Trading API & Market Data API
Interface Specifications

Version: 2.00
Document Release Date: 20250912

I. Revision Records, Approval Records and Audit Records

Revision Records

Version No.	Date of Revision	Major Revisions
Version: 2.00	20250829	

Approval Records

Approving Officers	Department (Unit)	Date of Approval

Audit Records

Auditors	Department (Unit)	Date of Audit

Table of contents

Part I Introduction to NGES Trading System Interface

Chapter 1 gives you an introduction to the two main APIs for the NGES Trading System—***TraderAPI*** and ***MduserAPI***. ***TraderAPI*** is designed for Member System to send instructions for trading and query, and to receive private stream, public stream, dialog stream and query stream; ***MduserAPI*** is designed for Member System and Market Data Vendor System to receive market data stream.

Chapter 2 introduces the ***Futures Trading Data (FTD) Exchange Protocol*** behind the two APIs, with a focus on illustration of data stream.

Chapter 3 introduces the programming interfaces of the two APIs with respect to different types of applications.

Chapter 4 introduces the operating mode of the two APIs, including inter-thread communication, heartbeat mechanisms, and transmission mechanisms of reliable data stream.

1. Introduction

1.1. Background

Under the unified leadership of the China Securities Regulatory Commission (CSRC), Shanghai Futures Exchange (SHFE), Zhengzhou Commodity Exchange (ZCE) and Dalian Commodity Exchange (DCE) jointly studied and developed the Futures Trading Data Exchange Protocol (i.e. ***FTD*** or ***FTD Protocol***). The CSRC officially released the ***FTD Protocol*** (JR/T 0016-2004) on 25th March, 2005, and implemented it as an industry standard ever since.

The NGES Trading System intrinsically uses the ***FTD Protocol*** as the access protocol for Exchange Member's remote trading. The ***FTD Protocol*** is relatively complex. In order to reduce the difficulty level of developing Exchange Member's remote Trading Systems and improve the reliability level of the Trading Systems, SHFE released a trading API (***TraderAPI***) and a market data API (***MduserAPI***) for the NGES Trading System.

The Member Systems call the ***TraderAPI*** to connect to the NGES Trading System, send request instructions, and receive responses or returns, after which the ***TraderAPI*** calls back the Member Systems. Similarly, the Member Systems or the Market Data Vendor Systems call the ***MduserAPI*** to dock with the NGES Trading System and receive market data, after which the ***MduserAPI*** calls back the Market Data Receiving Systems.

The systems used by Exchange Members and Market Data Vendors to receive the Exchange's market data are collectively termed the Market Data Receiving Systems. Both Member Systems and Market Data Receiving Systems are referred to as Member-End or Member-End Systems.

The ***TraderAPI*** encapsulates the complex protocol conversion, data synchronization and network communication between Member Systems and the NGES Trading System. Operating over TCP protocol, the ***TraderAPI*** establishes a virtual link communication channel with the trading front-end processor of the NGES Trading System, enabling trading and query operations for the Member Systems. The connection channel established via ***TraderAPI*** is characterized by its multi-address registration, automatic reconnection and trading data auto-retransmission, etc.

Similar to ***TraderAPI***, ***MduserAPI*** will establish a TCP-based virtual link channel to connect to the NGES Trading System's market data front-end, enabling subscription to and reception of market data.

1.2. TraderAPI Overview

TraderAPI is a C++-based class library that enables trading functionalities by utilizing and extending its provided interfaces. These functionalities include: order and quote entry, order and quote cancellation, submission of option exercise and option abandonment, cancellation of option exercise and option abandonment, request for quote, fund query, order and quote request, trade report query, client information query, member position query, clients position query, contract query, and contract trading status query.

