later. |
| 56 | Operation permitted for admin only | The user is not entitled to this operation. |
| 64 | The trading member does not match the clearing member | The trading member does not associate with the clearing member in the given transaction. |
| 108 | Clearing member-specific operation | The clearing member is entitled to set credit limit for the trading member. |
| 109 | Trading/clearing member mismatch | Trading/clearing member does not match. |
| 110 | Field error | The field contains an invalid value. |
| 113 | Credit limit not initialized | The matching record cannot be found during credit limit update or deletion. |

**nRequestID**：requests the ID designated by the member system in its administrative order request.

**bIsLast**：indicates whether or not this is the last response to the nRequestID.

### OnRspQryPartAccount Method

This callback is invoked to handle query results from the Exchange regarding the margin account balance.

**Function Signature：**

**void OnRspQryPartAccount(**

**CShfeFtdcRspPartAccountField \*pRspPartAccount,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters：**

**pRspPartAccount**：points to the object containing the margin balance, with the structure below:

struct CShfeFtdcRspPartAccountField

{

/// Trading date

TShfeFtdcDateType TradingDay;

///Settlement group code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///margin balance at last settlement

TShfeFtdcMoneyType PreBalance;

///margin balance at current

TShfeFtdcMoneyType CurrMargin;

///Profit & loss from position close (offset)

TShfeFtdcMoneyType CloseProfit;

/// Option premium

TShfeFtdcMoneyType Premium;

///Deposit

TShfeFtdcMoneyType Deposit;

///Withdrawal

TShfeFtdcMoneyType Withdraw;

///Margin balance allocated for settlement

TShfeFtdcMoneyType Balance;

///Withdrawable funds

TShfeFtdcMoneyType Available;

///Margin account number

TShfeFtdcAccountIDType AccountID;

///Margin in frozen

TShfeFtdcMoneyType FrozenMargin;

///Premium in frozen

TShfeFtdcMoneyType FrozenPremium;

///Base reserve

TShfeFtdcMoneyType BaseReserve;

};

**pRspInfo**: containing a message indicating if the request was successfully executed, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User has no permission | User was requesting operation he is not authorized |
| 57 | User not permitted to perform for other members | User is not permitted to check balance for other members that he does not belong to |

**nRequestID**: requests the ID specified by the member system in its margin balance query request

**bIsLast**: indicates whether the current response is the latest with respect to the nRequestID

### OnRspQryOrder Method

This callback is invoked to handle query results returned from the Exchange for orders.

**Function Signature:**

**void OnRspQryOrder(**

**CShfeFtdcOrderField \*pOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pOrder**：points to the object containing a returned order, with the structure below:

struct CshfeFtdcOrderField {

///Trading Date

TShfeFtdcDateType TradingDay;

///Settlement Group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Order Number

TShfeFtdcOrderSysIDType OrderSysID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Order Type

TShfeFtdcOrderPriceTypeType OrderPriceType;

///buy or sell

TShfeFtdcDirectionType Direction;

///Open or close position

TShfeFtdcCombOffsetFlagType CombOffsetFlag;

///Hedge or speculation

TShfeFtdcCombHedgeFlagType CombHedgeFlag;

///Limit Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Order validity

TShfeFtdcTimeConditionType TimeCondition;

///GTD Date, **NOT USED**

TShfeFtdcDateType GTDDate;

///Matched quantity condition

TShfeFtdcVolumeConditionType VolumeCondition;

///Minimum matched quantity

TShfeFtdcVolumeType MinVolume;

///Trigger condidtion

TShfeFtdcContingentConditionType ContingentCondition;

///TriggerPrice, **NOT USED**

TShfeFtdcPriceType StopPrice;

///reason of enforced closing (offset)

TShfeFtdcForceCloseReasonType ForceCloseReason;

///Local order ID

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag indicating whether or not to automatically suspend the order when an unexpected disconnection occurs

TShfeFtdcBoolType IsAutoSuspend;

///Origin of order

TShfeFtdcOrderSourceType OrderSource;

///Order Status

TShfeFtdcOrderStatusType OrderStatus;

///Order Type

TShfeFtdcOrderTypeType OrderType;

///Traded volume on current trade date

TShfeFtdcVolumeType VolumeTraded;

///Remainingquantity

TShfeFtdcVolumeType VolumeTotal;

///Order entry date

TShfeFtdcDateType InsertDate;

///Order entry time

TShfeFtdcTimeType InsertTime;

///Activation time, **NOT USED**

TShfeFtdcTimeType ActiveTime;

///Suspension time, **NOT USED**

TShfeFtdcTimeType SuspendTime;

///Last update time

TShfeFtdcTimeType UpdateTime;

///Cancellation time

TShfeFtdcTimeType CancelTime;

///Trader ID of last update

TShfeFtdcUserIDType ActiveUserID;

///Priority, **NOT USED**

TShfeFtdcPriorityType Priority;

///Sequence number (sorted by time), **NOT USED**

TShfeFtdcTimeSortIDType TimeSortID;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Business unit, **NOT USED**

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Transaction date

TShfeFtdcDateType ActionDay;

};

**pRspInfo**: containing a message indicating the status of a transaction, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User is not authorized to perform this operation | User was requesting an operation s/he is not authorized to make. |
| 57 | Operation cannot be conducted by other members | User cannot query for a member s/he does not belong to. |

**nRequestID**：requests the ID specified by the member system in its order query request

**bIsLast**: indicates whether the current response is the latest with respect to the nRequestID

### OnRspQryQuote Method

**This method is not currently supported by the business.**

This callback is invoked to handle query results returned from the Exchange about quotes.

**Function Signature：**

**void OnRspQryQuote(**

**CShfeFtdcQuoteField \*pQuote,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters：**

**pQuote**：points to the object containing returned quote information, with the structure below:

struct CShfeFtdcQuoteField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Quote number

TShfeFtdcQuoteSysIDType QuoteSysID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Quantity

TShfeFtdcVolumeType Volume;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Local IDofquote

TShfeFtdcQuoteLocalIDType QuoteLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Open or close position on long side

TShfeFtdcCombOffsetFlagType BidCombOffsetFlag;

///Hedge or speculation on long side

TShfeFtdcCombHedgeFlagType BidCombHedgeFlag;

///Bid price

TShfeFtdcPriceType BidPrice;

///Open or close position on short side

TShfeFtdcCombOffsetFlagType AskCombOffsetFlag;

///Hedge or speculation on short side

TShfeFtdcCombHedgeFlagType AskCombHedgeFlag;

///Offer price

TShfeFtdcPriceType AskPrice;

///Quote entrytime

TShfeFtdcTimeType InsertTime;

///Quote cancel time

TShfeFtdcTimeType CancelTime;

///Trade time

TShfeFtdcTimeType TradeTime;

///Buy-sideorder number

TShfeFtdcOrderSysIDType BidOrderSysID;

///Sell-side order number

TShfeFtdcOrderSysIDType AskOrderSysID;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Transaction date

TShfeFtdcDateType ActionDay;

};

Note: ActionDay is a newly added filed containing the transaction date. It could be null before this field is open to the public. Trade date should be retrieved from TradingDay field.

**pRspInfo**：containing information indicating whether a query was executed successfully, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User is not authorized for this operation | User was requesting an operation that s/he is not authorized to make. |
| 57 | Cannot perform for other members | User cannot query for members that s/he does not belong to. |

**nRequestID**：requests the ID specified by the member system in its quote query request

**bIsLast**：indicates whether this response is the latest return with respect to the nRequestID.

### OnRspQryTrade Method

This callback is invoked to handle query results returned from the Exchange for trade.

**Function Signature:**

**void OnRspQryTrade(**

**CShfeFtdcTradeField \*pTrade,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pTrade**：points to the object containing trade information, with the structure below:

struct CShfeFtdcTradeField {

///Trading Date

TShfeFtdcDateType TradingDay;

///Settlement Group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Trade ID

TShfeFtdcTradeIDType TradeID;

///Buy/Sell

TShfeFtdcDirectionType Direction;

///Order ID

TShfeFtdcOrderSysIDType OrderSysID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trading Role

TShfeFtdcTradingRoleType TradingRole;

///Account ID

TShfeFtdcAccountIDType AccountID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Open position or close position

TShfeFtdcOffsetFlagType OffsetFlag;

///Hedge or Speculation

TShfeFtdcHedgeFlagType HedgeFlag;

///Price

TShfeFtdcPriceType Price;

///Quantity

TShfeFtdcVolumeType Volume;

///Matching time

TShfeFtdcTimeType TradeTime;

///Trade Type

TShfeFtdcTradeTypeType TradeType;

///Trade Price Source

TShfeFtdcPriceSourceType PriceSource;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local Order ID

TShfeFtdcOrderLocalIDType OrderLocalID;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Business Unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Transaction date

TShfeFtdcDateType ActionDay;

};

**pRspInfo:** points to the object containing the transaction status, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | No permission for the requested action | User was requesting an action that s/he is not authorized to make. For example, a regular trader was trying to perform anoperation that is authorized only to the lead trader/controller. |
| 57 | Operation not permitted for a non-member | User was trying to query for a member s/he does not belong to. |

**nRequestID**：requests the IDspecified by the member system in its trade query request

**bIsLast**: indicates whether the current response is the latest with respect to the nRequestID

### OnRspQryClient Method

This callback is invoked to handle query results from the Exchange regarding client info.

**Function Signature:**

**void OnRspQryClient(**

**CShfeFtdcRspClientField\*pClient,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameter:**

**pClient**: points to the object containing client info, with the structure below:

struct CShfeFtdcRspClientField {

///Client ID

TShfeFtdcClientIDType ClientID;

///Client name

TShfeFtdcPartyNameType ClientName;

///Identification Type

TShfeFtdcIdCardTypeType IdentifiedCardType;

///Original Identification number

TShfeFtdcIdentifiedCardNoV1Type UseLess;

///Trading Role

TShfeFtdcTradingRoleType TradingRole;

///Client type

TShfeFtdcClientTypeType ClientType;

///Active or not

TShfeFtdcBoolType IsActive;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///ID Number

TShfeFtdcIdentifiedCardNoType IdentifiedCardNo;

};

**pRspInfo**：points to the object containing the transaction status, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User not authorized | User is not authorized for this operation. |
| 57 | Operation cannot be conducted for other members | User cannot query data under other members. |

**nRequestID:** requests the IDspecified by the member system in its client info query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryPartPosition Method

This callback is invoked to handle query results returned from the Exchange regarding a member’s position.

**Function Signature:**

**void OnRspQryPartPosition(**

**CShfeFtdcRspPartPositionField \*pRspPartPosition,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameter:**

**pRspPartPosition**：points to the object containing a member’s position, with the structure below:

struct CShfeFtdcRspPartPositionField {

///Trade Date

TShfeFtdcDateType TradingDay;

///Settlement Group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

///Long or short

TShfeFtdcPosiDirectionType PosiDirection;

///Position from previous trade date

TShfeFtdcVolumeType YdPosition;

///Position from current trade date

TShfeFtdcVolumeType Position;

///Long position being frozen

TShfeFtdcVolumeType LongFrozen;

///Short position being frozen

TShfeFtdcVolumeType ShortFrozen;

///Long position from previous trade day in frozen

TShfeFtdcVolumeType YdLongFrozen;

///Short position from previous trade day in frozen

TShfeFtdcVolumeType YdShortFrozen;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Trading role

TShfeFtdcTradingRoleType TradingRole;

};

**pRspInfo**：points to the object containing the transaction status, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User not authorized | User is not authorized for the requested operation. |
| 57 | Operation cannot be conducted for other members | User cannot query a position for a member s/he does not belong to. |

**nRequestID**：requests the IDspecified by the member system in its member’s position query

**bIsLast**: indicates whether thecurrent return is the latest with respect to the nRequestID.

### OnRspQryClientPosition Method

This callback is invoked to handle query results from the Exchange about a client’s position.

**Function Signature:**

**void OnRspQryClientPosition(**

**CShfeFtdcRspClientPositionField \*pRspClientPosition,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspClientPosition**：points to the object containing a client’s position, with the structure below:

struct CShfeFtdcRspClientPositionField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

//long or short position

TShfeFtdcPosiDirectionType PosiDirection;

///Previous trade day's position

TShfeFtdcVolumeType YdPosition;

///Position of thecurrent trade day

TShfeFtdcVolumeType Position;

///Long position being frozen

TShfeFtdcVolumeType LongFrozen;

///Short position being frozen

TShfeFtdcVolumeType ShortFrozen;

///Long position from previous trade date in frozen

TShfeFtdcVolumeType YdLongFrozen;

///Short position from previous trade date in frozen

TShfeFtdcVolumeType YdShortFrozen;

///Total tradingvolume on buy-side on current trade date

TShfeFtdcVolumeType BuyTradeVolume;

///Total trading volume on sell-side on current trade date

TShfeFtdcVolumeType SellTradeVolume;

///Total monetary value of position

TShfeFtdcMoneyType PositionCost;

///Total monetary value of position from previous trade date

TShfeFtdcMoneyType YdPositionCost;

///Margin used

TShfeFtdcMoneyType UseMargin;

///Frozen Margin

TShfeFtdcMoneyType FrozenMargin;

///Frozen marginon long

TShfeFtdcMoneyType LongFrozenMargin;

///Frozen Margin on short

TShfeFtdcMoneyType ShortFrozenMargin;

///Frozen premium

TShfeFtdcMoneyType FrozenPremium;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

};

**pRspInfo**：points to the object containing the transaction status, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User not authorized | User is not authorized for this operation. |
| 57 | Operation cannot be conducted for other members | User can only query for members he belongs to. |

**nRequestID**：requests the ID specified by the member system in its instrument query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID.

### OnRspQryInstrument Method

This callback is invoked to handle query results from the Exchange regarding an instrument.

**Function Signature:**

**void OnRspQryInstrument(**

**CShfeFtdcRspInstrumentField \*pRspInstrument,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspInstrument**: points to the object containing instrument info, with the structure below:

struct CShfeFtdcRspInstrumentField {

///Settlement group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Product ID

TShfeFtdcProductIDType ProductID;

///Product group ID

TShfeFtdcProductGroupIDType ProductGroupID;

///symbol of underlying instrument

TShfeFtdcInstrumentIDType UnderlyingInstrID;

///Product class

TShfeFtdcProductClassType ProductClass;

///Type of position

TShfeFtdcPositionTypeType PositionType;

///Strike price

TShfeFtdcPriceType StrikePrice;

///Option type

TShfeFtdcOptionsTypeType OptionsType;

///Contract multiplier

TShfeFtdcVolumeMultipleType VolumeMultiple;

///Contract multiplier for underlying commodity

TShfeFtdcUnderlyingMultipleType UnderlyingMultiple;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Instrument name

TShfeFtdcInstrumentNameType InstrumentName;

///Delivery year

TShfeFtdcYearType DeliveryYear;

///Delivery month

TShfeFtdcMonthType DeliveryMonth;

///Month in advance

TShfeFtdcAdvanceMonthType AdvanceMonth;

///currently the instrument is or is notbeing traded

TShfeFtdcBoolType IsTrading;

///Creation date

TShfeFtdcDateType CreateDate;

///Activationdate

TShfeFtdcDateType OpenDate;

///Expiration date

TShfeFtdcDateType ExpireDate;

///First delivery date

TShfeFtdcDateType StartDelivDate;

///Last delivery date

TShfeFtdcDateType EndDelivDate;

///Basis price for listing

TShfeFtdcPriceType BasisPrice;

///Maximum order quantity permitted for market order

TShfeFtdcVolumeType MaxLimitOrderVolume;

///Minimum order quantity permitted for market order

TShfeFtdcVolumeType MinLimitOrderVolume;

///Tick

TShfeFtdcPriceType PriceTick;

///Position opened by individual client (vs. institutional client) during delivery month

TShfeFtdcMonthCountType AllowDelivPersonOpen;

///Currency code

TShfeFtdcCurrencyIDType CurrencyID;

};

**pRspInfo**：points to the object containing instrument information, with the structure below:

struct CShfeFtdcRspInfoField {

///Error ID

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its instrument query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryInstrumentStatus Method

This callback is invoked to handle query results from the Exchange regarding instrument status.

**Function Signature:**

**void OnRspQryInstrumentStatus(**

**CShfeFtdcInstrumentStatusField \*pInstrumentStatus,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pInstrumentStatus**：points to the object containing the instrument status:

struct CshfeFtdcInstrumentStatusField {

///Settlement group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Instrument status

TShfeFtdcInstrumentStatusType InstrumentStatus;

///Trading session (segment) sequence number

TShfeFtdcTradingSegmentSNType TradingSegmentSN;

///Time of entering current status

TShfeFtdcTimeType EnterTime;

///Reason for entering current status

TShfeFtdcInstStatusEnterReasonType EnterReason;

};

**pRspInfo**：points to the object containing the status message of this transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///

TShfeFtdcErrorIDType ErrorID;

///

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its instrument status query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryBulletin Method

This callback is invoked to handle query result from the Exchange regarding advisories published by the Exchange.

**Function Signature:**

**void OnRspQryBulletin(**

**CShfeFtdcBulletinField \*pBulletin,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pBulletin**：points to the object containing returned advisory, with the structure below:

struct CShfeFtdcBulletinField {

///Trade date

TShfeFtdcDateType TradingDay;

///Bulletin number

TShfeFtdcBulletinIDType BulletinID;

///Sequence number

TShfeFtdcSequenceNoType SequenceNo;

///Bulletin type

TShfeFtdcNewsTypeType NewsType;

///level of Urgency

TShfeFtdcNewsUrgencyType NewsUrgency;

///Transmission time

TShfeFtdcTimeType SendTime;

///Message abstract

TShfeFtdcAbstractType Abstract;

///Source of message

TShfeFtdcComeFromType ComeFrom;

///Message body

TShfeFtdcContentType Content;

///WEB address

TShfeFtdcURLLinkType URLLink;

///Market code

TShfeFtdcMarketIDType MarketID;

};

**pRspInfo**：points to the object containting transaction status returned from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///Error ID

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the IDspecified by the member system in its advisory query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryMarketData Method

This callback is invoked to handle query results from the Exchange regarding comprehensive market data (vs. market by price level).

**Function Signature:**

**void OnRspQryMarketData(**

**CShfeFtdcMarketDataField \*pMarketData,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pMarketData**：points to the object containing returned market data, with the structure below:

struct CShfeFtdcMarketDataField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///The latest price

TShfeFtdcPriceType LastPrice;

///Settlement price of previous trade date

TShfeFtdcPriceType PreSettlementPrice;

///Close price of previous trade date

TShfeFtdcPriceType PreClosePrice;

///Open interest from previous trade date

TShfeFtdcLargeVolumeType PreOpenInterest;

///Current day open price

TShfeFtdcPriceType OpenPrice;

///Highest price of current day

TShfeFtdcPriceType HighestPrice;

///Lowest price of current day

TShfeFtdcPriceType LowestPrice;

///Quantity

TShfeFtdcVolumeType Volume;

///Turnover

TShfeFtdcMoneyType Turnover;

///Open interest

TShfeFtdcLargeVolumeType OpenInterest;

///Closing price of current day

TShfeFtdcPriceType ClosePrice;

///Settlement price of current day

TShfeFtdcPriceType SettlementPrice;

///Upper limit

TShfeFtdcPriceType UpperLimitPrice;

///Lower limit

TShfeFtdcPriceType LowerLimitPrice;

///Delta value from previous trade date

TShfeFtdcRatioType PreDelta;

///Delta value from current trade date

TShfeFtdcRatioType CurrDelta;

///Last modification time

TShfeFtdcTimeType UpdateTime;

///The last modified millisecond

TShfeFtdcMillisecType UpdateMillisec;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Date of the transaction

TShfeFtdcDateType ActionDay;

};

**pRspInfo**：points to the object containing the transaction status from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its market data query

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID.

### OnRspQryMBLMarketData Method

This callback is invoked to handle query results from the Exchange regarding market data at a certain price level.

**Function Signature:**

**void OnRspQryMBLMarketData(**

**CShfeFtdcMBLMarketDataField \*pMBLMarketData,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pMBLMarketData**：points to the object containing the price level, with the structure below:

struct CShfeFtdcMBLMarketDataField {

///Contract symbol

TShfeFtdcInstrumentIDType InstrumentID;

///Buy or sell

TShfeFtdcDirectionType Direction;

///Price

TShfeFtdcPriceType Price;

///volume

TShfeFtdcVolumeType Volume;

};

**pRspInfo**：points to the object containing the transaction status.

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its query request

**bIsLast**: indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryHedgeVolume Method

This callback is invoked to handle result from the Exchange regarding query for hedge quota.

**Function Signature:**

**void OnRspQryHedgeVolume(**

**CShfeFtdcHedgeVolumeField \*pHedgeVolume,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pHedgeVolume**：points to the object containing the returned hedge quota. The structure is shown below:

struct CShfeFtdcHedgeVolumeField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group's ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Initial approved hedge quota for long(unit: lot)

TShfeFtdcVolumeType LongVolumeOriginal;

///Initial approved hedge quota for short(unit: lot)

TShfeFtdcVolumeType ShortVolumeOriginal;

///Hedge quotaon long (unit: lot)

TShfeFtdcVolumeType LongVolume;

///Hedge quotaon short (unit: lot)

TShfeFtdcVolumeType ShortVolume;

};

**pRspInfo**：points to the object containing the transaction status returned from the Exchange, with the structure below:

|  |  |  |
| --- | --- | --- |
| struct CShfeFtdcRspInfoField {  /// Error code  TShfeFtdcErrorIDType ErrorID;  /// Error message  TShfeFtdcErrorMsgType ErrorMsg;  };  The possible errors | | |
| **Error code** | **Error message** | **Possible reasons** |
| 80 | User not authorized | User attempts to perform operation authorized for lead trader (controller) only |
| 57 | User cannot perform for non-member | User attempts to query for the member he does not belong to |

**nRequestID**：requests the ID specified in the member system in its query request

**bIsLast**: indicates whether this current return is the latest with respect to the nRequestID

### OnRtnTrade Method

This callback is invoked to handle trade notifications from the Exchange.

**Function Signature:**

**void OnRtnTrade(CShfeFtdcTradeField \*pTrade)；**

**Parameter:**

**pTrade**：points to the object containing returned execution reports about trading, with the structure below (some fields are not used and therefore contain no value):

struct CShfeFtdcTradeField {

///trade date

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Trade ID

TShfeFtdcTradeIDType TradeID;

///Buy or sell

TShfeFtdcDirectionType Direction;

///Order number

TShfeFtdcOrderSysIDType OrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trading role,**not used**

TShfeFtdcTradingRoleType TradingRole;

///Margin account,**not used**

TShfeFtdcAccountIDType AccountID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Open or close position

TShfeFtdcOffsetFlagType OffsetFlag;

///Speculation orhedge

TShfeFtdcHedgeFlagType HedgeFlag;

///Price

TShfeFtdcPriceType Price;

///Quantity

TShfeFtdcVolumeType Volume;

///Order matching time

TShfeFtdcTimeType TradeTime;

///Trade type,**not used**

TShfeFtdcTradeTypeType TradeType;

///Source of trade price (last trade price or best bid or best offer),**not used**

TShfeFtdcPriceSourceType PriceSource;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Business unit,**not used**

TShfeFtdcBusinessUnitType BusinessUnit;

///Business Local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Transaction date

TShfeFtdcDateType ActionDay;

};

Notes: ActionDay is a newly added field for the transaction date. Until it is open to the public, this field contains no value. The trade date should be parsed from the TradingDay field.

### OnRtnOrder Method

This callback is invoked to handle execution reports from the Exchange in terms of order related transactions, such as new order entries, order updates, order cancellations, or partial fills, which cause the order status to change.

**Function Signature:**

**void OnRtnOrder(CShfeFtdcOrderField \*pOrder);**

**Parameter:**

**pOrder**：points to the object containing the execution report of the order, with the structure below (some fields are empty and thus contain no value):

struct CShfeFtdcOrderField {

///Trade date,**not used**

TShfeFtdcDateType TradingDay;

///Settlement group's code,**not used**

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number,**not used**

TShfeFtdcSettlementIDType SettlementID;

///Order number

TShfeFtdcOrderSysIDType OrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Order type

TShfeFtdcOrderPriceTypeType OrderPriceType;

///Buy or sell

TShfeFtdcDirectionType Direction;

///Open or close position

TShfeFtdcCombOffsetFlagType CombOffsetFlag;

///Speculation or hedge

TShfeFtdcCombHedgeFlagType CombHedgeFlag;

///Limit Price

TShfeFtdcPriceType LimitPrice;

///Original Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Order validity

TShfeFtdcTimeConditionType TimeCondition;

///GTD DATE

TShfeFtdcDateType GTDDate;

///Matched quantity condition

TShfeFtdcVolumeConditionType VolumeCondition;

///Minimum matchedquantity

TShfeFtdcVolumeType MinVolume;

///Trigger conditions

TShfeFtdcContingentConditionType ContingentCondition;

///Trigger price

TShfeFtdcPriceType StopPrice;

///Reasonof enforced position closing (offset)

TShfeFtdcForceCloseReasonType ForceCloseReason;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag of auto-suspension

TShfeFtdcBoolType IsAutoSuspend;

///Origin of order,**not used**

TShfeFtdcOrderSourceType OrderSource;

///Order status

TShfeFtdcOrderStatusType OrderStatus;

///Order type,**not used**

TShfeFtdcOrderTypeType OrderType;

///Traded volume on current trade date,**not used**

TShfeFtdcVolumeType VolumeTraded;

///Remaining quantity

TShfeFtdcVolumeType VolumeTotal;

///Order entry date

TShfeFtdcDateType InsertDate;

///Order entry time,**not used**

TShfeFtdcTimeType InsertTime;

///Order activation time,**not used**

TShfeFtdcTimeType ActiveTime;

///Order suspension time,**not used**

TShfeFtdcTimeType SuspendTime;

///Last modification time

TShfeFtdcTimeType UpdateTime;

///Time of cancellation,**not used**

TShfeFtdcTimeType CancelTime;

///Trader ID who made last update

TShfeFtdcUserIDType ActiveUserID;

///Priority,**not used**

TShfeFtdcPriorityType Priority;

///Sequence number (ordered by time),**not used**

TShfeFtdcTimeSortIDType TimeSortID;

///Clearing member number

TShfeFtdcParticipantIDType ClearingPartID;

///Business unit,**not used**

TShfeFtdcBusinessUnitType BusinessUnit;

///Business Local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Transaction date

TShfeFtdcDateType ActionDay;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

### OnRtnQuote Method

This method is not currently supported by the business.

This callback is invoked to handle execution reports for quote-related transactions, such as new quote entries, quote updates, and cancellations which cause the order status to change, with the structure below:

**Function Signature:**

**void OnRtnQuote(CShfeFtdcQuoteField \*pQuote);**

**Parameter:**

**pQuote**：points to the object containing returned execution reports on quote transactions

struct CShfeFtdcQuoteField {

///Trade day

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Quote number

TShfeFtdcQuoteSysIDType QuoteSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Quantity

TShfeFtdcVolumeType Volume;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Local quote number

TShfeFtdcOrderLocalIDType QuoteLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Opening or closing position on buy-side

TShfeFtdcCombOffsetFlagType BidCombOffsetFlag;

///Hedge or Speculationon buy-side

TShfeFtdcCombHedgeFlagType BidCombHedgeFlag;

///Bid price

TShfeFtdcPriceType BidPrice;

///Opening or closing position on sell-side

TShfeFtdcCombOffsetFlagType AskCombOffsetFlag;

///Hedge or speculation on sell-side

TShfeFtdcCombHedgeFlagType AskCombHedgeFlag;

///Offer price

TShfeFtdcPriceType AskPrice;

///Quote entry Time

TShfeFtdcTimeType InsertTime;

///Quote cancellation time

TShfeFtdcTimeType CancelTime;

///Trade time

TShfeFtdcTimeType TradeTime;

///Buy-side order number

TShfeFtdcOrderSysIDType BidOrderSysID;

///Sell-side order number

TShfeFtdcOrderSysIDType AskOrderSysID;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Business Local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Action Day

TShfeFtdcDateType ActionDay;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TShfeFtdcMacAddressType MacAddress;

};

Note: ActionDay is a newly added field containing the transaction date. Until it is open to the public, it does not contain any value. Trade date should be parsed from TradingDay.

### OnRtnExecOrder Method

This method is not currently supported by the business.

This callback is invoked to retrieve the execution report of exercise (on option).

**Function Signature:**

**void OnRtnExecOrder(CShfeFtdcExecOrderField \*pExecOrder)；**

**Parameter:**

**pExecOrder**: points to the object containing the returned execution report of exercise, with the structure below:

struct CShfeFtdcExecOrderField {

///Trade day

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Contract number

TShfeFtdcInstrumentIDType InstrumentID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local exercise ID

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///quantity

TShfeFtdcVolumeType Volume;

///Opening or closing (offsetting) position

TShfeFtdcOffsetFlagType OffsetFlag;

///Hedge or speculation

TShfeFtdcHedgeFlagType HedgeFlag;

///Long or short position this exercise is applied to

TShfeFtdcPosiDirectionType PosiDirection;

///Reserving position or not after exercise

TShfeFtdcExecOrderPositionFlagType ReservePositionFlag;

///Offsetting or not the generated position resulted from exercise

TShfeFtdcExecOrderCloseFlagType CloseFlag;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Exercise id

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Exercise entry date

TShfeFtdcDateType InsertDate;

///Exercise entry time

TShfeFtdcTimeType InsertTime;

///Time of exercise cancellation

TShfeFtdcTimeType CancelTime;

///Execution result

TShfeFtdcExecResultType ExecResult;

///Settlement clearing number

TShfeFtdcParticipantIDType ClearingPartID;

///Business Local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Action Day

TShfeFtdcDateType ActionDay;

///IP Address

TShfeFtdcIPAddressType IPAddress;

///Mac Address

TShfeFtdcMacAddressType MacAddress;

};

Note: ActionDay is a newly added field, containing the transaction date. Before it is open to the public, it does not contain any value. The trade date should be parsed from TradingDay.

### OnRtnInstrumentStatus Method

This callback is invoked to handle changes in instrument status distributed from the Exchange. When the instrument status is changed, the Exchange will distribute a notification.

**Function Signature:**

**void OnRtnInstrumentStatus(**

**CShfeFtdcInstrumentStatusField \*pInstrumentStatus);**

**Parameter:**

**pInstrumentStatus**：points to the object containing a new instrument status, with the structure below:

struct CShfeFtdcInstrumentStatusField {

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Trading status of contract

TShfeFtdcInstrumentStatusType InstrumentStatus;

///Trading segment sequence number

TShfeFtdcTradingSegmentSNType TradingSegmentSN;

///Time of entering this status

TShfeFtdcTimeType EnterTime;

///Reasons for entering this status

TShfeFtdcInstStatusEnterReasonType EnterReason;

};

### OnRtnInsInstrument Method

This callback is invoked to handle newly added instruments. After logging into the Exchange, the member system will receive notification of newly added instruments from the public data stream.

**Function Signature:**

**void OnRtnInsInstrument(CShfeFtdcInstrumentField \*pInstrument)；**

**Parameter:**

**pInstrument**：points to the object containing new instrument information, with the structure below:

struct CShfeFtdcInstrumentField {

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Product id

TShfeFtdcProductIDType ProductID;

///Product group id

TShfeFtdcProductGroupIDType ProductGroupID;

///Code of underlying commodity

TShfeFtdcInstrumentIDType UnderlyingInstrID;

///Product class

TShfeFtdcProductClassType ProductClass;

///Type of position

TShfeFtdcPositionTypeType PositionType;

///Strike price

TShfeFtdcPriceType StrikePrice;

///Option type

TShfeFtdcOptionsTypeType OptionsType;

///Quantity multiplier

TShfeFtdcVolumeMultipleType VolumeMultiple;

///Basis multiplier for the underlying commodity

TShfeFtdcUnderlyingMultipleType UnderlyingMultiple;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Contract name

TShfeFtdcInstrumentNameType InstrumentName;

///Delivery year

TShfeFtdcYearType DeliveryYear;

///Delivery month

TShfeFtdcMonthType DeliveryMonth;

///The month immediately before the delivery month

TShfeFtdcAdvanceMonthType AdvanceMonth;

///Flag indicating whether or not the contract is currently being traded

TShfeFtdcBoolType IsTrading;

};

### OnRtnDelInstrument Method

This callback is invoked to handle the notification of deleted instruments from the Exchange. After logging in, the member system will receive information about the deleted instrument from the Exchange via the public data stream.

**Function Signature:**

**void OnRtnDelInstrument(CShfeFtdcInstrumentField \*pInstrument)；**

**Parameter:**

**pInstrument**: points to the instrument object, with the structure below:

struct CShfeFtdcInstrumentField {

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Product code

TShfeFtdcProductIDType ProductID;

///Product group code

TShfeFtdcProductGroupIDType ProductGroupID;

///Code of underlying commodity

TShfeFtdcInstrumentIDType UnderlyingInstrID;

///Product class

TShfeFtdcProductClassType ProductClass;

///Position type

TShfeFtdcPositionTypeType PositionType;

///Strike price

TShfeFtdcPriceType StrikePrice;

///Option type

TShfeFtdcOptionsTypeType OptionsType;

///Quantity multiplier

TShfeFtdcVolumeMultipleType VolumeMultiple;

///Basis multiplier for underlying commodity

TShfeFtdcUnderlyingMultipleType UnderlyingMultiple;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Contract name

TShfeFtdcInstrumentNameType InstrumentName;

///Delivery year

TShfeFtdcYearType DeliveryYear;

///Delivery month

TShfeFtdcMonthType DeliveryMonth;

///The month immediately before the delivery month

TShfeFtdcAdvanceMonthType AdvanceMonth;

///current contract status is trading or not

TShfeFtdcBoolType IsTrading;

};

### OnRtnInsCombinationLeg Method

**This method is not supported yet.**

This callback is invoked to handle the leg contract of new spread contracts added to the Exchange. The Exchange will distribute the leg contract info via the public data stream when a new spread contract is added.

**Function Signature：**

**void OnRtnInsCombinationLeg(**

**CShfeFtdcCombinationLegField \*pCombinationLeg);**

**Parameter：**

**pCombinationLeg**：points to the object containing the leg contract info, with the structure below:

struct CShfeFtdcCombinationLegField {

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Spread contract

TShfeFtdcInstrumentIDType CombInstrumentID;

///Leg number

TShfeFtdcLegIDType LegID;

///Leg contract symbol

TShfeFtdcInstrumentIDType LegInstrumentID;

///Buy or sell direction

TShfeFtdcDirectionType Direction;

///Leg multiplier

TShfeFtdcLegMultipleType LegMultiple;

///Implied level

TShfeFtdcImplyLevelType ImplyLevel;

};

### OnRtnDelCombinationLeg Method

**This method is not supported yet.**

This callback is invoked to handle legs from deleted spreads.The Exchange will distribute the leg information from the removed spread contract via the public data stream.

**Function Signature：**

**void OnRtnDelCombinationLeg(**

**CShfeFtdcCombinationLegField \*pCombinationLeg);**

**Parameter：**

**pCombinationLeg**：points to the object containing the removed leg information, with the structure below:

struct CShfeFtdcCombinationLegField {

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Portfolio contract code

TShfeFtdcInstrumentIDType CombInstrumentID;

///Leg number

TShfeFtdcLegIDType LegID;

///Single leg contract code

TShfeFtdcInstrumentIDType LegInstrumentID;

///Buy or sell

TShfeFtdcDirectionType Direction;

///Single leg multiplier

TShfeFtdcLegMultipleType LegMultiple;

///Deduction of layers

TShfeFtdcImplyLevelType ImplyLevel;

};

### OnRtnBulletin Method

This callback is invoked to handle advisories received from the Exchange.

**Function Signature:**

**void OnRtnBulletin(CShfeFtdcBulletinField \*pBulletin);**

**Parameter:**

**pBulletin**: points to the object containing advisories, with the structure below:

struct CShfeFtdcBulletinField {

///Trade date

TShfeFtdcDateType TradingDay;

///Bulletin number

TShfeFtdcBulletinIDType BulletinID;

///Sequence number

TShfeFtdcSequenceNoType SequenceNo;

///Bulletin type

TShfeFtdcNewsTypeType NewsType;

///Urgency

TShfeFtdcNewsUrgencyType NewsUrgency;

///Transmission time

TShfeFtdcTimeType SendTime;

///Message abstract

TShfeFtdcAbstractType Abstract;

///Source of message

TShfeFtdcComeFromType ComeFrom;

///Message body

TShfeFtdcContentType Content;

///WEB address

TShfeFtdcURLLinkType URLLink;

///Market code

TShfeFtdcMarketIDType MarketID;

};

### OnRtnAliasDefine Method

This callback is invoked to handle responses from the Exchange regarding alias definition requests.

**Function Signature:**

**void OnRtnAliasDefine(CShfeFtdcAliasDefineField \*pAliasDefine)；**

**Parameter:**

**pAliasDefine:** points to the alias definition request, with the structure below:

struct CShfeFtdcAliasDefineField {

///Starting position

TShfeFtdcStartPosType StartPos;

///Alias

TShfeFtdcAliasType Alias;

///Original text

TShfeFtdcOriginalTextType OriginalText;

};

### OnRtnFlowMessageCancelMethod

This callback is invoked to handle rolled back transactions from the Exchange after the member system logs back in and subscribes to a data steam (public or private). The Exchange may cancel certain execution reports due to data inconsistency after switching over from the primary system to the backup system.

**Function Signature:**

**void OnRtnFlowMessageCancel(**

**CShfeFtdcFlowMessageCancelField \*pFlowMessageCancel)；**

**Parameter:**

**pFlowMessageCancel**: points to the object containing rolled back transactions, with the structure below:

struct CShfeFtdcFlowMessageCancelField

{

///Data stream series number

TShfeFtdcSequenceSeriesType SequenceSeries;

///Trade date

TShfeFtdcDateType TradingDay;

///Datacenter id

TShfeFtdcDataCenterIDType DataCenterID;

/// Starting sequence number of rolled-back execution reports

TShfeFtdcSequenceNoType StartSequenceNo;

///Ending sequence number of rolled-back execution reports

TShfeFtdcSequenceNoType EndSequenceNo;

};

SequenceSeries：data stream series number（Private stream or public stream）

Execution report being rolled-back: （StartSequenceNo,EndSequenceNo］

### OnErrRtnOrderInsertMethod

This callback is invoked to handle errors returned from the Exchange with respect to new order entry requests.

**Function Signature:**

**void OnErrRtnOrderInsert(**

**CShfeFtdcInputOrderField \*pInputOrder,**

**CShfeFtdcRspInfoField \*pRspInfo)；**

**Parameters:**

**pInputOrder**：points to the new order entry request, with the structure below:

struct CShfeFtdcInputOrderField {

///order number, with value returned from the Exchange

TShfeFtdcOrderSysIDType OrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader id

TShfeFtdcUserIDType UserID;

///Contract symbol

TShfeFtdcInstrumentIDType InstrumentID;

///Order type (Limit order, Stop order, Market order etc.)

TShfeFtdcOrderPriceTypeType OrderPriceType;

///Buy or sell

TShfeFtdcDirectionType Direction;

///Open or Close position

TShfeFtdcCombOffsetFlagType CombOffsetFlag;

///Speculation or Hedge

TShfeFtdcCombHedgeFlagType CombHedgeFlag;

///Limit Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Order validity(GTD, GTC etc.)

TShfeFtdcTimeConditionType TimeCondition;

///GTD DATE

TShfeFtdcDateType GTDDate;

///Matched quantity type (random, fully matched or minimum matched etc.)

TShfeFtdcVolumeConditionType VolumeCondition;

///Min volume

TShfeFtdcVolumeType MinVolume;

///Trigger condition

TShfeFtdcContingentConditionType ContingentCondition;

///Trigger price

TShfeFtdcPriceType StopPrice;

///Reasons ofenforced closing (offset)

TShfeFtdcForceCloseReasonType ForceCloseReason;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Suspending order or not when unexpected disconnection occurs

TShfeFtdcBoolType IsAutoSuspend;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange for this transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnOrderActionMethod

This callback is invoked to handle errors returned from the Exchange with respect to action requests on existing orders.

**Function Signature:**

**void OnErrRtnOrderAction (**

**CShfeFtdcOrderActionField \*pOrderAction,**

**CShfeFtdcRspInfoField \*pRspInfo)；**

**Parameters:**

**pOrderAction**: points to the order action request, with the structure below:

struct CShfeFtdcOrderActionField {

///order number, returned from the Exchange

TShfeFtdcOrderSysIDType OrderSysID;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag of order operation type

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price

TShfeFtdcPriceType LimitPrice;

///Change in quantity

TShfeFtdcVolumeType VolumeChange;

///Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the engine regarding this transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnQuoteInsertMethod

**This method is not currentlysupported by the business.**

This callback is invoked to handle errors returned from the Exchange regarding new quote entry requests.