The Windows platform class library supports Windows32 and includes the following five files:

File Name	File Description
FtdcTraderApi.h	Trading API header file
FtdcUserApiStruct.h	Data structures header files
FtdcUserApiDataType.h	Data types header files
ftdtraderapi.dll	Dynamic-link library (DLL) binary file
ftdtraderapi.lib	Import library (.Lib) file

It is recommended to use Visual Studio 2017 or later versions of the compiler.

The Linux-like platform class library supports RHEL7 and Kylin V10, and includes the following four files:

File Name	File Description
FtdcTraderApi.h	Trading API header file
FtdcUserApiStruct.h	Data structures header files
FtdcUserApiDataType.h	Data types header files
libftdtraderapi.so	Dynamic-link library (DLL) binary file

1.3. MduserAPI Overview

MduserAPI is also a C++ based class library that enables market data subscription and reception functionalities through utilizing and extending its provided interfaces.

The Windows platform class library supports Windows32 and includes the following five files:

File Name	File Description
FtdcMduserApi.h	Market data API header file
FtdcUserApiStruct.h	Data structures header files
FtdcUserApiDataType.h	Data types header files
ftdmdapi.dll	Dynamic-link library (DLL) binary file
ftdmdapi.lib	Import library (.Lib) file

It is recommended to use Visual Studio 2017 or later versions of the compiler.

The Linux-like platform class library supports RHEL7 and Kylin V10, and includes the following four files:

File Name	File Description
FtdcMduserApi.h	Market data API header file
FtdcUserApiStruct.h	Data structures header files
FtdcUserApiDataType.h	Data types header files
libftdmdapi.so	Dynamic-link library (DLL) binary file

1.4. Platforms Supported by TraderAPI/MduserAPI

Currently, the following platforms are supported:

- ARM64/KylinV10: including .h files and .so files
- X86-64/KylinV10: including .h files and .so files
- X86-64/RHEL7: including .h files and .so files
- X86/Windows32: including .h files, .dll files, and .lib files

1.5. Contact

Tel: +86-021-68400802
E-mail: tech@shfe.com.cn

1.6. Version History

1.6.1. Version v2.00

The main changes in this version compared to API 1.0 are as follows:

- The following function interfaces have been removed in this version:
  - **Traderapi:** removed ***ReqAdminOrderInsert*** and ***OnRspAdminOrderInsert*** interfaces.
  - **Traderapi:** removed ***ReqQryMBLMarketData*** and ***OnRspQryMBLMarketData*** interfaces.
  - **Traderapi:** removed ***RegisterCertificateFile*** interface.
  - **Traderapi:** removed ***RegisterGMCertificateFile*** interface.
  - **Traderapi:** removed ***OnRtnAliasDefine*** interface.
  - **Traderapi:** removed ***ReqCombOrderInsert***, ***OnRspCombOrderInsert***, ***ReqQryCombOrder***, ***OnRspQryCombOrder***, ***OnRtnCombOrder***, ***OnErrRtnCombOrderInsert***, ***OnRtnInsCombinationLeg***, and ***OnRtnDelCombinationLeg*** interfaces.
  - **Traderapi:** removed ***ReqQryInformation*** and ***OnRspQryInformation*** interfaces.
  - **Traderapi:** removed ***OnRtnDelInstrument*** interface.
  - **Traderapi:** removed ***ReqQryCreditLimit*** and ***OnRspQryCreditLimit*** interfaces.
  - **Traderapi:** removed ***RegisterCryptAlgorithm*** interface.
  - **MDuserapi:** removed ***RegisterCertificateFile*** interface.
  - **MDuserapi:** removed ***RegisterGMCertificateFile*** interface.
  - **MDuserapi:** removed ***RegisterCryptAlgorithm*** interface.
- The following function interfaces have been added in this version:
  - ***MDuserAPI***: added "***User Password Update Request***" interface, cf. [Part III, 2.2.18: ReqUserPasswordUpdate Method].
  - ***MDuserAPI***: added "***Opening Request Log File***" and "***Opening Response Log File***" interfaces, cf. [Part III, 2.2.11 OpenRequestLog Method and 2.2.12 OpenResponseLog Method].

- The following function interfaces have been modified in this version:
  - ***TraderAPI***: ***RegisterFront*** and ***RegisterNameServer*** interfaces have been updated, with the parameter type changed from char* to const char*, cf. [Part II, 2.2.8 RegisterFront Method and 2.2.9 RegisterNameServer Method].
  - ***MduserAPI***: ***RegisterFront*** and ***RegisterNameServer*** interfaces have been updated with the parameter type has changed from char* to const char*, cf. [Part III, 2.2.8 RegisterFront Method and 2.2.9 RegisterNameServer Method].
- The following function return values and error reasons have been added in this version:
  - ***TraderAPI*** and ***MduserAPI***: added return values for request interfaces in, cf. [Part IV "API Return Value List"].
  - ***TraderAPI***: updated and expanded the API disconnection reasons, cf. [Part II, 2.1.2 OnFrontDisconnected Method].
  - ***MduserAPI***: updated and expanded the API disconnection reasons, cf. [Part III, 2.1.2 OnFrontDisconnected Method].
- The following enumeration values have been added in this version:
  - **ProductClass additions:** SHFE_FTDC_PC_Spread.
  - **HedgeFlag additions:** SHFE_FTDC_HF_None.
  - **OffsetFlag additions:** SHFE_FTDC_OF_None.
  - **TradeType additions:** SHFE_FTDC_TRDT_SpreadDerived.
  - **PriceSource additions:** SHFE_FTDC_PSRC_Imply.

2. FTD Architecture

2.1. Communication Mode

All communications in the ***FTD Protocol*** are based on specific communication modes. Essentially, a communication mode defines the coordination mechanism between communicating parties.

The FTD protocol supports three communication modes:

- **Dialog Mode**
- **Private Mode**
- **Public Mode** (i.e. the Broadcast Communication Mode in API 1.0)

Dialog Mode

The Member System initiates communication requests which are received, processed and responded by the Trading System. Typical operations include order entry and information queries. This mode follows the standard client/server architecture.

Private Mode

The Trading System actively pushed messages to specific members or their designated traders based on subscription requests initiated by the Member Systems. Instances include execution reports and market data notifications.