**Function Signature：**

**void OnErrRtnQuoteInsert (**

**CShfeFtdcInputQuoteField \*pInputQuote,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pInputQuote**：points to the quote entry request, with the structure below:

struct CShfeFtdcInputQuoteField {

///Quote number, returned from the Exchange

TShfeFtdcQuoteSysIDType QuoteSysID;

///Member Id

TShfeFtdcParticipantIDType ParticipantID;

///Client Id

TShfeFtdcClientIDType ClientID;

///Trader Id

TShfeFtdcUserIDType UserID;

///Quantity

TShfeFtdcVolumeType Volume;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Local quote number

TShfeFtdcQuoteLocalIDType QuoteLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Flag indicating whether to open or close position on buy-side

TShfeFtdcCombOffsetFlagType BidCombOffsetFlag;

///Flag indicating whether to hedge or speculate on buy-side

TShfeFtdcCombHedgeFlagType BidCombHedgeFlag;

///Bid price

TShfeFtdcPriceType BidPrice;

///Flag indicating whether to open or close position on sell-side

TShfeFtdcCombOffsetFlagType AskCombOffsetFlag;

///Flag indicating hedge or speculation on sell-side

TShfeFtdcCombHedgeFlagType AskCombHedgeFlag;

///Ask price

TShfeFtdcPriceType AskPrice;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the Exchange with respect to this transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnQuoteActionMethod

**This method is not yet supported by the business.**

This callback is invoked to handle errors returned from the Exchange with respect to action requests on existing quotes.

**Function Signature：**

**void OnErrRtnQuoteAction (**

**CShfeFtdcQuoteActionField \*pQuoteAction,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pQuoteAction**：points to the quote action request, with the structure below:

struct CShfeFtdcQuoteActionField {

///Quote number, returned from the Exchange

TShfeFtdcQuoteSysIDType QuoteSysID;

///Local quote number

TShfeFtdcOrderLocalIDType QuoteLocalID;

///Flag of order operation type

TShfeFtdcActionFlagType ActionFlag;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the Exchange with respect to this transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnExecOrderInsertMethod

**This method is not yet supported by the business.**

This callback is invoked to handle errors returned from the Exchange regarding new exercise entry requests.

**Function Signature：**

**void OnErrRtnExecOrderInsert (**

**CShfeFtdcInputExecOrderField \*pInputExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo)；**

**Function：**

**pInputExecOrder**：points to the exercise entry request, with the structure below:

struct CShfeFtdcInputExecOrderField {

///Contract number

TShfeFtdcInstrumentIDType InstrumentID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local exercise number

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Quantity

TShfeFtdcVolumeType Volume;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Flag indicating open or close(offset) position

TShfeFtdcOffsetFlagType OffsetFlag;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange with respect to the transaction:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnExecOrderActionMethod

**This method is not yet supported by the business.**

This callback is invoked to handle errors received from the Exchange for actions on exercises.

**Function Signature：**

**void OnErrRtnExecOrderAction (**

**CShfeFtdcExecOrderActionField \*pExecOrderAction,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pInputExecAction**：points to the exercise action request, with the structure below：

struct CShfeFtdcExecOrderActionField {

///Exercise ID

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Local exercise ID

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Flag indicating action type

TShfeFtdcActionFlagType ActionFlag;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local operation number

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnRspCombOrderInsertMethod

**This method is not yet supported by the business.**

This callback is invoked to handle responses from the Exchange regarding a new combo order entry.

**Function Signature：**

**void OnRspCombOrderInsert (**

**CShfeFtdcInputCombOrderField \*pInputCombOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pInputCombOrder：**points to the combo order entry object, with the structure below:

struct CShfeFtdcInputCombOrderField {

///combo order No.

TShfeFtdcOrderSysIDType CombOrderSysID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Limit Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Local order number

TShfeFtdcOrderLocalIDType CombOrderLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Contract code of leg 1

TShfeFtdcInstrumentIDType InstrumentID1;

///Buy or sell direction of leg 1

TShfeFtdcDirectionType Direction1;

///Leg 1 multiplier

TShfeFtdcLegMultipleType LegMultiple1;

///Flag of opening or closing positionon leg 1

TShfeFtdcOffsetFlagType OffsetFlag1;

///Flag of hedge or speculation on leg 1

TShfeFtdcHedgeFlagType HedgeFlag1;

///Contract code of leg 2

TShfeFtdcInstrumentIDType InstrumentID2;

///Buy or sell direction of leg 2

TShfeFtdcDirectionType Direction2;

///Leg 2 multiplier

TShfeFtdcLegMultipleType LegMultiple2;

///Flag of closing or openingposition on leg 2

TShfeFtdcOffsetFlagType OffsetFlag2;

///Flag of hedge or speculation on leg 2

TShfeFtdcHedgeFlagType HedgeFlag2;

///Contract code of leg 3

TShfeFtdcInstrumentIDType InstrumentID3;

///Buy-sell direction on leg 3

TShfeFtdcDirectionType Direction3;

///Leg 3 multiplier

TShfeFtdcLegMultipleType LegMultiple3;

///Flag of opening or closing position on leg 3

TShfeFtdcOffsetFlagType OffsetFlag3;

///Flag of hedge or speculation on leg 3

TShfeFtdcHedgeFlagType HedgeFlag3;

///Contract code of leg 4

TShfeFtdcInstrumentIDType InstrumentID4;

///Buy or sell direction on leg 4

TShfeFtdcDirectionType Direction4;

///Leg 4 multiplier

TShfeFtdcLegMultipleType LegMultiple4;

///Flag of open or close position on leg 4

TShfeFtdcOffsetFlagType OffsetFlag4;

///Flag of hedge or speculation on leg 4

TShfeFtdcHedgeFlagType HedgeFlag4;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange regarding the entry request, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its entry request

**bIsLast**：indicates whether or not this return is the latest with respect to the nRequestID.

### OnRspQryCombOrder Method

**This method isnot yet supported by the business.**

This callback is invoked to handle responses received from the Exchange regarding combo order query.

**Function Signature：**

**void OnRspCombOrderInsert (**

**CShfeFtdcCombOrderField \*pCombOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameter：**

**pCombOrder**：points to the combo order query request, with the structure below:

struct CShfeFtdcCombOrderField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///Combo order number

TShfeFtdcOrderSysIDType CombOrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Local order number

TShfeFtdcOrderLocalIDType CombOrderLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Contract code of leg 1

TShfeFtdcInstrumentIDType InstrumentID1;

///Buy-sell direction of leg 1

TShfeFtdcDirectionType Direction1;

///Leg 1 multiplier

TShfeFtdcLegMultipleType LegMultiple1;

///Flag of position opening orclosingon leg 1

TShfeFtdcOffsetFlagType OffsetFlag1;

///Flag of speculation or hedge on leg 1

TShfeFtdcHedgeFlagType HedgeFlag1;

///Contract code of leg 2

TShfeFtdcInstrumentIDType InstrumentID2;

///Buy-sell direction of leg 2

TShfeFtdcDirectionType Direction2;

///Leg 2 multiplier

TShfeFtdcLegMultipleType LegMultiple2;

///Flag of position opening orclosingon leg 2

TShfeFtdcOffsetFlagType OffsetFlag2;

///Flag of speculation or hedge on leg 2

TShfeFtdcHedgeFlagType HedgeFlag2;

///Contract code of leg 3

TShfeFtdcInstrumentIDType InstrumentID3;

///Buy-sell direction of leg 3

TShfeFtdcDirectionType Direction3;

///Leg 3 multiplier

TShfeFtdcLegMultipleType LegMultiple3;

///Flag of position opening orclosingon leg 3

TShfeFtdcOffsetFlagType OffsetFlag3;

///Flag of speculation or hedge on leg 3

TShfeFtdcHedgeFlagType HedgeFlag3;

///Contract code of leg 4

TShfeFtdcInstrumentIDType InstrumentID4;

///Buy-sell direction of leg 4

TShfeFtdcDirectionType Direction4;

///Leg 4 multiplier

TShfeFtdcLegMultipleType LegMultiple4;

///Flag of position opening orclosingon leg 4

TShfeFtdcOffsetFlagType OffsetFlag4;

///Flag of speculation or hedge on leg 4

TShfeFtdcHedgeFlagType HedgeFlag4;

///Origin of order

TShfeFtdcOrderSourceType OrderSource;

///Traded volume on current trade date

TShfeFtdcVolumeType VolumeTraded;

///Remaining quantity

TShfeFtdcVolumeType VolumeTotal;

///Date of order entry

TShfeFtdcDateType InsertDate;

///Time of order entry

TShfeFtdcTimeType InsertTime;

///Settlement clearing ID

TShfeFtdcParticipantIDType ClearingPartID;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///Operation date

TShfeFtdcDateType ActionDay;

};

Note: ActionDay is a new field containing the date when the operation is executed. The trade date is retrievable from TradingDay.ActionDay is left empty before it is supported.

**pRspInfo**：points to the response received from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its query request

**bIsLast**：indicates whether this return is the latest with respect to the nRequestID.

### OnRtnCombOrderMethod

**This method is not yet supported by the business.**

This callback handles responsesfrom the Exchange regarding combo order entry requests.

**Function Signature：**

**void OnRtnCombOrder (CShfeFtdcCombOrderField \*pCombOrder);**

**Parameter：**

**pCombOrder：**points to the combo order entry, with the structure below：

struct CShfeFtdcCombOrderField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

///combo order number

TShfeFtdcOrderSysIDType CombOrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Local order number

TShfeFtdcOrderLocalIDType CombOrderLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Contract code of leg 1

TShfeFtdcInstrumentIDType InstrumentID1;

///Buy-sell direction of leg 1

TShfeFtdcDirectionType Direction1;

///Leg 1 multiplier

TShfeFtdcLegMultipleType LegMultiple1;

///Flag of position opening orclosingon leg 1

TShfeFtdcOffsetFlagType OffsetFlag1;

///Flag of speculation or hedge on leg 1

TShfeFtdcHedgeFlagType HedgeFlag1;

///Contract code of leg 2

TShfeFtdcInstrumentIDType InstrumentID2;

///Buy-sell direction of leg 2

TShfeFtdcDirectionType Direction2;

///Leg 2 multiplier

TShfeFtdcLegMultipleType LegMultiple2;

///Flag of position opening orclosingon leg 2

TShfeFtdcOffsetFlagType OffsetFlag2;

///Flag of speculation or hedge on leg 2

TShfeFtdcHedgeFlagType HedgeFlag2;

///Contract code of leg 3

TShfeFtdcInstrumentIDType InstrumentID3;

///Buy-sell direction of leg 3

TShfeFtdcDirectionType Direction3;

///Leg 3 multiplier

TShfeFtdcLegMultipleType LegMultiple3;

///Flag of position opening orclosingon leg 3

TShfeFtdcOffsetFlagType OffsetFlag3;

///Flag of speculation or hedge on leg 3

TShfeFtdcHedgeFlagType HedgeFlag3;

///Contract code of leg 4

TShfeFtdcInstrumentIDType InstrumentID4;

///Buy-sell direction of leg 4

TShfeFtdcDirectionType Direction4;

///Leg 4 multiplier

TShfeFtdcLegMultipleType LegMultiple4;

///Flag of position opening orclosingon leg 4

TShfeFtdcOffsetFlagType OffsetFlag4;

///Flag of speculation or hedge on leg 4

TShfeFtdcHedgeFlagType HedgeFlag4;

///origin of order

TShfeFtdcOrderSourceType OrderSource;

///Traded volume

TShfeFtdcVolumeType VolumeTraded;

///Remaining quantity

TShfeFtdcVolumeType VolumeTotal;

///Combo order entry date

TShfeFtdcDateType InsertDate;

/// Combo order entry Time

TShfeFtdcTimeType InsertTime;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///operation date

TShfeFtdcDateType ActionDay;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

### OnErrRtnCombOrderInsert Method

**This method is not yet supported by the business.**

This callback is invoked to handle errors received from the Exchange regarding combo order (spread) entry requests.

**Function Signature：**

**void OnErrRtnCombOrderInsert (**

**CShfeFtdcInputCombOrderField \*pInputCombOrder,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pInputCombOrder：**points to the combo order request, with the structure below:

struct CshfeFtdcInputCombOrderField {

///combo order number

TShfeFtdcOrderSysIDType CombOrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Local combo order number

TShfeFtdcOrderLocalIDType CombOrderLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Contract code of leg 1

TShfeFtdcInstrumentIDType InstrumentID1;

///Buy-sell direction of leg 1

TShfeFtdcDirectionType Direction1;

///leg 1 multiplier

TShfeFtdcLegMultipleType LegMultiple1;

///Flag of position opening orclosingon leg 1

TShfeFtdcOffsetFlagType OffsetFlag1;

///Flag of speculation or hedge on leg 1

TShfeFtdcHedgeFlagType HedgeFlag1;

///Contract code of leg 2

TShfeFtdcInstrumentIDType InstrumentID2;

///Buy-sell direction of leg 2

TShfeFtdcDirectionType Direction2;

///Leg 2 multiplier

TShfeFtdcLegMultipleType LegMultiple2;

///Flag of position opening orclosingon leg 2

TShfeFtdcOffsetFlagType OffsetFlag2;

///Flag of speculation and hedge on leg 2

TShfeFtdcHedgeFlagType HedgeFlag2;

///Contract code on leg 3

TShfeFtdcInstrumentIDType InstrumentID3;

///Buy-sell direction on leg 3

TShfeFtdcDirectionType Direction3;

///Leg 3 multiplier

TShfeFtdcLegMultipleType LegMultiple3;

///Flag of position opening orclosingon leg 3

TShfeFtdcOffsetFlagType OffsetFlag3;

///Flag of speculation or hedge on leg 3

TShfeFtdcHedgeFlagType HedgeFlag3;

///Contract code of leg 4

TShfeFtdcInstrumentIDType InstrumentID4;

///Buy-sell direction of leg 4

TShfeFtdcDirectionType Direction4;

///Leg 4 multiplier

TShfeFtdcLegMultipleType LegMultiple4;

///Flag of position opening orclosingon leg 4

TShfeFtdcOffsetFlagType OffsetFlag4;

///Flag of speculation or hedge on leg 4

TShfeFtdcHedgeFlagType HedgeFlag4;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the Exchange , with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnRspQryExecOrder Method

**This method is not supported by the business until option trading is open to the public.**

This callback is invoked to handle responses from the Exchange about exercise query requests.

**Function Signature:**

**void OnRspQryExecOrder (**

**CShfeFtdcExecOrderField \*pExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pExecOrder：**points to the CShfeFtdcExecOrderField structure, with the structure below:

struct CShfeFtdcExecOrderField {

///Trade date

TShfeFtdcDateType TradingDay;

///Settlement Group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement ID

TShfeFtdcSettlementIDType SettlementID;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local ID of exercise entry

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Quantity

TShfeFtdcVolumeType Volume;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Flag indicating open or close (offset) position

TShfeFtdcOffsetFlagType OffsetFlag;

///Exercise ID

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Exercise entry date

TShfeFtdcDateType InsertDate;

///Exercise entry time

TShfeFtdcTimeType InsertTime;

///Exercise cancellation time

TShfeFtdcTimeType CancelTime;

///Exercise execution result

TShfeFtdcExecResultType ExecResult;

///Clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

///Local Business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///operation date

TShfeFtdcDateType ActionDay;

};

Note: ActionDay is a new field, containing the date value the operation in question happens. The trade date is retrieval from TradingDay field. ActionDay is set to empty before it is available to the public.

**pRspInfo:** points to the response received from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///

TShfeFtdcErrorIDType ErrorID;

///

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the IDspecified by the member system in its query request

**bIsLast**: indicates whether this return is the latest with respect to the nRequestID

### OnRspQryExchangeRate Method

This method is not supported until order requests and clearing with multiple currencies are permitted in the Exchange.

This callback is invoked to handle results returned from the Exchange regarding currency exchange rate queries.

**Function Signature:**

**void OnRspQryExchangeRate(**

**CShfeFtdcRspExchangeRateField \*pRspExchangeRate,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pRspExchangeRate:** points to the CShfeFtdcRspExchangeRateField structure, with the structure below:

struct CShfeFtdcRspExchangeRateField {

///Trade date

TShfeFtdcDateType TradingDay;

///Currency ID

TShfeFtdcCurrencyIDType CurrencyID;

///Currency exchange rate unit

TShfeFtdcRateUnitType RateUnit;

///Currency exchange rate price

TShfeFtdcExRatePriceType RatePrice;

};

**pRspInfo**: points tothe response from the Exchange, with the structure below:

struct CShfeFtdcRspInfoField {

///

TShfeFtdcErrorIDType ErrorID;

///

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the IDspecified by the member system in its exchange rate query request

**bIsLast**: indicates whether this return is the latestwith respect to the nRequestID.

### OnRspQryInformation Method

This callback is invoked to handle responses from the Exchange regarding query requests.

**Function Signature:**

**void OnRspQryInformation(**

**CShfeFtdcInformationField \*pInformation,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameter:**

**pInformation：**points to the CShfeFtdcInformationField, with the structure below:

struct CShfeFtdcInformationField {

///information ID

TShfeFtdcInformationIDType InformationID;

///sequence number

TShfeFtdcSequenceNoType SequenceNo;

///contents

TShfeFtdcContentType Content;

///content length

TShfeFtdcContentLengthType ContentLength;

///flag indicating completion or not

TShfeFtdcBoolType IsAccomplished；

};

**pRspInfo**: points to CShfeFtdcRspInfoField, with the structure below:

struct CShfeFtdcRspInfoField {

///

TShfeFtdcErrorIDType ErrorID;

///

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its query request

**bIsLast**: indicates whether this return is the latest with respect to the nRequestID.

### OnRspAbandonExecOrderInsert Method

**This method is not supported by the business until option trading is open to the public.**

This callback is invoked to handle responses from the Exchange about exercise abandonment requests.

**Function signature：**

**void OnRspAbandonExecOrderInsert (**

**CShfeFtdcInputAbandonExecOrderField \*pInputAbandonExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pInputAbandonExecOrder：**points to the exercise abandonment entry request, with the structure below:

struct CShfeFtdcInputAbandonExecOrderField {

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

/// Trader ID

TShfeFtdcUserIDType UserID;

/// Local IDof exercise abandonment

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

///Quantity

TShfeFtdcVolumeType Volume;

///Flag indicating speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

///Long or short position to which the abandonment applies. Only long position holders can request exercise abandonment

TShfeFtdcPosiDirectionType PosiDirection;

///Business Unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the request, with the structure below:

struct CShfeFtdcRspInfoField {

///error code

TShfeFtdcErrorIDType ErrorID;

///error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its exercise abandonment entry request

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRspAbandonExecOrderAction Method

**This method is not supported by the business until option trading is open to the public.**

This callback is invoked to handle responses from the Exchange about acting on existing exercise abandonments.

**Function signature：**

**void OnRspAbandonExecOrderAction (**

**CShfeFtdcAbandonExecOrderActionField \*pAbandonExecOrderAction,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pAbandonExecOrderAction**：points to the request to abandon an exercise, with the structure below:

struct CShfeFtdcAbandonExecOrderActionField {

///Exercise Abandonment ID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

///Exercise Abandonment Local ID

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

///Flag of action

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local ID of the operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response from the Exchange regarding the request

struct CShfeFtdcRspInfoField {

///error code

TShfeFtdcErrorIDType ErrorID;

///error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the Exchange in its action request to abandon an exercise.

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryAbandonExecOrder Method

**This method is not supported by the business until option trading is permitted in the Exchange.**

This callback is invoked to handle responses from the Exchange about querying exercise abandonments.

**Function signature：**

**void OnRspQryAbandonExecOrder (**

**CShfeFtdcAbandonExecOrderField \*pAbandonExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters：**

**pAbandonExecOrder：**points to the request to query an exercise abandonment, with the structure below:

struct CShfeFtdcAbandonExecOrderField {

///trade date

TShfeFtdcDateType TradingDay;

/// settlement group code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

/// settlement ID

TShfeFtdcSettlementIDType SettlementID;

/// contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// Exercise Abandonment Local ID

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// quantity

TShfeFtdcVolumeType Volume;

/// Flag indicating speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

///long or short position this abandonment applies to. Only long position holders can request exercise abandonment

TShfeFtdcPosiDirectionType PosiDirection;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

/// Exercise Abandonment ID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

/// exercise abandonment entry date

TShfeFtdcDateType InsertDate;

/// exercise abandonment entry time

TShfeFtdcTimeType InsertTime;

/// exercise abandonment time

TShfeFtdcTimeType CancelTime;

/// Abandonment execution result

TShfeFtdcExecResultType AbandonExecResult;

/// clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

/// operation date

TShfeFtdcDateType ActionDay;

};

**pRspInfo**：points to the response from the Exchange regarding the query request, with the structure below:

struct CShfeFtdcRspInfoField {

///error code

TShfeFtdcErrorIDType ErrorID;

///error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID specified by the member system in its query request

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRtnAbandonExecOrder method

**This method is not supported by the business until option trading is permitted in the Exchange.**

This callback is invoked to handle responsesfrom the Exchange about execution result of exercise abandonment.

**Function prototype：**

**void OnRtnAbandonExecOrder(**

**CShfeFtdcAbandonExecOrderField \*pAbandonExecOrder);**

**Parameter：**

**pAbandonExecOrder：**points to the exercise abandonment request, with the structure below:

struct CShfeFtdcAbandonExecOrderField {

///trade date

TShfeFtdcDateType TradingDay;

/// settlement group code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

/// settlement ID

TShfeFtdcSettlementIDType SettlementID;

/// contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// Trader ID

TShfeFtdcUserTransactionProcessing IDType UserID;

/// Abandonment Execution declaration Local ID

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// quantity

TShfeFtdcVolumeType Volume;

/// Flag indicating speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

/// long or short position this abandonment applies to. Only long position holder can request abandonment.

TShfeFtdcPosiDirectionType PosiDirection;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// business local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

/// Exercise AbandonmentID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

/// Exercise abandonment entry date

TShfeFtdcDateType InsertDate;

/// Exercise abandonment entry time

TShfeFtdcTimeType InsertTime;

/// Exercise abandonment cancellation time

TShfeFtdcTimeType CancelTime;

/// execution result of abandonment

TShfeFtdcExecResultType AbandonExecResult;

/// clearing member ID

TShfeFtdcParticipantIDType ClearingPartID;

/// operation day

TShfeFtdcDateType ActionDay;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

### OnErrRtnAbandonExecOrderInsert method

**This method is not supported by the business until option trading is permitted in the Exchange.**

This callback is invoked to handle error received from the Exchange about adding exercise abandonment entry.

**Function prototype：**

**void OnErrRtnAbandonExecOrderInsert(**

**CShfeFtdcInputAbandonExecOrderField \*pInputAbandonExecOrder,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pInputAbandonExecOrder：**points to the exercise abandonment entry, with the structure below:

struct CShfeFtdcInputAbandonExecOrderField {

/// contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

///trader ID

TShfeFtdcUserIDType UserID;

/// local ID of exercise abandonment

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// quantity

TShfeFtdcVolumeType Volume;

/// Flag indicating speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

/// long or short position the abandonment applies to. Only long position holder can request exercise abandonment.

TShfeFtdcPosiDirectionType PosiDirection;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// business local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange

struct CShfeFtdcRspInfoField {

/// error code

TShfeFtdcErrorIDType ErrorID;

/// error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnErrRtnAbandonExecOrderAction Method

**This method is not supported by the business until option trading is permitted in the Exchange.**

This callback handles error received from the Exchange regarding acting on existing exercise abandonment.

**Function signature：**

**void OnErrRtnAbandonExecOrderAction(**

**CShfeFtdcAbandonExecOrderActionField \*pAbandonExecOrderAction,**

**CShfeFtdcRspInfoField \*pRspInfo);**

**Parameters：**

**pAbandonExecOrderAction：**points to the action request on existing exercise abandonments, with the structure below:

struct CShfeFtdcAbandonExecOrderActionField {

/// exercise abandonment ID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

/// exercise abandonment Local ID

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// flag of actiontype

TShfeFtdcActionFlagType ActionFlag;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// traderID

TShfeFtdcUserIDType UserID;

/// local ID of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// business local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**pRspInfo**：points to the response received from the Exchange regarding the request

struct CShfeFtdcRspInfoField {

/// error code

TShfeFtdcErrorIDType ErrorID;

/// error message

TShfeFtdcErrorMsgType ErrorMsg;

};

### OnRspQuoteDemand method

**This method is not supported by the business until market making on options is permitted in the Exchange.**

This callback is invoked to handle responses from the Exchange regarding requests for quotes (RFQ).

**Function prototype:**

**void OnRspQuoteDemand(**

**CShfeFtdcQuoteDemandInfoField \*pQuoteDemandInfo,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pQuoteDemandInfo：**points to the quote entry demand, with the structure below:

struct CShfeFtdcQuoteDemandInfoField {

///trade date

TShfeFtdcDateType TradingDay;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// contract code

TShfeFtdcInstrumentIDType InstrumentID;

/// quote entry request local ID

TShfeFtdcOrderLocalIDType QuoteDemandLocalID;

/// request time

TShfeFtdcTimeType DemandTime;

/// action day

TShfeFtdcDateType ActionDay;

};

**pRspInfo**：points to the response received from the Exchange regarding the quote entry request

struct CShfeFtdcRspInfoField {

///error code

TShfeFtdcErrorIDType ErrorID;

///error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the IDspecified by the member system in its quote entry request

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRtnQuoteDemandNotify method

**This method is not supported by the business until market making on options is permitted in the Exchange.**

This callback is invoked to handle notifications distributed from the Exchange regarding RFQ. The Exchange will notify authorized market makers when RFQ is received.

**Function signature:**

**void OnRtnQuoteDemandNotify(**

**CShfeFtdcQuoteDemandNotifyField \*pQuoteDemandNotify);**

**Parameters：**

**pQuoteDemandNotify：**points to the quote notification object, with the structure below:

struct CShfeFtdcQuoteDemandNotifyField {

/// contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// RFQ date

TShfeFtdcDateType DemandDay;

/// RFQ time

TShfeFtdcTimeType DemandTime;

};

## CShfeFtdcTraderApi Interfaces

The **CShfeFtdcTraderApi** interface contains many functions such as order and quote entry/cancellation/suspension/reactivation/update/query, fill query, member and client query, member/client position query, instrument query, instrument trading status query, and Exchange advisory query.

The Exchange sets a threshold for the maximum permitted requests per second at session level. If the member system exceeds the maximum permitted requests, it will receive an error message. For the value of the threshold, please contact the Exchange.

### CreateFtdcTraderApi Method

This method is invoked to create an instance of the **CShfeFtdcTraderApi.** This instance cannot be obtained from the “new” function.

**Function Signature:**

**static CShfeFtdcTraderApi \*CreateFtdcTraderApi(const char \*pszFlowPath = "");**

**Parameter:**

**pszFlowPath**: points to a file directory to store the response from the Trading System. The default value is the current directory.

**Return Value：**

The invocation return points to an instance of**CShfeFtdcTraderApi**.

### GetVersion Method

This method is invoked to get the API version.

**Function Signature:**

**const char \*GetVersion(int &nMajorVersion, int &nMinorVersion) ;**

**Parameter:**

nMajorVersion: returns the primary version number.

nMinorVersion: returns the secondary version number.

**Returned Value：**

This returns a constant string of the current API version.

### Release Method

This method is invoked to release an instance of the **CshfeFtdcTraderApi.** The release cannot be achieved by the “delete” method.

**Function Signature:**

**void Release()；**

### Init Method

This method is invoked to establish a connection from the member system to the Exchange system. Once the connection is built, the member system can login.

**Function Signature:**

**void Init()；**

### Join Method

This method is invoked from the member system waiting for the completion of an instance.

**Function Signature:**

**void Join()；**

### GetTradingDay Method

This method is invoked to get the current trade date after the member system successfully logs in to the Exchange system.

**Function Signature:**

**const char \*GetTradingDay()；**

**Return Value:**

This returns the constant characters of the date.

### RegisterSpi Method

This method is invoked to register an instance of**CShfeFtdcTraderSpi** to handle events.

**Function Signature:**

**void RegisterSpi(CShfeFtdcTraderSpi \*pSpi) ;**

**Parameter:**

pSpi: points to the instance of **ShfeFtdcTraderSpi**

### RegisterFront Method

This method is invoked to register the address of the Exchange’s trading gateways. The Exchange may have multiple trading gateways. The member system can register multiple trading gateways.

This method must be invoked before the **Init** Method.

**Function Signature:**

**void RegisterFront(char \*pszFrontAddress);**

**Parameter:**

**pszFrontAddress**：points to the network address of the Exchange’s trading gateway. The address is in the format of “**protocol://ipaddress:port**”, e.g. “tcp://127.0.0.1:17001”. “tcp” refers to the transmission protocol, “127.0.0.1” is the server address, and “17001” is the port number.

### RegisterNameServer Method

This method is invoked to register the network address of the Exchange’s **NameServer,** which conveys a list of the Exchange’s trading gateways. The Exchange’s system may have multiple NameServers. The member system can register multiple NameServers.

This method must be invoked before the **Init** Method.

**Function Signature:**

**void RegisterNameServer (char \*pszNsAddress);**

**Parameter:**

**pszNsAddress**: points to the network addresses of the Exchange’s NameServer. The network address is in the format of “**protocol://ipaddress:port**”,e.g. ”tcp://127.0.0.1:17001”.“tcp” refers to the transmission protocol, “127.0.0.1” is the server address, and “17001” is the server port number.



### RegisterCryptAlgorithm Method

This method is invoked to register the encryption algorithm of the client’s trading software. The software vendor will be certified by the Exchange, and the encryption key is supplied by the Exchange.

This method is effective only after the Exchange delares a certification requirement. It should be invoked before the Init () method.

**Function Signature:**

**int RegisterCryptAlgorithm(const char\* pszKeyFileName, const char \*pszKeyFilePassword,**

**TShfeFtdcEncryptTypeType nEncryptType = SHFE\_FTDC\_ET\_RSA);**

**Parameters：**

**pszKeyFileName**：file name of the encryption key of the software vendor；

**pszKeyFilePassword**：password for the encryption key file. If the software vendor does not set a password, this value is set to NULL;

nEncryptType：encryption type.

* SHFE\_FTDC\_ET\_RSA：RSA encryption

**Return Value：**

* 0，success
* -1，failure to read the key file；
* -2，failure to parse the key file；
* -3，the encryption type is not supported.

### RegisterCertificateFile Method

This method is invoked to register the client’s certificate file.

This method is effective after the Exchange announces SSL connection, and should be invoked before the Init () method.

**Function Signature：**

**int RegisterCertificateFile(const char\* pszCertFileName,const char \*pszKeyFileName,const char \*pszCaFileName, const char \*pszKeyFilePassword);**

**Paramters：**

**pszCertFileName**：client’s certificate file name

**pszKeyFileName**：client’s private key file name

**pszCaFileName**：trusted CA certificate file name

**pszKeyFilePassword**：password of the client’s private key file

**返回值：**

* 0: success
* -1: failure to load the trusted CA certificate
* -2: failure to load the client’s certificate
* -3: failure to load the client’s private key
* -4: failure to verify the client’s certificate

### SetHeartbeatTimeout Method

This method is invoked to set a heartbeat timeout for network communication. After TraderAPI connects to the Exchange’s trading system, it will regularly send heartbeat messages to detect abnormal communication. **The Exchange suggests setting the timeout between 10 seconds and 30 seconds.**

**Function Signature:**

**virtual void SetHeartbeatTimeout(unsigned int timeout);**

**Parameter:**

**Timeout:** heartbeat timeout (in seconds). If a message is not received from the Exchange’s trading system after “timeout/2” seconds, **CShfeFtdcTraderApi::OnHeartBeatWarning()**is triggered. If a message is not received from the Exchange’s trading system after the “timeout” seconds, the connection will be aborted and **CShfeFtdcTraderApi ::OnFrontDisconnected()** will be triggered.

**For the heartbeat mechanism, please refer to Part I Section 4.8.**

### OpenRequestLog Method

This method is invoked to open the log file for storing all requests sent to the Exchange’s trading system.

**Function Signature:**

**virtual int OpenRequestLog(const char \*pszReqLogFileName);**

**Parameter:**

pszReqLogFileName：the name of the log file

### OpenResponseLog Method

This method is invoked to open the log file to store all responses sent from the Exchange’s trading system, including order acknowledgements, trade notifications, and responses from other requests.

**Function Signature:**

**virtual int OpenResponseLog(const char \*pszRspLogFileName);**

**Parameter:**

pszRspLogFileName：the name of the log file

### SubscribePrivateTopic Method

This method is invoked to subscribe to a member-specific private stream. This method must be invoked before the **Init** method. If this method is not invoked, data conveyed from the private stream will not be received.

**Function Signature:**

**void SubscribePrivateTopic(TE\_RESUME\_TYPE nResumeType);**

**Parameter:**

**nResumeType：**message replay methods in private streams:

* TERT\_RESTART：replay from the start of thecurrent trade date
* TERT\_RESUME：resume from where it stopped last time. The **Exchange recommends this method so that the member system can acquire the lost data. Once it is fully recovered, the member system can proceed with subsequent responses.**
* TERT\_QUICK：receive from the moment the member system logs in. The **Exchange does not recommend this method, as it does not help to restore the lost data.**

### SubscribePublicTopic Method

This method is invoked to subscribe to the public stream, and must be invoked before the **Init** Method. With this subscription, data from the public stream will not be received.

**Function Signature:**

**void SubscribePublicTopic(TE\_RESUME\_TYPE nResumeType);**

**Parameter:**

**nResumeType：**message replay methods in the public stream

* TERT\_RESTART：replay from the start of the current trade date
* TERT\_RESUME：resume from where it stopped last time
* TERT\_QUICK：receive from the moment the member system logs in

### SubscribeUserTopic Method

This method is invoked to subscribe to a trader-specific private stream. This method must be invoked before the **Init** method. Without this subscription, trader-specific private stream data will not be received.

**Function Signature:**

**void SubscribeUserTopic(TE\_RESUME\_TYPE nResumeType);**

**Parameter:**

**nResumeType：**message replay method in private stream (similar to **Section 2.2.13** above):

* TERT\_RESTART：replay from the start of the current trade date
* TERT\_RESUME：resume from where it stopped last time. The **Exchange recommends this method so that the member system can restore all lost data before proceeding with subsequent responses.**
* TERT\_QUICK：receive the message from the moment member system logs in. **The Exchange does not recommend this method, as it does not help to restore the lost data.**

### ReqUserLogin Method

This method is invoked for the member system to log in.

**Function Signature:**

**int ReqUserLogin(**

**CShfeFtdcReqUserLoginField \*pReqUserLoginField,**

**int nRequestID)；**

**Parameter:**

**pReqUserLoginField**：points to the login request, with the structure below:

struct CShfeFtdcReqUserLoginField {

///Trade date

TShfeFtdcDateType TradingDay;

///Trader code

TShfeFtdcUserIDType UserID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Password

TShfeFtdcPasswordType Password;

///Client side trading system information

TShfeFtdcProductInfoType UserProductInfo;

///product information with respect to the interface; Not used

TShfeFtdcProductInfoType InterfaceProductInfo;

///Protocol information,not used

TShfeFtdcProtocolInfoType ProtocolInfo;

///Data center ID

TShfeFtdcDataCenterIDType DataCenterID;

};

**Client is required to fill the field of "UserProductInfo" in terms of ISV and version etc. For example, ”SFIT Trader V100” represents the ISV and version numberof the trading software.**

If the member system maintains the sequence number of responses, it must fill in the “TradingDay" and "DataCenterID” fields with values obtained from responses to previous login requests. For a first-time login or replay this is not required; the member system can input an empty string (“”) into "TradingDay" and 0 into “DataCenterID”.

**nRequestID**：requests the ID specified by the member system in its login request.

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqUserLogout Method

This method is for member system to log out.

**Function Signature:**

**int ReqUserLogout(**

**CShfeFtdcReqUserLogoutField \*pReqUserLogout,**

**int nRequestID)；**

**Parameter:**

**pReqUserLogout**：points to the logout request with the structure below:

struct CShfeFtdcReqUserLogoutField {

///Trading User ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

};

**nRequestID**：requests the ID specified by the member system in its logout request

**Returned Value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqUserPasswordUpdate Method

This method is invoked for member system to update password.

**Function Signature:**

**int ReqUserPasswordUpdate(**

**CShfeFtdcUserPasswordUpdateField \*pUserPasswordUpdate,**

**int nRequestID)；**

**Parameters:**

**pUserPasswordUpdate**：points to the password update request, with the structure below:

struct CShfeFtdcUserPasswordUpdateField {

///Trader ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Old Password

TShfeFtdcPasswordType OldPassword;

///New Password

TShfeFtdcPasswordType NewPassword;

};

**nRequestID**：requests the ID specified by the member system in its password-update request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqSubscribeTopic Method

This method is invoked to request a subscription to a topic (subject), and should be invoked after login.

**Function Signature:**

**int ReqSubscribeTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**int nRequestID)；**

**Parameters:**

**pDissemination**：points to the subscription object, comprising the topic of interest and the starting sequence number, with the structure below:

struct CShfeFtdcDisseminationField {

///Sequence series number

TShfeFtdcSequenceSeriesType SequenceSeries;

///Sequence number

TShfeFtdcSequenceNoType SequenceNo;

};

SequenceSeries：topic to be subscribed to

SequenceNo：=-1 representing a request for transmission in a “QUICK” manner; otherwise it represents the starting sequence number

**nRequestID**：requests the ID specified by the member system in its subscription request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryTopic Method

This method is invoked to request a topic query, and should be invoked after login.

**Function Signature:**

**int ReqQryTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**int nRequestID)；**

**Parameter:**

**pDissemination**：points to the topic query object, comprising the topic of interest, with the structure below:

struct CShfeFtdcDisseminationField {

///Sequence Series

TShfeFtdcSequenceSeriesType SequenceSeries;

///Sequence Number

TShfeFtdcSequenceNoType SequenceNo;

};

SequenceSeries：topic to be queried

SequenceNo：not required

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqOrderInsert Method

This method is invoked for the member system to add a new order entry.

**Function Signature:**

**int ReqOrderInsert(**

**CShfeFtdcInputOrderField \*pInputOrder,**

**int nRequestID)；**

**Parameters:**

**pInputOrder**：points to the order entry object, with the structure below:

struct CShfeFtdcInputOrderField {

///Order number, returned by the Exchange’s trading system.

TShfeFtdcOrderSysIDType OrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// order type;currently it supports "limit order" only

TShfeFtdcOrderPriceTypeType OrderPriceType;

///Buy-sell direction

TShfeFtdcDirectionType Direction;

///Flag indicating whether to open orcloseposition

TShfeFtdcCombOffsetFlagType CombOffsetFlag;

///Flag indicating speculation or hedge

TShfeFtdcCombHedgeFlagType CombHedgeFlag;

///Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Order validity. Currently it supports "good to the day" and "Immediately filledor cancelled"

TShfeFtdcTimeConditionType TimeCondition;

///GTD DATE; not used

TShfeFtdcDateType GTDDate;

///Match quantity type; it supports "random(AV)". If the order validity is set to IOC, it also supports“fully filled(CV)” and “minimum filled quantity(MV)”

TShfeFtdcVolumeConditionType VolumeCondition;

///The minimum filled quantity.It is set when VolumeCondition is set to “minimum filled quality(MV)”

TShfeFtdcVolumeType MinVolume;

///Trigger conditions; currently it supports "immediately" only.

TShfeFtdcContingentConditionType ContingentCondition;

///trigger price, not used

TShfeFtdcPriceType StopPrice;

///Reasons for enforced offset; currently it supports "NotForceClose"

TShfeFtdcForceCloseReasonType ForceCloseReason;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag indicating whether to suspend or not when unexpected disconnection occurs

TShfeFtdcBoolType IsAutoSuspend;

///Business unit; not used

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

\* OrderLocalID：the local order number is maintained by the member system. This number is incremented unidirectional. After each login, the member system can parse the returned "*CShfeFtdcRspUserLoginField*" from “OnRspUserLogin” to retrieve the maximum order number submitted last time.Sincethe Exchange system compares OrderLocalID alphanumerically, the member system needs to fill "*TShfeFtdcOrderLocalIDType*" with a value.

**nRequestID**：requests the ID specified by the member system in its order entry request

**Returned Value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqOrderAction Method

This method is invoked to perform an action on the submitted order, including a cancellation, suspension, reactivation, and modification. Currently, modification via this method is not supported.

**Function Signature:**

**int ReqOrderAction(**

**CshfeFtdcOrderActionField \*pOrderAction,**

**int nRequestID)；**

**Parameters:**

**pOrderAction**: points to the order action object, with the structure below:

struct CshfeFtdcOrderActionField {

///Order number

TShfeFtdcOrderSysIDType OrderSysID;

///Local order number

TShfeFtdcOrderLocalIDType OrderLocalID;

///Flag of action. It currently supports “deletion, suspension, and activation"

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price，not used

TShfeFtdcPriceType LimitPrice;

///Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

/// Quantitychange, not used

TShfeFtdcVolumeType VolumeChange;

///Business unit，not used

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

\* OrderSysID and OrderLocalID are both used to designate the target order. The member system can input a value into one of them.