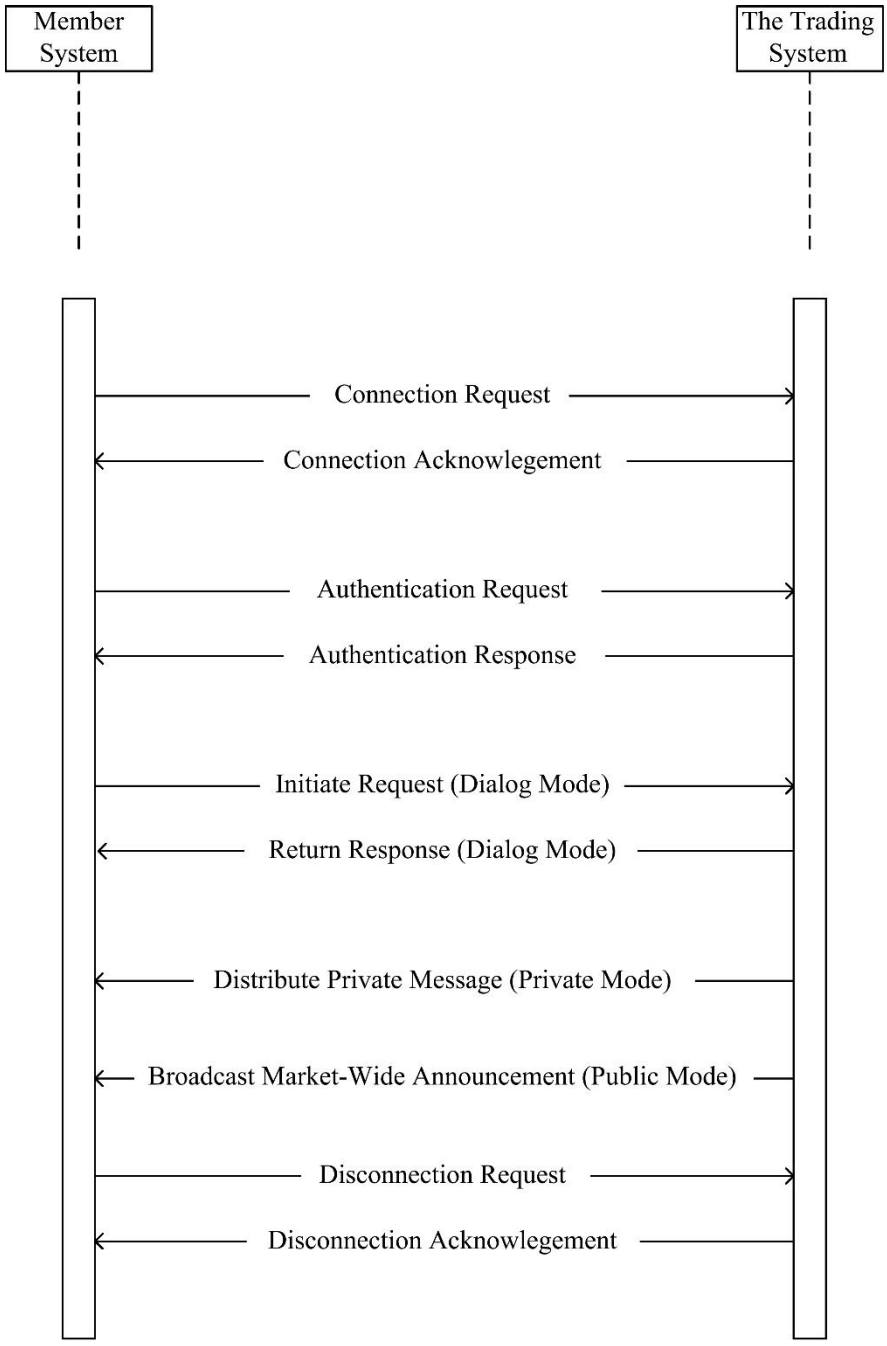
Public Mode

The Trading System broadcast identical messages to all members. Instances include

public announcements and market-wide information.

Communication modes and network connections don't maintain simple one-to-one relationships. To be specific, a single network connection may carry messages sent in multiple communication modes, while messages sent in one communication mode can also be transmitted across multiple network connections.

All communication modes follow the process shown in Figure 1:



Figture 1: Message Flow Diagrams for All Communication Modes

2.2. Data Flows

The Trading Front-end supports ***Dialog Mode***, ***Private Mode***, and ***Public Mode***, while

the Market Data Front-end supports only ***Dialog Mode*** and ***Private Mode***.

1) Dialog Mode

The ***Dialog Mode*** is bidirectional, supporting both dialog data flows (referred to as ***DialogFlows***) and query data flows (referred to as ***QueryFlows***).

The Member System sends a trading request or query request, and the Trading System returns a response. No state is maintained for ***DialogFlows*** or ***QueryFlows***. In the event of a system failure, both streams will be reset, and in-transit data may be lost.

2) Private Communication Mode

Private Communication Mode is unidirectional and supports member's private streams, trader's private streams, and market data topic streams (referred to as market data streams).

In ***Private Communication Mode***, the data streams are reliable, and the Trading System maintains the private or market data streams across the entire system. Within a trading day, when Member End System resumes its connection after a disconnection, it can request the Trading System to send the data within private streams or market data streams following a designated sequence number. The private stream delivers information such as order return and trade return to the Member System, while the market data stream provides market data information to the Market Data Receiving System. Private streams are classified into member's private stream and trader's private stream.