\* OrderLocalID：the local order number is maintained by the member system. This number is incremented unidirectional. After each login, the member system can parse the returned "*CShfeFtdcRspUserLoginField*" from “OnRspUserLogin” to retrieve the maximum order number submitted last time. Since the Exchange system compares OrderLocalID alphanumerically, the member system needs to fill "*TShfeFtdcOrderLocalIDType*" with a value.

**nRequestID**: requests the ID specified by the member system in its order action request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQuoteInsert Method

**This method is not supported by the business until the quote request is open to the public.**

This method is invoked by the member system to add a new quote entry.

**Function Signature：**

**int ReqQuoteInsert(**

**CShfeFtdcInputQuoteField \*pInputQuote,**

**int nRequestID)；**

**Parameters：**

**pInputQuote**：points to the quote entry object, with the structure below:

struct CShfeFtdcInputQuoteField {

///Quote number returned from the Exchange’s trading system.

TShfeFtdcQuoteSysIDType QuoteSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Quantity

TShfeFtdcVolumeType Volume;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Local quote Number

TShfeFtdcQuoteLocalIDType QuoteLocalID;

///Business unit,not used

TShfeFtdcBusinessUnitType BusinessUnit;

///Flag indicating whether to open or close (offset)buy-side position

TShfeFtdcCombOffsetFlagType BidCombOffsetFlag;

///Flag indicating hedge or speculation on buy-side

TShfeFtdcCombHedgeFlagType BidCombHedgeFlag;

///Buy-side price

TShfeFtdcPriceType BidPrice;

///Flag indicating whether to open orclose sell-sideposition

TShfeFtdcCombOffsetFlagType AskCombOffsetFlag;

///Flag indicating whether to speculate or hedge on sell-side

TShfeFtdcCombHedgeFlagType AskCombHedgeFlag;

///Sell-side price

TShfeFtdcPriceType AskPrice;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specified by the member system in its quote entry request

**Return value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQuoteAction Method

**This method is not supported by the business until the quote request is open to the public.**

This method is invoked by the member system to request an action on a submitted quote, including a cancellation, suspension, reactivation, and modification.

**Function Signature：**

**int ReqQuoteAction(**

**CShfeFtdcQuoteActionField \*pQuoteAction,**

**int nRequestID)；**

**Parameters：**

**pQuoteAction**：points to the quote action object, with the structure below:

struct CShfeFtdcQuoteActionField {

///Quote number

TShfeFtdcQuoteSysIDType QuoteSysID;

///Local quote number

TShfeFtdcOrderLocalIDType QuoteLocalID;

///Flag of action

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

/// Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specified by the member system in its quote action request

**Return value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqExecOrderInsert Method

**This method is not supported by the business until option trading is open to the public.**

It is used to request an exercise.

**Function Signature：**

**int ReqExecOrderInsert(**

**CshfeFtdcInputExecOrderField \*pInputExecOrder,**

**int nRequestID)；**

**Parameters：**

**pInputExecOrder**：points tothe exercise request object, with the structure below:

struct CShfeFtdcInputExecOrderField {

///Contract number

TShfeFtdcInstrumentIDType InstrumentID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local exercise number

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Quantity

TShfeFtdcVolumeType Volume;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specifiedby the member system in its exercise entry request

**Return value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqExecOrderAction Method

**This method is not supported by the business until option trading is open to the public.**

It is used to request an action on the submitted exercise.

**Function Signature：**

**int ReqExecOrderAction(**

**CShfeFtdcExecOrderActionField \*pExecOrderAction,**

**int nRequestID);**

**Parameters：**

**pExecOrderAction**：points to the exercise action request, with the structure below:

struct CShfeFtdcExecOrderActionField {

///exercise number

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Local exercise number

TShfeFtdcOrderLocalIDType ExecOrderLocalID;

///Flag of action

TShfeFtdcActionFlagType ActionFlag;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specified by the member system in its exercise action request

**Return value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqAdminOrderInsert method

The efficacy of credit limit setting in this method is determined by the Exchange, and will be communicated with a notification.

This method is invoked to insert an administrative order.

**Interface Prototype：**

**int ReqAdminOrderInsert(**

**CShfeFtdcInputAdminOrderField \*pInputAdminOrder,**

**int nRequestID);**

**Parameters：**

**pInputAdminOrder**：points to the administrative order object, with the structure below:

struct CShfeFtdcInputAdminOrderField {

///instrument

TShfeFtdcInstrumentIDType InstrumentID;

///administrative order command

TShfeFtdcAdminOrderCommandFlagType AdminOrderCommand;

///clearing member (participant) ID

TShfeFtdcParticipantIDType ClearingPartID;

///trading member (participant) ID

TShfeFtdcParticipantIDType ParticipantID;

///amount

TShfeFtdcMoneyType Amount;

///settlement group ID

TShfeFtdcSettlementGroupIDType SettlementGroupID;

};

**nRequestID**：requests the ID of an execution declaration. It is designated and maintained by the client side.

**Return Value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests exceeds the permitted maximum

### ReqQryPartAccount Method

This method is invoked to query a member’s margin (maintained by the trading system). This query will be removed if the execution exceeds the timeout (this process applies to all subsequently described query requests).

**Function Signature:**

**int ReqQryPartAccount(**

**CShfeFtdcQryPartAccountField \*pQryPartAccount,**

**int nRequestID)；**

**Parameters:**

**pQryPartAccount**: points to the margin query request, with the structure below:

struct CShfeFtdcQryPartAccountField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///Capital account，optional

TShfeFtdcAccountIDType AccountID;

};

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryOrder Method

This method is invoked to query an order.

**Function Signature:**

**int ReqQryOrder(**

**CShfeFtdcQryOrderField \*pQryOrder,**

**int nRequestID)；**

**Parameters:**

**pQryOrder**: points to the order query request. The fields in this request are in logic AND relationship with each other. If the field is empty, it is ignored and will not participate in the query.

struct CShfeFtdcQryOrderField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///Order number, optional

TShfeFtdcOrderSysIDType OrderSysID;

///Contract code,optional

TShfeFtdcInstrumentIDType InstrumentID;

///Client code,optional

TShfeFtdcClientIDType ClientID;

///Trader ID,optional

TShfeFtdcUserIDType UserID;

///The starting time,optional

TShfeFtdcTimeType TimeStart;

///The ending time,optional

TShfeFtdcTimeType TimeEnd;

};

**nRequestID：requests the IDspecified by the member system in its query request**

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryQuote Method

**This method is not supported by the business until the quote request is open to the public.**

It is used to query a quote.

**Function Signature：**

**int ReqQryQuote(**

**CShfeFtdcQryQuoteField \*pQryQuote,**

**int nRequestID)；**

**Parameters：**

**pQryQuote**：points to the quote query object, with the structure below:

struct CShfeFtdcQryQuoteField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///Quote number

TShfeFtdcQuoteSysIDType QuoteSysID;

///Client code

TShfeFtdcClientIDType ClientID;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///TraderID

TShfeFtdcUserIDType UserID;

};

**nRequestID**：requests the ID specified by the member system in its quote query request

**Return value：**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryTrade Method

This method is invoked to request a trade query.

**Function Signature:**

**int ReqQryTrade(**

**CShfeFtdcQryTradeField \*pQryTrade,**

**int nRequestID)；**

**Parameters:**

**pQryTrade**：points to the trade query request, with the structure below:

struct CShfeFtdcQryTradeField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///The starting contract code, optional

TShfeFtdcInstrumentIDType InstIDStart;

///The ending contract code, optional

TShfeFtdcInstrumentIDType InstIDEnd;

///Trade number,optional

TShfeFtdcTradeIDType TradeID;

///Client code,optional

TShfeFtdcClientIDType ClientID;

///Trader ID,optional

TShfeFtdcUserIDType UserID;

///The starting time,optional

TShfeFtdcTimeType TimeStart;

///The finishing time,optional

TShfeFtdcTimeType TimeEnd;

};

**NRequestID:** requests the ID specified in the member system in its trade query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryClient Method

This method is invoked to query member/client information.

**Function Signature:**

**int ReqQryClient(**

**CShfeFtdcQryClientField \*pQryClient,**

**int nRequestID)；**

**Parameters:**

**pQryClient**：points to the member/client query object, with the structure below:

struct CShfeFtdcQryClientField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///The starting client code, optional

TShfeFtdcClientIDType ClientIDStart;

///The ending client code, optional

TShfeFtdcClientIDType ClientIDEnd;

};

**nRequestID**：requests the ID specified by the member system in its member/client query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryPartPosition Method

This method is invoked to query a member’s position.

**Function Signature:**

**int ReqQryPartPosition(**

**CShfeFtdcQryPartPositionField \*pQryPartPosition,**

**int nRequestID)；**

**Parameters:**

**pQryPartPosition**：points to the member’s position query request, with the structure below:

struct CShfeFtdcQryPartPositionField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///The starting contract code, optional

TShfeFtdcInstrumentIDType InstIDStart;

///The ending contract code, optional

TShfeFtdcInstrumentIDType InstIDEnd;

};

**nRequestID**：requests the ID specified by the client in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryClientPosition Method

This method is invoked to query the client’s position.

**Function Signature:**

**int ReqQryClientPosition(**

**CShfeFtdcQryClientPositionField \*pQryClientPosition,**

**int nRequestID);**

**Parameters:**

**pQryClientPosition**：points to the client’s position query request, with the structure below:

struct CShfeFtdcQryClientPositionField {

///The starting member code

TShfeFtdcParticipantIDType PartIDStart;

///The ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///The starting client code, optional

TShfeFtdcClientIDType ClientIDStart;

///The ending client code, optional

TShfeFtdcClientIDType ClientIDEnd;

///The starting contract code, optional

TShfeFtdcInstrumentIDType InstIDStart;

///The ending contract code, optional

TShfeFtdcInstrumentIDType InstIDEnd;

///Client type,optional

TShfeFtdcClientTypeType ClientType;

};

**nRequestID**：requests the IDspecified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryInstrument Method

This method is invoked to query an instrument.

**Function Signature:**

**int ReqQryInstrument(**

**CshfeFtdcQryInstrumentField \*pQryInstrument,**

**int nRequestID)；**

**Parameters:**

**pQryInstrument**：points to the instrument query request, with the structure below:

struct CshfeFtdcQryInstrumentField {

///Settlement group’s code, optional

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Product suite’s code, optional

TShfeFtdcProductGroupIDType ProductGroupID;

///Product code, optional

TShfeFtdcProductIDType ProductID;

///Contract code ,optional

TShfeFtdcInstrumentIDType InstrumentID;

};

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryInstrumentStatus Method

This method is invoked to query the status of an instrument.

**Function Signature:**

**int ReqQryInstrumentStatus(**

**CShfeFtdcQryInstrumentStatusField \*pQryInstrumentStatus,**

**int nRequestID)；**

**Parameters:**

**pQryInstrumentStatus**：points to the instrument request, with the structure below:

struct CShfeFtdcQryInstrumentStatusField {

///The starting contract code, optional

TShfeFtdcInstrumentIDType InstIDStart;

///The ending contract code, optional

TShfeFtdcInstrumentIDType InstIDEnd;

};

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryMarketData Method

This method is invoked to query market data.

**Function Signature:**

**int ReqQryMarketData(**

**CShfeFtdcQryMarketDataField \*pQryMarketData,**

**int nRequestID)；**

**Parameters:**

**pQryMarketData**：points to the market data query object, with the structure below:

struct CShfeFtdcQryMarketDataField {

///Product code,optional

TShfeFtdcProductIDType ProductID;

///Contract code ,optional

TShfeFtdcInstrumentIDType InstrumentID;

};

**nRequestID**：requests the IDspecified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryBulletin Method

This method is invoked to query an exchange advisory.

**Function Signature:**

**int ReqQryBulletin(**

**CShfeFtdcQryBulletinField \*pQryBulletin,**

**int nRequestID)；**

**Parameters:**

**pQryBulletin**：points to the advisory query object, with the structure below:

struct CShfeFtdcQryBulletinField {

///Trade Date, optional

TShfeFtdcDateType TradingDay;

///market ID, optional

TShfeFtdcMarketIDType MarketID;

///bulletin ID, optional

TShfeFtdcBulletinIDType BulletinID;

///bulletin type, optional

TShfeFtdcNewsTypeType NewsType;

///urgency level, optional

TShfeFtdcNewsUrgencyType NewsUrgency;

};

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryMBLMarketData Method

This method is invoked to query book information of an instrument.

**Function Signature:**

**int ReqQryMBLMarketData(**

**CShfeFtdcQryMBLMarketDataField \*pQryMBLMarketData,**

**int nRequestID);**

**Parameters:**

**pQryMBLMarketData**：points to the instrument query object. The structure is shown below:

struct CShfeFtdcQryMBLMarketDataField {

///starting contract ID, optional

TShfeFtdcInstrumentIDType InstIDStart;

///ending contract ID, optional

TShfeFtdcInstrumentIDType InstIDEnd;

///buy/sell direction, optional

TShfeFtdcDirectionType Direction;

};

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryHedgeVolume Method

This method is invoked to query the hedge quota.

**Function Signature:**

**int ReqQryHedgeVolume(**

**CShfeFtdcQryHedgeVolumeField \*pQryHedgeVolume,**

**int nRequestID);**

**Parameters:**

**pQryHedgeVolume**：points to the hedge quota query object, with the structure below:

struct CshfeFtdcQryHedgeVolumeField {

{

///starting member ID

TShfeFtdcParticipantIDType PartIDStart;

///ending member ID

TShfeFtdcParticipantIDType PartIDEnd;

///starting client ID, optional

TShfeFtdcClientIDType ClientIDStart;

///ending client ID, optional

TShfeFtdcClientIDType ClientIDEnd;

///starting instrument ID, optional

TShfeFtdcInstrumentIDType InstIDStart;

///ending instrument ID, optional

TShfeFtdcInstrumentIDType InstIDEnd;

};

**nRequestID**: requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqCombOrderInsert Method

**This method is not currently open to the public.**

It is invoked to enter a combo order (spread).

**Function Signature：**

**int ReqCombOrderInsert (**

**CShfeFtdcInputCombOrderField \*pInputCombOrder,**

**int nRequestID)；**

**Parameters：**

**pInputCombOrder：**points to the combo order object. The structure is shown below:

struct CShfeFtdcInputCombOrderField {

///combo order number

TShfeFtdcOrderSysIDType CombOrderSysID;

///Member code

TShfeFtdcParticipantIDType ParticipantID;

///Client code

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Price

TShfeFtdcPriceType LimitPrice;

///Quantity

TShfeFtdcVolumeType VolumeTotalOriginal;

///Local order number

TShfeFtdcOrderLocalIDType CombOrderLocalID;

///Business unit

TShfeFtdcBusinessUnitType BusinessUnit;

///Contract code for leg 1

TShfeFtdcInstrumentIDType InstrumentID1;

///Buy/sell direction for leg 1

TShfeFtdcDirectionType Direction1;

///Leg1 multiplier

TShfeFtdcLegMultipleType LegMultiple1;

///Flag indicating whether to open or close(offset) position on leg 1

TShfeFtdcOffsetFlagType OffsetFlag1;

///Flag indicating speculation or hedge on leg 1

TShfeFtdcHedgeFlagType HedgeFlag1;

///Contract code for leg 2

TShfeFtdcInstrumentIDType InstrumentID2;

///Buy/sell direction on leg 2

TShfeFtdcDirectionType Direction2;

///Leg 2 multiplier

TShfeFtdcLegMultipleType LegMultiple2;

///Flag indicating whether to open or close(offset) on leg 2

TShfeFtdcOffsetFlagType OffsetFlag2;

///Flag indicating speculation or hedge on leg 2

TShfeFtdcHedgeFlagType HedgeFlag2;

///Contract code for leg 3

TShfeFtdcInstrumentIDType InstrumentID3;

///Buy/sell direction on leg 3

TShfeFtdcDirectionType Direction3;

///Leg 3 multiplier

TShfeFtdcLegMultipleType LegMultiple3;

///Flag indicating whether to open or close position on leg 3

TShfeFtdcOffsetFlagType OffsetFlag3;

///Flag indicating speculation or hedge on leg 3

TShfeFtdcHedgeFlagType HedgeFlag3;

///Contract code for leg 4

TShfeFtdcInstrumentIDType InstrumentID4;

///Buy/sell direction on leg 4

TShfeFtdcDirectionType Direction4;

///Leg 4 multiplier

TShfeFtdcLegMultipleType LegMultiple4;

///Flag indicating whether to open or close (offset) position on leg 4

TShfeFtdcOffsetFlagType OffsetFlag4;

///Flag indicating speculation or hedge on leg 4

TShfeFtdcHedgeFlagType HedgeFlag4;

///Local business ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specifiedby the member system in its combo order entry request

**Return value：**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryCombOrder Method

**This method is not supported by the business until the combo order is open to the public.**

It is invoked to query a submitted combo order.

**Function Signature：**

**int ReqQryCombOrder (**

**CShfeFtdcQryCombOrderField \*pQryCombOrder,**

**int nRequestID)；**

**Parameters：**

**pQryCombOrder：**points to the combo order query object, with the structure below:

struct CShfeFtdcQryCombOrderField

{

///Starting member ID

TShfeFtdcParticipantIDType PartIDStart;

///Ending member ID

TShfeFtdcParticipantIDType PartIDEnd;

///Combo order ID

TShfeFtdcOrderSysIDType CombOrderSysID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

};

**nRequestID**：requests the ID specifiedby the member system in its combo order query request

**Return value：**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryExecOrder Method

**This method is not supported by the business until option trading is permitted.**

It is invoked to query an exercise.

**Function Signature**:

**int ReqQryExecOrder (**

**CShfeFtdcQryExecOrderField \*pQryExecOrder,**

**int nRequestID)；**

**Parameters:**

**pQryExecOrder：**points to the exercise query object, with the structure below:

struct CShfeFtdcQryExecOrderField

{

///Starting member ID

TShfeFtdcParticipantIDType PartIDStart;

///Ending member ID

TShfeFtdcParticipantIDType PartIDEnd;

///Exercise ID

TShfeFtdcExecOrderSysIDType ExecOrderSysID;

///Instrument ID

TShfeFtdcInstrumentIDType InstrumentID;

///Client ID

TShfeFtdcClientIDType ClientID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Starting time

TShfeFtdcTimeType TimeStart;

///Ending time

TShfeFtdcTimeType TimeEnd;

};

**nRequestID**：requests the ID specifiedby the member system in its exercise query request

**Return value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryExchangeRate Method

**This method is not supported until order entry and clearing in multiple currencies are permitted in the Exchange.**

It is invoked to query an exchange rate.

**Function Signature:**

**int ReqQryExchangeRate (**

**CShfeFtdcQryExchangeRateField \*pQryExchangeRate,**

**int nRequestID)；**

**Parameters:**

**pQryExchangeRate:** points tothe Exchange rate query request, with the structure below:

struct CShfeFtdcQryExchangeRateField

{

///Currency ID

TShfeFtdcCurrencyIDType CurrencyID;

};

**nRequestID**：requests the ID specifiedby the member system in its exchange rate query request

**Return value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryInformation Method

This method is invoked to query information.

**Function Signature:**

**int ReqQryInformation (**

**CShfeFtdcQryInformationField \*pQryInformation,**

**int nRequestID)；**

**Parameters:**

**pQryInformation:** points to the query object, with the structure below:

struct CShfeFtdcQryInformationField

{

///Starting information ID

TShfeFtdcInformationIDType InformationIDStart;

///Ending Information ID

TShfeFtdcInformationIDType InformationIDEnd;

};

**nRequestID**：requests the ID specifiedby the member system in its query request

**Return value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqAbandonExecOrderInsert method

**This method is not supported by the business until option trading is permitted in the Exchange.**

It is invoked to request the abandonment of an exercise, and is available to the buy-side only.

**Function signature：**

**int ReqAbandonExecOrderInsert(**

**CShfeFtdcInputAbandonExecOrderField \*pInputAbandonExecOrder,**

**int nRequestID)；**

**Parameter：**

**pInputAbandonExecOrder**: points to the exercise abandonment request object, with the structure below:

struct CShfeFtdcInputAbandonExecOrderField {

/// contract ID

TShfeFtdcInstrumentIDType InstrumentID;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// local ID of abandonment

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// quantity

TShfeFtdcVolumeType Volume;

/// Flag indicating speculation or hedge

TShfeFtdcHedgeFlagType HedgeFlag;

/// long/short position for this abandonment. Only long position holders can request abandonment

TShfeFtdcPosiDirectionType PosiDirection;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// business local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specified by the member system in its abandonment request

**Return value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqAbandonExecOrderAction method

**This method is not supported until option trading is permitted in exchange.**

This method is invoked to request operation on submitted exercise abandonment.

**Function signature：**

**int ReqAbandonExecOrderAction(**

**CShfeFtdcAbandonExecOrderActionField \*pAbandonExecOrderAction,**

**int nRequestID);**

**Parameter：**

**pAbandonExecOrderAction**: points to the request object for action on exercise abandonment, with the structure below:

struct CShfeFtdcAbandonExecOrderActionField {

/// exercise abandonment ID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

/// exercise abandonment local ID

TShfeFtdcOrderLocalIDType AbandonExecOrderLocalID;

/// flag of action

TShfeFtdcActionFlagType ActionFlag;

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// local number of operation

TShfeFtdcOrderLocalIDType ActionLocalID;

/// business unit

TShfeFtdcBusinessUnitType BusinessUnit;

/// business local ID

TShfeFtdcBusinessLocalIDType BusinessLocalID;

///IP

TShfeFtdcIPAddressType IPAddress;

///Mac

TShfeFtdcMacAddressType MacAddress;

};

**nRequestID**：requests the ID specified by the member system in its action request

**Return value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryAbandonExecOrder method

**This method is not supported.**

It is invoked to query a submitted exercise abandonment.

**Function signature：**

**int ReqQryAbandonExecOrder (**

**CShfeFtdcQryAbandonExecOrderField \*pQryAbandonExecOrder,**

**int nRequestID)；**

**Parameter：**

**pQryAbandonExecOrder：**points to the query request for an exercise abandonment, with the structure below:

struct CShfeFtdcQryAbandonExecOrderField

{

/// starting member code

TShfeFtdcParticipantIDType PartIDStart;

///ending member code

TShfeFtdcParticipantIDType PartIDEnd;

///exercise abandonment ID

TShfeFtdcExecOrderSysIDType AbandonExecOrderSysID;

/// contract code

TShfeFtdcInstrumentIDType InstrumentID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// starting time

TShfeFtdcTimeType TimeStart;

/// ending time

TShfeFtdcTimeType TimeEnd;

};

**nRequestID**：requests the IDspecified by the member system in its query request

**Return value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQuoteDemand method

**This method is not supported until market making on options is permitted in the Exchange.**

It is invoked to request the addition of a new quote.

**Function signature：**

**int ReqQuoteDemand (**

**CShfeFtdcInputQuoteDemandField \*pInputQuoteDemand,**

**int nRequestID)；**

**Parameter：**

**pInputQuoteDemand：**points to the quote entry request, with the structure below:

struct CShfeFtdcInputQuoteDemandField{

/// member code

TShfeFtdcParticipantIDType ParticipantID;

/// client code

TShfeFtdcClientIDType ClientID;

/// trader ID

TShfeFtdcUserIDType UserID;

/// contract code

TShfeFtdcInstrumentIDType InstrumentID;

/// local ID of quote entry

TShfeFtdcOrderLocalIDType QuoteDemandLocalID;

};

**nRequestID**：requests the ID specified by the member system in its quote entry request

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

# TraderAPI—A Development Example

// tradeapitest.cpp :

// An example illustrating the use of CShfeFtdcTraderApi and CShfeFtdcTraderSpi

// This example shows how an order is entered

#include <stdio.h>

#include <windows.h>

#include "FtdcTraderApi.h"

// Member code

TShfeFtdcParticipantIDType g\_chParticipantID;

// TraderID

TShfeFtdcUserIDType g\_chUserID;

class CSimpleHandler : public CShfeFtdcTraderSpi

{

public:

// Constructerto obtain an instance of CShfeFtdcMduserApi

CSimpleHandler(CShfeFtdcTraderApi \*pUserApi) : m\_pUserApi(pUserApi) {}

~CSimpleHandler() {}

//This callback is invoked once the member system connects to the Exchange system. The member system needs to log in

virtual void OnFrontConnected()

{

CShfeFtdcReqUserLoginField reqUserLogin;

// get trading member id (4 digit number)

printf("participantid:");

scanf("%s", &g\_chParticipantID);

strcpy(reqUserLogin.ParticipantID, g\_chParticipantID);

// get trading session ID (a.k.a seat ID)

printf("userid:");

scanf("%s", &g\_chUserID);

strcpy(reqUserLogin.UserID, g\_chUserID);

// get trading session password (a.k.a seat password)

printf("password:");

scanf("%s", &reqUserLogin.Password);

// Send the login request

m\_pUserApi->ReqUserLogin(&reqUserLogin, 0);

}

// This callback is invoked when the member system disconnects from the Exchange system in

virtual void OnFrontDisconnected(int nReason)

{

// When disconnection occurs, API will automatically reconnect, so the member system does not need to handle reconnection

printf("OnFrontDisconnected.\n");

}

// This callback is invoked after the Exchange system responds to the member system’s login request

virtual void OnRspUserLogin(CshfeFtdcRspUserLoginField \*pRspUserLogin, CshfeFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

printf(“OnRspUserLogin:\n”);

printf(“ErrorCode=[%d], ErrorMsg=[%s]\n”, pRspInfo->ErrorID, pRspInfo->ErrorMsg);

printf(“RequestID=[%d], Chain=[%d]\n”, nRequestID, bIsLast);

if (pRspInfo->ErrorID != 0) {

// Member system handles login errors

printf("Failed to login, errorcode=%d errormsg=%s requestid=%d chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);

exit(-1);

}

// After successful login, the member system issues a new order request

CShfeFtdcInputOrderField ord;

memset(&ord, 0, sizeof(ord));

// Trading Member ID (TMF ID)

strcpy(ord.ParticipantID, g\_chParticipantID);

// Trading code (client level ID)

strcpy(ord.ClientID, "12345");

// Trading session ID (a.k.a seat ID)

strcpy(ord.UserID, g\_chUserID);

// Contract symbol

strcpy(ord.InstrumentID, "cn0601");

// order price type

ord.OrderPriceType = SHFE\_FTDC\_OPT\_LimitPrice;

// Buy-sell direction

ord.Direction = SHFE\_FTDC\_D\_Buy;

// Flag indicating whether to open or close position

strcpy(ord.CombOffsetFlag, "0");

// Flag indicating speculation orhedge

strcpy(ord.CombHedgeFlag, "1");

// Price

ord.LimitPrice = 50000;

// Quantity

ord.VolumeTotalOriginal = 10;

// order validity

ord.TimeCondition = SHFE\_FTDC\_TC\_GFD;

// GTD DATE

strcpy(ord.GTDDate, "");

// matched quantity type

ord.VolumeCondition = SHFE\_FTDC\_VC\_AV;

// The Minimum matched quantity

ord.MinVolume = 0;

// Trigger condition

ord.ContingentCondition = SHFE\_FTDC\_CC\_Immediately;

// trigger price (for stop order)

ord.StopPrice = 0;

// Reason ofenforced offset

ord.ForceCloseReason = SHFE\_FTDC\_FCC\_NotForceClose;

// Local order number

strcpy(ord.OrderLocalID, "0000000001");

// Flag indicating whether or notto suspend an order in the case of unexpected disconnection

ord.IsAutoSuspend = 0;

m\_pUserApi->ReqOrderInsert(&ord, 1);

}

// Response to order entry request

virtual void OnRspOrderInsert(CShfeFtdcInputOrderField \*pInputOrder, CShfeFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast)

{

// Output of order entry result

printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);

// Notify the completion of the order entry

SetEvent(g\_hEvent);

};

///callback for order acknowledgement

virtual void OnRtnOrder(CShfeFtdcOrderField \*pOrder)

{

printf("OnRtnOrder:\n");

printf("OrderSysID=[%s]\n", pOrder->OrderSysID);

}

// callback for notifying error

virtual void OnRspError(CShfeFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast) {

printf("OnRspError:\n");

printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);

printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

// member system is required to handle the error

exit(-1);

}

private:

// Points to the instance of CShfeFtdcMduserApi

CShfeFtdcTraderApi \*m\_pUserApi;

};

int main()

{

// file name of encryption key

const char \*pKeyFileName=”KeyFileName”;

// create an instance of CShfeFtdcTraderApi

CShfeFtdcTraderApi \*pUserApi = CShfeFtdcTraderApi::CreateFtdcTraderApi();

// create an instance of event handler

CSimpleHandler sh(pUserApi);

// Register the event handler

pUserApi->RegisterSpi(&sh);

// Subscribingto a private stream

// TERT\_RESTART:start from the current trading date

// TERT\_RESUME:resume from where it stopped last time

// TERT\_QUICK: start from the moment of login

pUserApi->SubscribePrivateTopic (TERT\_RESUME);

// Subscribing the public stream

// TERT\_RESTART:start from the current trading date

// TERT\_RESUME:resume from where it stopped last time

// TERT\_QUICK:start from the moment of login

pUserApi->SubscribePublicTopic(TERT\_RESUME);

//Set the heartbeat timout

pUserApi->SetHeartbeatTimeout(19);

// Set the encryption algorithm

pUserApi->RegisterCryptAlgorithm(pKeyFileName,NULL,SHFE\_FTDC\_ET\_RSA);

// Set the address of NameServer to acquire the Exchange gateways

pUserApi->RegisterNameServer("tcp://172.16.0.31:17001");

// setting encryption type

pUserApi->RegisterCryptAlgorithm(pKeyFileName,NULL,SHFE\_FTDC\_ET\_RSA);

// initiating connection to the Exchange system

pUserApi->Init();

// wait for the abortion of API instance

pUserApi->Join();

// Release API instance

pUserApi->Release();

return 0;

}

# MduserAPI Reference Manual

Part III covers developing market data acquisition.

Chapter 1 describes available interfaces and methods in **MduserAPI** from the perspective of system admin and business service; it also marks the interfaces that are not currently open to the public.

Chapter 2 is the **MduserAPI** reference manual.

Chapter 3 is a programming example for **MduserAPI**.

The error codes, enumeration values, and data types are listed in Part IV. Appendix.

# Categories of MduserAPI Interfaces

## Admin Interfaces

MduserAPI admin interfaces manage the life cycle and operation parameters of the API.

| **Interface Type** | **Interface name** | **Explanation** |
| --- | --- | --- |
| Lifecycle Management Interfaces | **CShfeFtdcMduserApi:: CreateFtdcMduserApi** | Create an **MduserApi** instance |
| **CShfeFtdcMduserApi::GetVersion** | Get API version |
| **CShfeFtdcMduserApi:: Release** | Delete the instance of the interface |
| **CShfeFtdcMduserApi:: Init** | Initialization |
| **CShfeFtdcMduserApi:: Join** | Wait for the interface thread to terminate |
| Parameter Management Interfaces | **CShfeFtdcMduserApi:: RegisterSpi** | Register callback |
| **CShfeFtdcMduserApi:: RegisterFront** | Register **Exchange gateway** network addresses |
| **CShfeFtdcTraderApi:: RegisterNameServer** | Register **Exchange NameServer** Network addresses |
| **CShfeFtdcTraderApi:: RegisterCryptAlgorithm** | Register the client encryption algorithm for certification |
| **CShfeFtdcMduserApi:: SetHeartbeatTimeout** | Set the heartbeat timeout |
| Subscription Interfaces | **CShfeFtdcMduserApi:: SubscribeMarketDataTopic** | Subscribe to market data of a specific topic |
| Communication Status Interfaces | **CShfeFtdcMduserSpi:: OnFrontConnected** | The method is invoked when connection to the Exchange is established but the client has not yet logged in. |
| **CshfeFtdcMduserSpi:: OnFrontDisconnected** | This method is invoked when communication to the Exchange is disconnected. |
| **CshfeFtdcMduserSpi:: OnHeartBeatWarning** | The method is invoked when a heartbeat is not received after a preset period of time. |

## Business Interfaces

| **Business Type** | **Business** | **Request Interface / Response Interface** | **Data Stream** |
| --- | --- | --- | --- |
| Login-Logout | Login | **CShfeFtdcMduserApi::ReqUserLogin**  **CShfeFtdcMduserSpi::OnRspUserLogin** | N/A |
| Logout | **CShfeFtdcMduserApi::ReqUserLogout**  **CShfeFtdcMduserSpi::OnRspUserLogout** | Dialog Stream |
| Subscription | Topic Subscription | **CShfeFtdcMduserApi::ReqSubscribeTopic**  **CShfeFtdcMduserSpi::OnRspSubscribeTopic** | Dialog Stream |
| Query Subscription | **CShfeFtdcMduserApi::ReqQryTopic**  **CShfeFtdcMduserSpi::OnRspQryTopic** | Query Stream |
| Market Data | Market Data Notification | **CShfeFtdcMduserSpi::OnRtnDepthMarketData** | Market Data Stream |
| Disaster Recovery | Data Cancellation | **CShfeFtdcMduserSpi::OnRtnFlowMessageCancel** | Market Data Stream |

# MduserAPI Reference Manual

## CShfeFtdcMduserSpi Interface

**CShfeFtdcMduserSpi** is an event notification interface. Users implement **CShfeFtdcMduserSpi** and the comprised methods to handle various events of interest.

### OnFrontConnected Method

This callback is invoked to handle a connection established between the client (market data acquisition) and the Exchange via TCP. The connection is automatically established by the API.

**Function Signature:**

**void OnFrontConnected()；**

Note: The invocation of this method signals that the TCP connection is established. The member system must log in before performing any operation. If the connection fails, this method will not be invoked.

### OnFrontDisconnected Method

This callback is invoked to handle the disconnection of a member system from the Exchange. After disconnection occurs, the API will attempt reconnection on behalf of the member system. The API will reconnect to the original gateway or, if the original is not available, other gateways supported by the system. The choice is automatically handled by the API.

**Function Signature:**

**void OnFrontDisconnected (int nReason)；**

**Parameter：**nReason: cause of disconnection

* 0x1001 network reading failure
* 0x1002 network writing failure
* 0x2001 heartbeat receipt timeout
* 0x2002 heartbeat sending timeout
* 0x2003 error message received

### OnHeartBeatWarning Method

This callback is invoked to handle warnings issued from the Exchange after a heartbeat is not received over the preset time period.

**Function Signature:**

**void OnHeartBeatWarning(int nTimeLapse)；**

**Parameter:** nTimeLapse：time elapsed from last heartbeat received

### OnPackageStart Method

This callback is invoked to handle the arrival of a data package received by the API. The package may contain multiple message blocks.

**Function Signature:**

**void OnPackageStart (int nTopicID, int nSequenceNo)；**

**Parameters:**

**nTopicID**：topic ID(e.g. private stream, public stream, market data stream)

**nSequenceNo**：sequence number of message

### OnPackageEnd Method

This callback is invoked to handle the completion of data processing from all blocks contained in a package.

**Function Signature:**

**void OnPackageEnd (int nTopicID, int nSequenceNo)；**

**Parameters:**

**nTopicID**： topic ID(e.g. private stream, public stream, market data stream)

**nSequenceNo**：sequence number of message

### OnRspUserLogin Method

This callback is invoked to handle responses from the Exchange regarding the member system’s login request. It states whether or not the login request was successfully executed.

**Function Signature:**

**void OnRspUserLogin(**

**CShfeFtdcRspUserLoginField \*pRspUserLogin,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspUserLogin**：points to the response from the Exchange system with respect to the client’s login request

struct CShfeFtdcRspUserLoginField {

///Trade date

TShfeFtdcDateType TradingDay;

/// login time

TShfeFtdcTimeType LoginTime;

///Maximum local ID, **NOT USED**

TShfeFtdcOrderLocalIDType MaxOrderLocalID;

///Trader ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Trading System Name

TShfeFtdcTradingSystemNameType TradingSystemName;

///Data Center ID

TShfeFtdcDataCenterIDType DataCenterID;

///Current length of member’s private flow

TShfeFtdcSequenceNoType PrivateFlowSize;

/// Current size of client’s private flow

TShfeFtdcSequenceNoType UserFlowSize;

/// Transaction date

TShfeFtdcDateType ActionDay;

};

Note：ActionDay is a newly added field. ActionDay reflects the transaction date while TradingDay reflects the trade date. ActionDay is set to empty until it is open to the public.

**pRspInfo**: points to the response to the client’s request. **Please note**: **if there are multiple successful responses, it may be null, but this will not be the case for the first response.** Error code0 means a successful transaction. The structure is shown below:

struct CShfeFtdcRspInfoField {

/// Error code

TShfeFtdcErrorIDType ErrorID;

/// Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID designated by the client upon login.

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRspUserLogout Method

This callback is invoked to handle responses from the Exchange regarding the client’s logout request. The response states whether or not the logout was successfully executed.

**Function Signature:**

**void OnRspUserLogout(**

**CShfeFtdcRspUserLogoutField \*pRspUserLogout,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pRspUserLogout**：points to the user logout request.

The structure of the logout request is shown below:

struct CShfeFtdcRspUserLogoutField {

///Trader ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

};

**pRspInfo**: points to the response from the Exchange with respect to the request, with the structure below:

struct CShfeFtdcRspInfoField {

/// Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

**nRequestID**：requests the ID maintained by the member system in the client’s logout request

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRspSubscribeTopic Method

This callback is invoked after the Exchange system responds to the member system’s subscription request.

**Function Signature:**

**void OnRspSubscribeTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pDissemination**：points to the subscription request, including the subscribed topic and the starting sequence number, with the structure below:

struct CShfeFtdcDisseminationField {

///Sequence series

TShfeFtdcSequenceSeriesType SequenceSeries;

///Sequence number

TShfeFtdcSequenceNoType SequenceNo;

};

**pRspInfo**：points to the status with respect to the transaction, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

The possible errors:

Error code Error message Possible reasons

66 User hasn't logged in yet Not logged in yet

**nRequestID**：requests the IDspecified by the member system in its subscription request

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID

### OnRspQryTopic Method

This callback is invoked to handle results returned from the Exchange regarding the member system’s query for a topic.

**Function Signature:**

**void OnRspQryTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

**Parameters:**

**pDissemination**：points to the client’s query request, including queried topic and number of messages returned with respect to the topic, with the structure below:

struct CShfeFtdcDisseminationField {

///Sequence series

TShfeFtdcSequenceSeriesType SequenceSeries;

///Sequence number

TShfeFtdcSequenceNoType SequenceNo;

};

**pRspInfo**：points to the transaction status regarding the request, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error message

TShfeFtdcErrorMsgType ErrorMsg;

};

The possible errors:

Error code Error message Possible reasons

66 User hasn't logged in yet Not logged in yet

**nRequestID**：requests the ID specified by the member system in its query request.

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID.

### OnRspError Method

This callback is invoked when an error occurs with respect tothe client’s request.

**Function Signature:**

**void OnRspError(**

**CShfeFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast);**

**Parameters:**

**pRspInfo**：points to the response returned from the trading system, with the structure below:

struct CShfeFtdcRspInfoField {

///Error code

TShfeFtdcErrorIDType ErrorID;

///Error Message

TShfeFtdcErrorMsgType ErrorMsg;

};

Possible Errors：

**Error Code Error Msg Possible Causes**

1 Not Login Not Login yet

Too High FTD Version Too High FTD Version

Invalid interface product info Invalid interface product info

Unrecognized ftd tid Invalid FTD package head

997 api authentication failure Illegal api usage

**nRequestID**：request ID specified by the member system in its request.

**bIsLast**：indicates whether the current return is the latest with respect to the nRequestID.

### OnRtnDepthMarketData Method

This callback is invoked after the Exchange system sends member systemmarket data to the member system reflecting the changes on the book.