The Trading System maintains the private stream of each member. All member-specific return messages, such as order return and trade return, will be released through the member's private stream. Access to a member's private stream requires the trader to have the corresponding subscription permissions.

Trader's private stream is similar to member's private stream, but it only covers return message for trades initiated by a particular trader. Every trader has the right to subscribe to his or her own trader's private stream.

The market data provided by the Trading System is organized by topics. Each topic contains market data for a group of contracts. The Exchange defines which topics each market data user is allowed to subscribe to. Each market data topic corresponds to a market data stream.

To receive market data notifications, the Market Data Receiving System must subscribe to one or more market data topics after connecting to the market data front-end processor.

3) Public Communication Mode

The ***Public Communication Mode*** is bidirectional and supports public data streams (referred to as public streams).

The Trading System sends market public information to the Member System. The public stream is reliable, and the Trading System maintains the public streams across the entire system. Within a trading day, when Member System resumes its connection after a disconnection, it can request the Trading System to send the data within public streams following a designated sequence number.

3. Interface Mode

3.1. TraderAPI Interface

TraderAPI provides two interfaces: ***CShfeFtdcTraderApi*** and ***CShfeFtdcTraderSpi***. These two interfaces are encapsulation on the ***FTD Protocol***.

Member System can send operating requests via ***CShfeFtdcTraderApi***; and it can handle/process the response and reply from the NGES Trading System by inheriting ***CShfeFtdcTraderSpi*** and reloading the callback functions.

3.1.1. Dialog Stream and Query Stream Programming Interface

The programming interface for communication through dialog stream typically looks like below.

```
////Request:
int CShfeFtdcTraderApi::ReqXXX(
        CShfeFtdcXXXField* pReqXXX,
        int nRequestID);
////Response:
void CShfeFtdcTraderSpi::OnRspXXX(
        CShfeFtdcXXXField* pRspXXX,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

The request interface contains two parameters.

The 1st parameter is the requested content, and it cannot be left as empty. This parameter would use a class according to the type of the request command/content. Please refer to the appendix "**Enumeration Value List**" and "**Data Type List**" for variable types and allowed values for the members of this class.

The 2nd parameter is the request ID. The request ID is maintained by Member System and the Exchange advises that every request ID should be unique. The request ID filled in upon sending the request would be sent back to Member System together with the response from the NGES Trading System, and user can match a particular request with a particular response by using this number.

The ***CShfeFtdcTraderSpi*** callback function/method would be called upon getting reply from the Trading System. If there is more than one piece of response data, the callback function/method would be called multiple times.

The callback function requires four input parameters:

The 1st parameter is the actual data in the response. If there is an error in the process or if there is no such result, this field may be NULL.

The 2nd parameter is the processed result, indicating whether the processing of the result for the current request is a success or a failure. If multiple callbacks occur, the value for this parameter from the 2nd callback onwards might all be NULL.

The 3rd parameter is the request ID filled in when sending the request.

The 4th parameter is the flag for the end of response, indicating whether this is the last callback for the current response.

3.1.2. Private Stream Programming Interface

As described in section 2.2, data via the private stream is private information for a particular Exchange Member or a particular trader, including order return, transaction return, quote return, declaration return etc.

The programming interface for receiving return message via private stream typically looks like:

```
void CShfeFtdcTraderSpi::OnRtnXXX(CShfeFtdcXXXField* pXXX);
////or
void CShfeFtdcTraderSpi::OnErrRtnXXX(
        CShfeFtdcXXXField* pXXX,
        CShfeFtdcRspInfoField* pRspInfo);
```

The ***CShfeFtdcTraderSpi*** callback function/method would be called upon getting return data from the Trading System via the private stream. The parameter of the callback function is the specific content of the return.

3.1.3. Public Stream Programming Interface

Public stream data includes public information such as Exchange contracts and announcements.

The programming interface for receiving return message via public stream typically looks like:

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

The ***CShfeFtdcTraderSpi*** callback function/method would be called upon getting return data from the Trading System via the public stream. The parameter of the callback function is the specific content of the return.

3.2. MduserAPI Interface

Similar to the ***TraderAPI***, ***MduserAPI*** also provides two interfaces: ***CShfeFtdcMduserApi*** and ***CShfeFtdcMduserSpi***. These two interfaces are encapsulation on the ***FTD Protocol***.

Market Data Receiving System can send operation request via ***CShfeFtdcMduserApi*** and it can process the return or response from the NGES Trading System by inheriting ***CShfeFtdcMduserSpi*** and reloading the callback functions.

3.2.1. Dialog Stream Programming Interface

The programming interface for communication through dialog stream typically looks like below.

```
////Request:
int CShfeFtdcMduserApi::ReqXXX(
        CShfeFtdcXXXField* pReqXXX,
        int nRequestID);
////Response:
void CShfeFtdcMduserSpi::OnRspXXX(
        CShfeFtdcXXXField* pRspXXX,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

The request interface contains two parameters.

The 1st parameter is the requested content, and it cannot be left as empty.

The 2nd parameter is the request ID. The request ID is maintained by Market Data Receiving System and the Exchange advises that every request ID should be unique. The request ID filled in upon sending the request would be sent back to Member System together with the response from the NGES Trading System, and user can match a particular request with a particular response by using this number.

The ***CShfeFtdcMduserSpi*** callback function/method would be called upon getting reply from the Trading System. If there is more than one piece of response data, the callback function/method would be called multiple times.

The callback function requires four input parameters:

The 1st parameter is the actual data in the response. If there is an error in the process or if there is no such result, this field may be NULL.

The 2nd parameter is the processed result, indicating whether the processing of the result for the current request is a success or a failure. If multiple callbacks occur, the value for this parameter from the 2nd callback onwards might all be NULL.

The 3rd parameter is the request ID filled in when sending the request.

The 4th parameter is the flag for the end of response, indicating whether this is the last callback for the current response.

3.2.2. Market Data Stream Programming Interface

Market data stream carries market data information released by the Trading System.

The programming interface for receiving return message via market data stream typically looks like:

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

The ***CShfeFtdcMduserSpi*** callback function/method would be called when receiving market data. The parameter of the callback function is the specific content of the declaration.

4. Operating Mode

4.1. Workflow

The interaction process between the Member End System and the Trading System can be divided into two stages: the initialization phase and the function calling phase.

4.1.1. Initialization Phase

In the initialization phase, Member End System has to complete the steps below (for more details, please refer to the IDs in the Development Instance section).

Steps	Member System	Market Data Receiving System
1	Generate an instance of CShfeFtdcTraderApi;	Generate an instance of CShfeFtdcMduserApi;
2	Generate an event handler instance;	Generate 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	Register the network communication address of the trading front-end NameServer.	Register the network communication address of the market data front-end NameServer.
6	Initialization	Initialization

4.1.2. Function Calling Phase

In the function calling phase, Member End System can call any of request methods from the trading or market data interface, e.g. ***ReqUserLogin***, ***ReqOrderInsert***, etc, and also provide callback functions to receive response and return messages. It should be noted that:

1) Input parameters for the API request function cannot be NULL.
2) The meaning of the output parameter returned from the API request function is: 0 stands for success, other numbers indicate an error. For details of error IDs, please refer to the Appendix for "**Return Value List**".
3) The Member End System is subject to flow control when sending request commands. If the flow control limit is exceeded, the current request will fail to be sent.
4) Flow control includes communication flow control and in-transit flow control. Communication flow control limits the number of requests that can be sent within one second, while in-transit flow control limits the number of requests that have been sent but have not yet received a response.