**Function Signature:**

**void OnRtnDepthMarketData (CShfeFtdcDepthMarketDataField \*pTrade)；**

**Parameter:**

**pDepthMarketData**：points to the returned market data object. Note: some fields are not used. The market data structure is shown below:

struct CShfeFtdcDepthMarketDataField

{

///Business day

TShfeFtdcDateType TradingDay;

///Settlement group's code

TShfeFtdcSettlementGroupIDType SettlementGroupID;

///Settlement number

TShfeFtdcSettlementIDType SettlementID;

/// Latest price

TShfeFtdcPriceType LastPrice;

/// Yesterday's settlemnt

TShfeFtdcPriceType PreSettlementPrice;

///Yesterday's close

TShfeFtdcPriceType PreClosePrice;

///Yesterday's open interest

TShfeFtdcLargeVolumeType PreOpenInterest;

///Today's open

TShfeFtdcPriceType OpenPrice;

///The highest price

TShfeFtdcPriceType HighestPrice;

///The lowest price

TShfeFtdcPriceType LowestPrice;

///Quantity

TShfeFtdcVolumeType Volume;

///Turnover

TShfeFtdcMoneyType Turnover;

///Open interest

TShfeFtdcLargeVolumeType OpenInterest;

///Today's close

TShfeFtdcPriceType ClosePrice;

///Today's settlement

TShfeFtdcPriceType SettlementPrice;

///The upward price limit

TShfeFtdcPriceType UpperLimitPrice;

///The downward price limit

TShfeFtdcPriceType LowerLimitPrice;

///Yesterday's Delta value, not used

TShfeFtdcRatioType PreDelta;

///Today's Delta value，not used

TShfeFtdcRatioType CurrDelta;

///Last modification time

TShfeFtdcTimeType UpdateTime;

/// The last modified millisecond

TShfeFtdcMillisecType UpdateMillisec;

///Contract code

TShfeFtdcInstrumentIDType InstrumentID;

///Bid price 1

TShfeFtdcPriceType BidPrice1;

/// Bid volume 1

TShfeFtdcVolumeType BidVolume1;

///Ask price 1

TShfeFtdcPriceType AskPrice1;

///Ask volume 1

TShfeFtdcVolumeType AskVolume1;

/// Bid price 2

TShfeFtdcPriceType BidPrice2;

/// Bid volume 2

TShfeFtdcVolumeType BidVolume2;

/// Ask price 2

TShfeFtdcPriceType AskPrice2;

/// Ask volume 2

TShfeFtdcVolumeType AskVolume2;

///Bid price 3

TShfeFtdcPriceType BidPrice3;

///Bid volume 3

TShfeFtdcVolumeType BidVolume3;

/// Ask price 3

TShfeFtdcPriceType AskPrice3;

///Ask volume 3

TShfeFtdcVolumeType AskVolume3;

///Bid price 4

TShfeFtdcPriceType BidPrice4;

///Bid volume 4

TShfeFtdcVolumeType BidVolume4;

///Ask price 4

TShfeFtdcPriceType AskPrice4;

///Ask volume 4

TShfeFtdcVolumeType AskVolume4;

///Bid price 5

TShfeFtdcPriceType BidPrice5;

///Bid volume 5

TShfeFtdcVolumeType BidVolume5;

/// Ask price 5

TShfeFtdcPriceType AskPrice5;

///Ask price 5

TShfeFtdcVolumeType AskVolume5;

///

TShfeFtdcDateType ActionDay;

};

Note：ActionDay is a new field. ActionDay is used for business occurrence day, and TradingDay is used when the trading day is required. When ActionDay is not supported by the Exchange, it should be left as empty.

### OnRtnFlowMessageCancel method

When the Exchange system switches from the primary data center to the backup data center, it may need to cancel some messages as result of data loss or inconsistency. This callback is invoked to handle the cancelled messages sent by the Exchange system after the member system logs back in and re-subscribes to a stream (private or public).

**Function Signature:**

**void OnRtnFlowMessageCancel(**

**CShfeFtdcFlowMessageCancelField \*pFlowMessageCancel)；**

**Parameter:**

**pFlowMessageCancel:** points to a stream with rolled back messages, with the structure below:

struct CShfeFtdcFlowMessageCancelField

{

///Sequence Series

TShfeFtdcSequenceSeriesType SequenceSeries;

///TradeDate

TShfeFtdcDateType TradingDay;

///Data Center ID

TShfeFtdcDataCenterIDType DataCenterID;

///Starting Sequence number

TShfeFtdcSequenceNoType StartSequenceNo;

///Ending Sequence number

TShfeFtdcSequenceNoType EndSequenceNo;

};

SequenceSeries：the data stream containing cancelled transactions, either private or public

The cancelled messages are in this range: (StartSequenceNo, EndSequenceNo)

## CShfeFtdcMduserApi Interfaces

The functions included in the **CShfeFtdcMduserApi** interface are login/logout, market data subscription, etc.

### CreateFtdcMduserApi Method

This method is invoked to create an instance of the **CShfeFtdcMduserApi.** The instance cannot be instantiated by a “new” function.

**Function Signature:**

**static CShfeFtdcMduserApi \*CreateFtdcMduserApi(const char \*pszFlowPath = "");**

**Parameter:**

**pszFlowPath**：the constant character points to a file directory that stores the status of the advisory published by the Exchange System. The default value is the current directory.

**Return Value：**points to the created instance of the **CShfeFtdcMduserApi**.

### GetVersion Method

This method returns the API version.

**Function Signature:**

**const char \*GetVersion(int &nMajorVersion, int &nMinorVersion) ;**

**Parameters:**

nMajorVersion：returns the main/primary version number

nMinorVersion：returns the minor/secondary version number

**Returned Value：**

The return is a constant string containing the value of the API version.

### Release Method

This method is invoked to release an instance of **CShfeFtdcMduserApi.** This function cannot be performed via the “delete” method.

**Function Signature:**

**void Release()；**

### Init Method

This method is invoked to establish a connection from the client (market data acquisition) system to the Exchange system. Users can log in after successful connection.

**Function Signature:**

**void Init()；**

### Join Method

This method is invoked to await the completion of a thread implemented from an interface.

**Function Signature:**

**void Join()；**

### GetTradingDay Method

This method is invoked to get the current trade date. It is retrievable after successful login.

**Function Signature:**

**const char \*GetTradingDay()；**

**Return Value:**

This returns a constant pointer the string containing the trade date.

### RegisterSpi Method

This method is invoked to register an instance implemented from **CShfeFtdcMduserSpi**. The instance handles events.

**Function Signature:**

**void RegisterSpi(CShfeFtdcMduserSpi \*pSpi) ;**

**Parameter:**

pSpi: points to an instance of implementing **ShfeFtdcTraderSpi**.

### RegisterFront Method

This method is invoked to set the network address of the Exchange market data gateway. The Exchange system usually provides multiple market data gateways. The member system can register more than one.

This method must be invoked before the **Init** method.

**Function Signature:**

**void RegisterFront(char \*pszFrontAddress);**

**Parameter:**

**pszFrontAddress**：points tothe Exchange’s market data gateway. The network address is in the format of “**protocol://ipaddress:port**”, e.g. “tcp://127.0.0.1:17001”. “tcp” in the example is the transmission protocol, “127.0.0.1” represents the server address, and “17001” represents the port number

### RegisterNameServer Method

This method is invoked to set the network address of the Exchange's NameServer, and provides a list of market data gateways. The Exchange system usually has multiple NameServers, and the member system can register with more than one.

This method must be invoked before the Init method.

**Function Signature：**

**void RegisterNameServer (char \*pszNsAddress);**

**Parameter：**

**pszNsAddress**：Points to the network address of the Exchange's NameServer. The network address is in the format of “protocol://ipaddress:port” such as “tcp://127.0.0.1:17001”. “tcp” represents the transmission protocol, “127.0.0.1” represents the server's address, and “17001” represents the port number.

### RegisterCryptAlgorithm

This method is invoked to register the encryption algorithm used by the client’s software vendor. It is required to certify the client’s trading software. The encryption key is supplied by the Exchange.

This method must be invoked prior to Init().

**Function Signature：**

**int RegisterCryptAlgorithm(const char\* pszKeyFileName, const char \*pszKeyFilePassword,**

**TshfeFtdcEncryptTypeType nEncryptType = SHFE\_FTDC\_ET\_RSA);**

**Parameter：**

**pszKeyFileName：file name of client’s software vendor’s encryption key**

**pszKeyFilePassword：password to the encryption key. If the client’s software vendor does not set the password, this value is set to NULL;**

**nEncryptType：encryption type**

* SHFE\_FTDC\_ET\_RSA：RSA

**Return Value：**

* 0: success
* -1: failure to read the key file
* -2: failure to parse the key file
* -3: the encryption algorithm is not supported

### RegisterCertificateFile Method

This method is invoked to register the client’s certificate file.

This method is effective after the Exchange announces an SSL connection, and should be invoked before the Init() method.

**Function Signature：**

**int RegisterCertificateFile(const char\* pszCertFileName,const char \*pszKeyFileName,const char \*pszCaFileName, const char \*pszKeyFilePassword);**

**Paramters：**

**pszCertFileName**：client’s certificate file name

**pszKeyFileName**：client’s private key file name

**pszCaFileName**：trusted CA certificate file name

**pszKeyFilePassword**：password to the client’s private key file

**Return value:**

* 0:success
* -1: failure to load the trusted CA certificate
* -2: failure to load the client’s certificate
* -3: failure to load the client’s private key
* -4: failure to verify the client’s certificate

### SetHeartbeatTimeout Method

This method is invoked to set a heartbeat timeout. After the member system connects to the Exchange system, the API will send a heartbeat at regular intervals, which is used to detect an abnormal connection. **The heartbeat timeout recommended by the Exchange is a value between 10second and 30seconds.**

**Function Signature:**

**virtual void SetHeartbeatTimeout(unsigned int timeout);**

**Parameter:**

**Timeout:** heartbeat timeout (in seconds). If a message is not received from the Exchange system after “timeout/2” seconds, **CShfeFtdcMduserApi::OnHeartBeatWarning()**will be triggered. If a message is not received from the Exchange system after “timeout” seconds, the connection will be aborted and **CShfeFtdcMduserApi ::OnFrontDisconnected()**will be triggered.

### SubscribeMarketDataTopic Method

This method is invoked to subscribe to market data requested by the client. After the subscription, the Exchange system will stream the market data to the member system.

**Function Signature:**

**void SubscribeMarketDataTopic(int nTopicID, TE\_RESUME\_TYPE nResumeType);**

**Parameters:**

**nTopicID：**market data topic. Available topics are announced by the Exchange.

**nResumeType：**market data replay methods:

* TERT\_RESTART：transmits from the start of the current trade date
* TERT\_RESUME：resumes from where it was stopped last time
* TERT\_QUICK： sends out a snapshot of the current market data, followed by subsequent data changes. **The Exchange recommends this method for quick market data recovery.**

### ReqUserLogin Method

This method is invoked by the member system to log in.

**Function Signature:**

**int ReqUserLogin(**

**CShfeFtdcReqUserLoginField \*pReqUserLoginField,**

**int nRequestID)；**

**Parameters:**

**pReqUserLoginField**：points to the login request, with the structure below:

struct CShfeFtdcReqUserLoginField {

///Trade Day

TShfeFtdcDateType TradingDay;

///Trader ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

///Password

TShfeFtdcPasswordType Password;

///User-end product information

TShfeFtdcProductInfoType UserProductInfo;

///Interface-port product information, **NOT USED**

TShfeFtdcProductInfoType InterfaceProductInfo;

///Protocol information, **NOT USED**

TShfeFtdcProtocolInfoType ProtocolInfo;

///Data Center ID

TShfeFtdcDataCenterIDType DataCenterID;

};

**The member system must input a value in the “UserProductInfo” field, indicating a market data acquisition system (such as ISV) and a version number. For example,“SFIT Mduser V100” represents the market data acquisition system developed by SFIT and the version number.**

**For a first-time login, the member system can input 0 or the primary data center ID published by the Exchange into the“DataCenterID” field. Subsequent logins must input the data center ID returned from the previous login request.**

**nRequestID**：requests the ID specified by the client in its login request.

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds permitted maximum
* -3: the number of requests per second exceeds the permitted maximum

### ReqUserLogout Method

This method is invoked by the member system to request logout.

**Function Signature:**

**int ReqUserLogout(**

**CShfeFtdcReqUserLogoutField \*pReqUserLogout,**

**int nRequestID)；**

**Parameters:**

**pReqUserLogout**：points to the logout request, with the structure below:

struct CShfeFtdcReqUserLogoutField {

///Trading User ID

TShfeFtdcUserIDType UserID;

///Member ID

TShfeFtdcParticipantIDType ParticipantID;

};

**nRequestID**：requests the ID specified by the client in its logout request

**Returned Value:**

* 0: success
* -1: network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqSubscribeTopic Method

This method is invoked to request a subscription to atopic, and should be invoked after login.

**Function Signature:**

**int ReqSubscribeTopic (**

**CShfeFtdcDisseminationField \*pDissemination,**

**int nRequestID)；**

**Parameters:**

**pDissemination**：points to the subscription object including the topic (subject) and starting sequence number, with the structure below:

struct CshfeFtdcDisseminationField {

///Sequence series number

TshfeFtdcSequenceSeriesType SequenceSeries;

///Sequence number

TshfeFtdcSequenceNoType SequenceNo;

};

**SequenceSeries:** topics to be subscribed

**SequenceNo:**=-1 representing subscribing in “QUICK” method; the other values indicating the starting sequence number.

**nRequestID**：requests the ID specified by the member system in its subscription request

**Returned Value:**

* 0: success
* -1:network connection failure
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

### ReqQryTopic Method

This method is invoked to request the querying of a topic, and should be invoked after login.

**Function Signature:**

**int ReqQryTopic (**

**CshfeFtdcDisseminationField \*pDissemination,**

**int nRequestID)；**

**Parameters:**

**pDissemination**: points to the query object including the topic to be queried, with the structure below:

struct CShfeFtdcDisseminationField {

///Sequence Series

TShfeFtdcSequenceSeriesType SequenceSeries;

///Sequence Number

TShfeFtdcSequenceNoType SequenceNo;

};

SequenceSeries：ttopics to be queried

SequenceNo：no need to fill in

**nRequestID**：requests the ID specified by the member system in its query request

**Returned Value:**

* 0: success.
* -1: network connection failure;
* -2: the number of unprocessed requests exceeds the permitted maximum
* -3: the number of requests sent per second exceeds the permitted maximum

# MduserAPI—A Development Example

// mdusertest.cpp :

// A simple example to illustrate the usage of**CShfeFtdcMduserApi**&**CShfeFtdcMduserSpi**interfaces

//DBL\_MAX（1.7976931348623157e+308）

#include <stdio.h>

#include <string.h>

#include "FtdcMduserApi.h"

class CSimpleHandler : public CShfeFtdcMduserSpi

{

public:

//constructor requires a pointer to an instance of **CShfeFtdcMduserApi**

CSimpleHandler(CShfeFtdcMduserApi \*pUserApi) : m\_pUserApi(pUserApi) {}

~CSimpleHandler() {}

//this callback is invoked after the member system connects to the Exchange

void OnFrontConnected() {

CShfeFtdcReqUserLoginField reqUserLogin;

strcpy(reqUserLogin.ParticipantID, "P001"); //trading member ID (4 digits)

strcpy(reqUserLogin.UserID, "U001"); //trading session ID (a.k.a seat ID)

strcpy(reqUserLogin.Password, "P001"); //trading session psw (a.k.a seat psw)

m\_pUserApi->ReqUserLogin(&reqUserLogin, 0);

}

//this callback is invoked when the member system disconnects from the Exchange

void OnFrontDisconnected() {

// When disconnection occurs, the API will automatically reconnect. The member system does not need to handle reconnection.

printf("OnFrontDisconnected.\n");

}

// This method is invoked to receive a response from the Exchange system as to whether or not the login request was successfully executed.

void OnRspUserLogin(CShfeFtdcRspUserLoginField \*pRspUserLogin, CShfeFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast) {

printf("OnRspUserLogin: ErrorCode=[%d], ErrorMsg=[%s]\n",

pRspInfo->ErrorID, pRspInfo->ErrorMsg);

printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

if (pRspInfo->ErrorID != 0) {

//login error handling

printf("Failed to login, errorcode=%d errormsg=%s requestid=%d chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);

}

}

//this callback is invoked after the Exchange system sends market data

void OnRtnDepthMarketData(CShfeFtdcDepthMarketDataField \*pMarketData) {

//member system (MD acquisition system) process the returned data

if(pMarketData->OpenPrice!=DBL\_MAX)

{

printf(“OpenPrice=%.2f\n”,pMarketData->OpenPrice);

}

}

//this callback is invoked if a user’s request fails

void OnRspError(CShfeFtdcRspInfoField \*pRspInfo, int nRequestID, bool bIsLast) {

printf("OnRspError:\n");

printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);

printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

//Themember system handles error

}

private:

// Points to an instance of **CShfeFtdcMduserApi**

CShfeFtdcMduserApi \*m\_pUserApi;

};

int main()

{

// file name of encryption key

const char \*pKeyFileName=”KeyFileName”;

// Creates an instance of CShfeFtdcTraderApi

CShfeFtdcMduserApi \*pUserApi = CShfeFtdcMduserApi::CreateFtdcMduserApi();

//Creates an instance of an event handler

CsimpleHandler sh(pUserApi);

//Registersthe event handler

pUserApi->RegisterSpi(&sh);

// Registersthe desired market data subject

///TERT\_RESTART: transmits from the start of the current trade date

/// TERT\_RESUME: resumesfrom where it stops

///TERT\_QUICK: sendsa snapshotof the current market then the subsequent data

pUserApi-> SubscribeMarketDataTopic (102, TERT\_RESUME);

//Sets the timeout for heartbeat

pUserApi->SetHeartbeatTimeout(19);

// Sets the address of the nameserver forth market data gateway

pUserApi->RegisterNameServer("tcp://192.168.1.1:17011");

// Sets the encryption algorithm type

pUserApi->RegisterCryptAlgorithm(pKeyFileName,NULL,SHFE\_FTDC\_ET\_RSA);

//Starts connecting to the market data gateway

pUserApi->Init();

// Releases the MduserAPI instance

pUserApi->Release();

return 0;