4.2. Working Thread

The Member End System consists of at least two threads: one is the application program as the main thread, and the other is the API working thread (***TraderAPI*** or ***MduserAPI***). The communication between the application program and the trading front-end or market data

front-end is driven by the API working thread.

The interfaces provided by ***CShfeFtdcTraderApi*** and ***CShfeFtdcMduserApi*** are thread-safe and can be invoked simultaneously by multiple threads.

The callback interface provided by ***CShfeFtdcTraderSpi*** is driven by the working threads of ***TraderAPI***. It receives the required data from the front-end of the Trading System by implementing the interface method of SPI.

Similarly, the callback interface provided by ***CShfeFtdcMduserSpi*** is driven by the ***MduserAPI*** working thread. It collects the required data from market data front-end by implementing the interface method of SPI.

If there is blocking in callback function of the overloaded application program, ***TraderAPI*** or ***MduserAPI*** working thread would also be blocked. In the case, the communication between API and trading front-end or market data front-end would stop; therefore, usually quick return is required for callback functions.

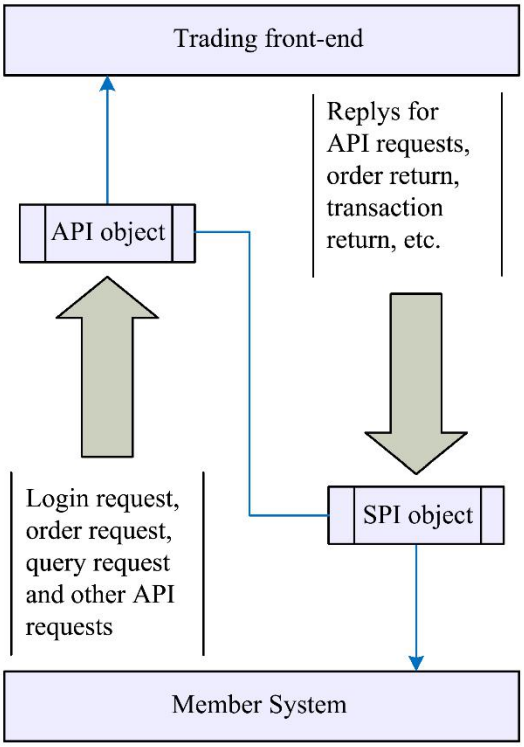


Chart 2. TradeAPI working thread

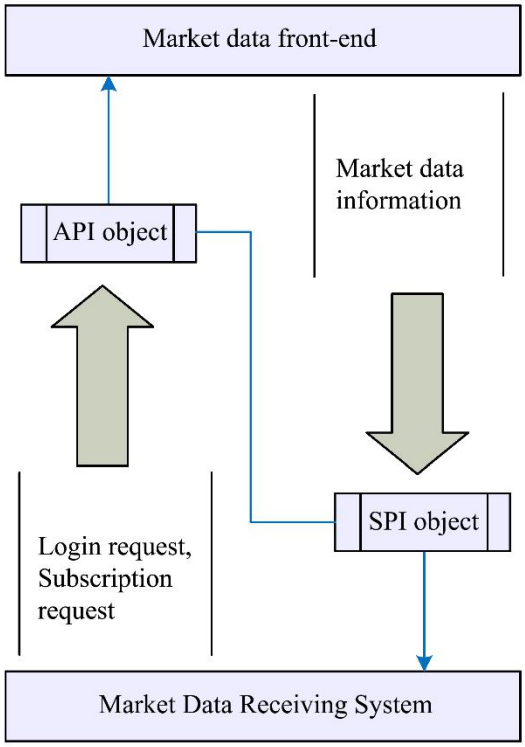


Chart 3. MduserAPI working thread

4.3. Connection with the Trading System

TraderAPI and ***MduserAPI***, using the ***FTD protocol***, communicate with the trading front-end and market data front-end, respectively. They register the ***Front-End Name Server (FENS)*** addresses via the ***RegisterNameServer*** method to establish connections with the trading and market data front-ends.