# Appendix

# Error Codes

| **Error No.** | **Error message** | **Cause of error** |
| --- | --- | --- |
| 1 | Incorrect dialogue | Illegal dialogue was found in an operation |
| 2 | Missing contract | Missing contract in new order, quote, OTC order, or exercise |
| 3 | Missing Member ID | Missing trading member ID |
| 4 | Missing Client ID | Missing client ID |
| 6 | Illegal values in new order | Illegal value found in new order (prohibited enumeration value) |
|  |  | Enforced-Offset reason was filled for non-enforced-offset order |
| 7 | Illegal values in new quote | Illegal values found in new quote (prohibited enumeration value) |
| 8 | Incorrect value in order operation | Illegal value found in the order operations (prohibited enumeration value) |
| 9 | Incorrect value in quote operation | Illegal value found in the quote operations (prohibited enumeration value) |
| 12 | Duplicate order | Local order number in the new order is duplicated |
| 13 | Duplicate quote | Local quote number in the new quote is duplicated |
| 15 | Client does not have an account with the designated member | The client does not have an account with the designated trading member |
| 16 | IOC order to be added in non-continuous trading mode. | Attempt to add an IOC order in non-continuous trading mode |
| 17 | GFA to be added in auction mode | Attempt to add a GFA order during auction |
| 18 | IOC not set for market order | Attempt to add a market order without turning on IOC |
| 19 | Quantity restriction must be user for order with IOC | Attempt to add a new order entry with a quantity restriction, but the time condition is not set to IOC |
| 20 | GTD order with past date | Attempt to add a GTD order with a expiration date that has passed |
| 21 | The minimum trade quantity exceeds the quantity of order | Attempt to add a new order with minimum match quantity larger than the total quantity of the order |
| 22 | Data not fully synchronized in trading system | Data was not fully synchronized in the trading system. |
| 23 | Data not fully synchronized in clearing system | Data was not fully synchronized in the clearing system |
| 24 | Order cannot be found | Order cannot be found when the specific operation is requested to this order |
| 25 | Quote cannot be found | Quote cannot be found when the specific operation is requested to this quote |
| 26 | Operation not permitted in the current trading status | Attempt to add a new order when the status is neither trading nor auction |
|  |  | Attempt to activate an order when the status is neither trading nor auction;  Non-admin user attempts to conduct an operation to an order when the status is neither trading nor auction;  Attempt to cancel or suspend an order when trading status is closed |
|  |  | Attempt to add a new OTC order outside of the continuous trading mode |
| 27 | Illegal status for transitioning instrument trading status | Attempt to change instrument status in a prohibited trading mode |
| 28 | Attempt to operate on an order after it is fully filled | Attempt to operate on an order after it is fully filled |
| 29 | Attempt to operate on an order after it is cancelled | Attempt to operate on an order after it is cancelled |
| 31 | The client's position is insufficient at the time of offsetting | When operating on an order with an action that will result in offsetting the position, there is insufficient position remaining to be offset |
| 32 | Exceeds client's position limit | Attempt to operate on an order with an action that will exceed the client’s position limit |
| 34 | Exceeds member's position limit | Attempt to operate on an order with an action that will exceed the member’s position limit |
| 35 | Missing account | Account cannot be found during various operations |
| 36 | Insufficient capital | Insufficient capital in the account |
| 37 | Illegal quantity | Quantity is not integral multiples of the minimum quantity, or exceeds the maximum quantity when adding or operating an order or OTC order |
| 45 | Settlement group not in valid initialization status | Settlement group not in valid initialization status |
| 48 | The price is not an integral multiple of the tick | The price is not an integral multiple of the tick |
| 49 | Price exceeds the upper limit | Price exceeds the upper limit |
| 50 | Price falls below the lower limit | Price falls below the lower limit |
| 51 | Not authorized to trade | A trading member or a client is not authorized to trade the designated contract, or a trader is not authorized to trade. |
| 52 | User can close position only | When a member or client or trader attempts an operation that may increase a position, the user has the authority to close the position only to the designated contract |
| 53 | Not an authorized role | While adding a new order entry, it is found that the member does not have authorization to perform the action requested by the client |
| 57 | Not permitted to act for another Member | User attempts to perform an action for a member s/he does not belong to |
| 58 | Unmatched user | User attempts to perform an action that does not match the one in the dialog |
| 59 | Duplicate login | User attempts a duplicate login |
| 60 | Invalid username or password | Username or password is found when user attempts to log in or change password |
| 62 | Inactive user | User is found to be inactive when attempting to log in |
| 64 | User does not match member | User is found not to belong to the claimed member during his/her attempt to log in |
| 65 | Incorrect IP | User is found to log in from invalid IP address during his/her attempt to log in |
| 67 | Logout user does not match login user | When a user is logging out, it is found that s/he is not the same one who logged in |
| 66 | User not logged in yet | User is not yet logged in to perform his/her attempted operations |
| 68 | Logout user does not match login member | When a user attempts to log out or is forced out, or is attempting to change his/her password, the associating member does not match the one used at login |
| 70 | Quote has been cancelled | Attempt to act on a quote that has been cancelled |
| 76 | Order has been suspended | Attempt to act on an order that has been suspended |
| 77 | Order has been activated | Attempt to activate an order that has already been activated |
| 78 | Missing expiration date in GTD date | Missing expiration date when attempt to add GTD date |
| 79 | Unsupported order type | Attempt to add a new order of a type not yet supported by the Exchange system |
| 80 | Unauthorized operation | Attempt to perform an action that is authorized for an admin user only |
| 83 | Stop order not permitted to be added in non-trading mode | Attempt to add a stop order when the trading statuses not continuous trading |
| 84 | Missing IOC or GFD in stop order | Time condition is neither IOC nor GFD when attempting to add a stop order |
| 89 | Invalid value in exercise | Attempt to add an exercise containing an invalid value (prohibited enumeration value) |
| 90 | Incorrect value found in exercise | Invalid value found when attempting to exercise(prohibited enumeration value) |
| 91 | Duplicate exercise | Local exercise number is duplicated during an attempt to add a new exercise |
| 92 | Attempt to execute an exercise that has been cancelled | Attempt to execute an exercise that has been cancelled |
| 93 | Exercise cannot be found | Attempt to execute an exercise that cannot be found |
| 94 | Request exercise on non-option contract | Attempt to request a new exercise on non-option contract |
| 95 | Missing trigger price | The stop order is missing a trigger price |
| 96 | Insufficient hedge quota | It is found that client's hedge quota is insufficient during an operation that will increase position |
| 97 | Duplicate operation | It is found that the local operation number is duplicate when performing operation on order, quote or exercise |
| 99 | Perform operation for another user | User attempt to add an order for another user |
| 100 | Invalid user type | Trader attempts to log in with market data user account |
| 103 | Attempt to close hedge position on current trade date | Attempt to close hedge position on current trade date |
| 104 | Unknown administration command | Unknown administration command |
| 114 | Missing IOC in best priced order | Attempt to add a best priced order that is missing an IOC flag |
| 121 | Invalid value found in exercise abandonment request | Invalid value found in exercise abandonment request |
| 122 | Invalid value found when executing exercise abandonment | Invalid value found when executing exercise abandonment |
| 123 | Duplicate exercise abandonment | Local number of exercise abandonment is duplicated |
| 124 | Exercise has been abandoned | Attempt to operate on an exercise that has been abandoned |
| 125 | Exercise abandonment not found | Attempt to operate on an exercise abandonment that cannot be found |
| 126 | Exercise abandonment is for option only | Attempt to request an exercise abandonment on a non-option contract |
| 127 | Outside of exercising period | Attempt to add an exercise or exercise abandonment in non-exercisable period |
| 128 | Exercise abandonment is permitted to long position holders only | Option seller does not allow an exercise to be abandoned |
| 129 | OffsetFlag cannot be set to ‘open position’ in exercise or exercise abandonment | When requesting or abandoning an exercise, the offset Flag cannot be set to false, which means an open position |
| 130 | Insufficient positions to be reserved | When an option seller requests to keep the future position after an exercise, the requested quantity exceeds the quantity of his/her total positions |
| 131 | Exceeds the daily limit of open positions permitted for a specific contract under the client | The client attempts to open more positions on a contract than his/her daily limit |
| 132 | Exceeds TPS threshold | The client attempts to send more transactions on a product in a second than the TPS threshold allows |
| 133 | Exceeds max cancellations | The client attempts to cancel more transactions on a product in a second than the threshold allows |
| 134 | API verification failure | The client’s trading software uses unofficial API libraries |
| 135 | Certification failure | The client’s trading software has not passed certification |
| 136 | Not entitled to connect directly to the Exchange gateway | The client attempts to connect to the Exchange gateway directly instead of via FENS, from which the IP of gateway is allocated |

# Enumeration Value List—Translated

| **Auto** | **Enumeration Description** | **Enumeration Prefix** | **Enumeration Name** | **Code Description** | **Code Name** | **Numerical Value of Code** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | Trading role | ER | TradingRole | Broker | Broker | 1 |
| Proprietary trading | Host | 2 |
| Market maker | MarketMaker | 3 |
| 2 | User type | UT | UserType | Trader | Trader | 1 |
| Trade manager | TradeManager | 2 |
| Market data user | MDUser | 3 |
| Unauthorized trader | SingleTrader | 4 |
| 3 | Product type | PC | ProductClass | Future | Futures | 1 |
| Option | Options | 2 |
| Spread | Combination | 3 |
| Spot | Spot | 4 |
| EFP | EFP | 5 |
| 4 | Option type | OT | OptionsType | Non-option | NotOptions | 0 |
| Bullish (call) | CallOptions | 1 |
| Bearish (put) | PutOptions | 2 |
| 5 | Instrument status | IS | InstrumentStatus | Pre-opening | BeforeTrading | 0 |
| Non-trading | NoTrading | 1 |
| Continuous trade | Continuous | 2 |
| Auction order | AuctionOrdering | 3 |
| Auction balancing | AuctionBalance | 4 |
| Auction Matching | AuctionMatch | 5 |
| Closed | Closed | 6 |
| 6 | Buy-sell direction | D | Direction | Buy | Buy | 0 |
| Sell | Sell | 1 |
| 7 | Type of position | PT | PositionType | Net position | Net | 1 |
| Gross position | Gross | 2 |
| 8 | Direction position | PD | PosiDirection | Net | Net | 1 |
| Long | Long | 2 |
| Short | Short | 3 |
| 9 | Synchronization status of the Exchange's data | EDS | ExchangeDataSyncStatus | Unsynchronized | Asynchronous | 1 |
| Synchronizing | Synchronizing | 2 |
| Synchronized | Synchronized | 3 |
| 10 | Synchroniza-tion status of settlement group's data | SGDS | SGDataSyncStatus | Unsynchronized | Asynchronous | 1 |
| Synchronizing | Synchronizing | 2 |
| Synchronized | Synchronized | 3 |
| 11 | Flag of speculation or hedge | HF | HedgeFlag | Speculation | Speculation | 1 |
| Hedge | Hedge | 3 |
| 12 | Type of client | CT | ClientType | Person | Person | 0 |
| Institution | Company | 1 |
| Investment fund | Fund | 2 |
| 13 | Cause of instrument status change | IER | InstStatusEnterReason | Automatic switch | Automatic | 1 |
| Manual switch | Manual | 2 |
| Automatic Circuit breaking | Automatic circuit breaking | 3 |
| Manual circuit breaking | Manual circuit breaking | 4 |
| 14 | Order price type | OPT | OrderPriceType | Random price | AnyPrice | 1 |
| Limited Price | LimitPrice | 2 |
| Best price | BestPrice | 3 |
| 15 | Offset type | OF | OffsetFlag | Open Position | Open | 0 |
| Close position | Close | 1 |
| Enforced close | ForceClose | 2 |
| Close current trade date position | CloseToday | 3 |
| Closing prior trade date position | CloseYesterday | 4 |
| 16 | Cause of enforced closing | FCC | ForceCloseReason | Non-forced closing | NotForceClose | 0 |
| Insufficient deposit | LackDeposit | 1 |
| Client exceeding position limit | ClientOverPositionLimit | 2 |
| Member exceeding position limit | MemberOverPositionLimit | 3 |
| Position is not in integral multiple | NotMultiple | 4 |
| Market violation found | Violation | 5 |
| Others | Other | 6 |
| Person client approach to the delivery | PersonDeliv | 7 |
| Hedge exceeding position limit | HedgeOverPositionLimit | 8 |
| 17 | Order status | OST | OrderStatus | Fully filled | AllTraded | 0 |
| Partly filled and on book | PartTradedQueueing | 1 |
| Partly filled and off book | PartTradedNotQueueing | 2 |
| Not matched and on book | NoTradeQueueing | 3 |
| Not matched and off book | NoTradeNotQueueing | 4 |
| Cancelled | Cancelled | 5 |
| 18 | Order Type | ORDT | OrderType | Normal | Normal | 0 |
| Derived from quote | DeriveFromQuote | 1 |
| Derived from spread | DeriveFromCombination | 2 |
| 19 | OTC order status | OOS | OTCOrderStatus | Submit by a party | Inputed | 0 |
| Confirmed by a party | Confirmed | 1 |
| Cancelled | Cancelled | 2 |
| Rejected | Refused | 3 |
| 20 | Time Condition | TC | TimeCondition | Immediately performed or cancelled | IOC | 1 |
| Good for this session | GFS | 2 |
| Good for the day | GFD | 3 |
| Good till date | GTD | 4 |
| Good till cancelled | GTC | 5 |
| Good for auction | GFA | 6 |
| 21 | Match quantity condition | VC | VolumeCondition | Random | AV | 1 |
| Minimum quantity | MV | 2 |
| Full quantity | CV | 3 |
| 22 | Trigger conditions | CC | ContingentCondition | Immediately | Immediately | 1 |
| Trigger | Touch | 2 |
| 23 | Order operation type | AF | ActionFlag | Deletion | Delete | 0 |
| Suspension | Suspend | 1 |
| Activation | Active | 2 |
| Modification | Modify | 3 |
| 24 | Origin of order | OSRC | OrderSource | From participant | Participant | 0 |
| From administrator | Administrator | 1 |
| 25 | Trade type | TRDT | TradeType | From order matching | Common | 0 |
| From option exercise | OptionsExecution | 1 |
| From OTC matching | OTC | 2 |
| From the Exchange For Physical | EFPDerived | 3 |
| From Spread Implied | CombinationDerived | 4 |
| 26 | Source of trade price | PSRC | PriceSource | Last trade Price | LastPrice | 0 |
| Bid price | Buy | 1 |
| Ask price | Sell | 2 |
| 27 | Acccount status | ACCS | AccountStatus | Enabled | Enable | 0 |
| Disabled | Disable | 1 |
| 28 | Member type | MT | MemberType | Trading member | Trading | 0 |
| Settlement member | Settlement | 1 |
| Compositive member | Compositive | 2 |
| 29 | Execution result | OER | ExecResult | Not executed | NoExec | n |
| Cancelled | Cancelled | c |
| Successfully executed | OK | 0 |
| Insufficient position on option | NoPosition | 1 |
| Insufficient deposit | NoDeposit | 2 |
| Non-existing member | NoParticipant | 3 |
| Non-existing client | NoClient | 4 |
| Contract does not exist | NoInstrument | 6 |
| Not authorized | NoRight | 7 |
| Invalid quantity | InvalidVolume | 8 |
| Inadequate historical trade | NoEnoughHistoryTrade | 9 |
| 30 | Type of admin order | AOC | AdminOrderCommandFlag | Enforced offset resulting from position in delivery month not in integral multiples | NotMultipleForceClose | 1 |
| Initializing member's credit limit | InitCreditLimit | 2 |
| Adjusting member's credit limit | AlterCreditLimit | 3 |
| Canceling member's creadit limit | CancelCreditLimit | 4 |
| 31 | Whether or not to reserve the positions after exercise | EOPF | ExecOrderPositionFlag | Reserve | Reserve | 0 |
| Unreserve | Unreserve | 1 |
| 32 | Whether or not to close the position after exercise | EOCF | ExecOrderCloseFlag | Automatically close | AutoClose | 0 |
| Do not close | NotToClose | 1 |
| 33 | Encryption Type | ET | Encryption Type | RSA | RSA | 0 |

# Data Type List—Translated

| **Name of data type** | **Basic data type** | **Description of data type** |
| --- | --- | --- |
| TShfeFtdcErrorIDType | int | Error code |
| TShfeFtdcPriorityType | int | Priority |
| TShfeFtdcSettlementIDType | int | Settlement number |
| TShfeFtdcMonthCountType | int | Month number |
| TShfeFtdcTradingSegmentSNType | int | Trading session number |
| TShfeFtdcVolumeType | int | Quantity |
| TShfeFtdcTimeSortIDType | int | Sequence number by time |
| TShfeFtdcFrontIDType | int | Gateway ID |
| TShfeFtdcSessionIDType | int | Session ID |
| TShfeFtdcSequenceNoType | int | Sequence number |
| TShfeFtdcBulletinIDType | int | Bulletin ID |
| TShfeFtdcInformationIDType | int | MessageID |
| TShfeFtdcMillisecType | int | Time (in milliseconds） |
| TShfeFtdcVolumeMultipleType | int | Contract multiplier |
| TShfeFtdcImplyLevelType | int | Implied level |
| TShfeFtdcStartPosType | int | Starting position |
| TShfeFtdcAliasType | char[3] | Alias |
| TShfeFtdcOriginalTextType | char[3] | Original text |
| TShfeFtdcParticipantIDType | char[11] | Member ID |
| TShfeFtdcParticipantNameType | char[51] | Member name |
| TShfeFtdcParticipantAbbrType | char[9] | Abbreviation of member name |
| TShfeFtdcUserIDType | char[16] | Trader ID |
| TShfeFtdcPasswordType | char[41] | Password |
| TShfeFtdcClientIDType | char[11] | Client ID |
| TShfeFtdcInstrumentIDType | char[31] | Contract code |
| TShfeFtdcProductIDType | char[9] | Product code |
| TShfeFtdcProductNameType | char[21] | Product name |
| TShfeFtdcExchangeIDType | char[9] | Exchange ID |
| TShfeFtdcDateType | char[9] | Date |
| TShfeFtdcTimeType | char[9] | Time |
| TShfeFtdcInstrumentNameType | char[21] | Contract name |
| TShfeFtdcProductGroupIDType | char[9] | Product group code |
| TShfeFtdcProductGroupNameType | char[21] | Product groupname |
| TShfeFtdcMarketIDType | char[9] | Market ID |
| TShfeFtdcSettlementGroupIDType | char[9] | Settlement group ID |
| TShfeFtdcOrderSysIDType | char[13] | Order number |
| TShfeFtdcOTCOrderSysIDType | char[13] | OTC order number |
| TShfeFtdcExecOrderSysIDType | char[13] | System number of exercise declaration |
| TShfeFtdcQuoteSysIDType | char[13] | Quote number |
| TShfeFtdcTradeIDType | char[13] | Trade number |
| TShfeFtdcOrderLocalIDType | char[13] | Local order number |
| TShfeFtdcComeFromType | char[21] | Origin of message |
| TShfeFtdcAccountIDType | char[13] | Account number |
| TShfeFtdcNewsTypeType | char[3] | Bulletin type |
| TShfeFtdcAdvanceMonthType | char[4] | Months in advance |
| TShfeFtdcCommodityIDType | char[9] | Commodity code |
| TShfeFtdcIPAddressType | char[16] | IP address |
| TShfeFtdcProductInfoType | char[41] | Product information |
| TShfeFtdcProtocolInfoType | char[41] | Protocol information |
| TShfeFtdcBusinessUnitType | char[21] | Business unit |
| TShfeFtdcTradingSystemNameType | char[61] | Name of trading system |
| TShfeFtdcTradingRoleType | char | Trading role |
| TShfeFtdcUserTypeType | char | Trader type |
| TShfeFtdcProductClassType | char | Product sector |
| TShfeFtdcOptionsTypeType | char | Option type |
| TShfeFtdcInstrumentStatusType | char | Instrument status |
| TShfeFtdcDirectionType | char | Buy-sell direction |
| TShfeFtdcPositionTypeType | char | Position type |
| TShfeFtdcPosiDirectionType | char | Position direction (long or short) |
| TShfeFtdcExchangeDataSyncStatusType | char | Synchronization status of the Exchange's data |
| TShfeFtdcSGDataSyncStatusType | char | Synchronization status of settlement group's data |
| TShfeFtdcHedgeFlagType | char | Flag of speculation or hedge |
| TShfeFtdcClientTypeType | char | Client type |
| TShfeFtdcInstStatusEnterReasonType | char | Cause for instrument entering current status |
| TShfeFtdcOrderPriceTypeType | char | Price condition |
| TShfeFtdcOffsetFlagType | char | Flag of position opening orclosing |
| TShfeFtdcForceCloseReasonType | char | Reasons for enforced closing (offset) |
| TShfeFtdcOrderStatusType | char | Order status |
| TShfeFtdcOrderTypeType | char | Order type |
| TShfeFtdcOTCOrderStatusType | char | OTC order status |
| TShfeFtdcTimeConditionType | char | Time condition |
| TShfeFtdcVolumeConditionType | char | Quantity condition |
| TShfeFtdcContingentConditionType | char | Trigger conditions |
| TShfeFtdcActionFlagType | char | Action type |
| TShfeFtdcOrderSourceType | char | Origin of order |
| TShfeFtdcTradeTypeType | char | Trade type |
| TShfeFtdcPriceSourceType | char | Source of trade price |
| TShfeFtdcAccountStatusType | char | Account status |
| TShfeFtdcMemberTypeType | char | Member type |
| TShfeFtdcExecResultType | char | Execution result |
| TShfeFtdcYearType | int | Year |
| TShfeFtdcMonthType | int | Month |
| TShfeFtdcLegMultipleType | int | Leg multiplier |
| TShfeFtdcLegIDType | int | Leg ID |
| TShfeFtdcBoolType | int | Bool type |
| TShfeFtdcUserActiveType | int | Trader active or not |
| TShfeFtdcPriceType | double | Price type |
| TShfeFtdcUnderlyingMultipleType | double | Underlying commodity multiplier |
| TShfeFtdcCombOffsetFlagType | char[5] | Flag of position opening orclosingfor spread |
| TShfeFtdcCombHedgeFlagType | char[5] | Flag of speculation or hedge for spread |
| TShfeFtdcRatioType | double | Ratio |
| TShfeFtdcMoneyType | double | Fund type |
| TShfeFtdcLargeVolumeType | double | Large quantity |
| TShfeFtdcNewsUrgencyType | char | Urgency |
| TShfeFtdcSequenceSeriesType | short | Series number |
| TShfeFtdcCommPhaseNoType | short | Communication phase number |
| TShfeFtdcContentLengthType | int | Length of payload |
| TShfeFtdcErrorMsgType | char[81] | Error message |
| TShfeFtdcAbstractType | char[81] | Message abstract |
| TShfeFtdcContentType | char[501] | Message body |
| TShfeFtdcURLLinkType | char[201] | Web address |
| TShfeFtdcIdentifiedCardNoType | char[51] | IdentificationNumber |
| TShfeFtdcIdentifiedCardNoV1Type | char[21] | Original IdentificationNumber |
| TShfeFtdcPartyNameType | char[81] | Name of participant |
| TShfeFtdcIdCardTypeType | char[16] | Type of ID |
| TShfeFtdcAdminOrderCommandFlagType | char | Administrative order |
| TShfeFtdcDataCenterIDType | int | Datacenter ID |
| TShfeFtdcBusinessLocalIDType | int | Local business ID |
| TShfeFtdcCurrencyIDType | char[4] | Currency ID |
| TShfeFtdcRateUnitType | int | Exchange Rate Unit |
| TShfeFtdcExRatePriceType | double | Exchange Rate |
| TShfeFtdcExecOrderPositionFlagType | char | Flag indicating whether or not to keep position after option being exercised |
| TShfeFtdcExecOrderCloseFlagType | char | Flag indicating whether or not to offset position after option being exercised |
| TShfeFtdcExecOrderCloseFlag  Type | Char | Flag indicating whether or not the position resulting from the exercise will be offset |
| TShfeFtdcMacAddressType | Char[21] | Mac address |

1. The contents in the unused fields are meaningless, therefore the member system should not assume any meaning from them. Keeping these unused fields in the structure will not add to the bandwidth because TraderAPI compress the messages. Keeping them here is for the purpose of version compatibility and scalability. [↑](#footnote-ref-1)