The Exchange deploys multiple trading and market data front-end servers to achieve load balancing and mutual backup, thereby enhancing system performance and reliability. To ensure communication reliability during trading, ***TraderAPI*** and ***MduserAPI*** can register multiple FENS addresses. After initialization, the API will attempt to establish a network connection by selecting one of the registered FENS addresses. If the connection fails, it will

continue trying the remaining addresses one by one until a connection is successfully established.

The Exchange will publish at least two FENS addresses; therefore, the Member System should register at least two FENS addresses to avoid single points of failure in case the connected FENS becomes unavailable.

4.4. Interaction Between TraderAPI and the Trading Front-end

The Member System interacts with the trading front-end through ***TraderAPI***. Requests from the Member System are sent to the Trading Front-end via ***TraderAPI***. Responses and returns from the Trading Front-end are returned to the Member System through ***TraderAPI***.

The trading interfaces and private stream interfaces of ***TraderAPI*** are correlated. For instance, when a user (i.e., trader) submits an order entry request via ***ReqOrderInsert***, the system will return an order response ***OnRspOrderInsert*** to indicate that the Trading System has received the order. Once the order enters the Trading System, if there is any change in the order's status, an order return ***OnRtnOrder*** will be returned. If the order is matched (either fully or partially), a transaction return ***OnRtnTrade*** will be received. In such cases, the order and transaction returns of one user may also be received by other traders under the same member firm, provided those traders have the permission to subscribe to the member's private stream and have done so.

Let's illustrate the concept with a day-to-day trading instance. Assuming there are two Member Systems A and B, the interaction process is as follows:

1) Trader A places an order, with details: cu2511, buy, 20 lots, 74,000 RMB
   - **CShfeFtdcTraderApi::ReqOrderInsert:** Order entry request. This function is called by the main application thread of Member System, and sent to the front-end of the Trading System through dialog stream.
   - **Order Processing of the Trading System:** The order's System ID is numbered 1. Because there is no counterparty in matching queue at the moment, the order status is "**Not Traded and Still Queuing**". The front-end of the Trading System send order response to the dialog stream of Trader A; the delivered order is returned to the private stream of Trader A and the private stream of the member to whom Trader A is subordinate. Both the order response and the order return message are processed by ***TraderAPI*** working thread with the calling of the SPI object methods.
   - **CShfeFtdcTraderSpi::OnRspOrderInsert:** The front-end of the Trading System provides a reply for the request with contents: entry is successful, and the order with Local ID 1 is numbered as System ID 1. This function is called by ***TraderAPI*** working thread after receiving the reply from the front-end of the Trading System.
   - **CShfeFtdcTraderSpi::OnRtnOrder:** The front-end of the Trading System immediately provides order return to private stream of Trader A or private stream of the Member to whom Trader A is subordinate. This function is called by the ***TraderAPI*** working thread after receiving the order return from the front-end of the Trading System. If there are other traders of Member A who login into the Trading System and subscribe to the private stream of Member

A, they will receive the same order return message (similarly in the below case).

2) TraderB places an order, with details: cu2511, sell, 10 lots, 74,000 RMB
   - **CShfeFtdcTraderApi::ReqOrderInsert:** Order entry request.
   - **Order Processing of the Trading System:** The order's System ID is numbered 2. Matching is attempted and succeeds, thus the order is in the status of "All Filled". The front-end of the Trading System sends: order response to Trader B's dialog stream; order return to the private stream of Trader B and the private stream of the Member to whom Trader B is subordinate; transaction return to the private stream of Trader B and the private stream of the Member to whom Trader B is subordinate; order return to the private stream of Trader A and the private stream of the Member to whom Trader A is subordinate, informing that the status of the order with System ID 1 has been changed by the Trading System to "**Partially Filled and Still Queuing**", and that the "remaining unfilled lot" is 10; transaction return to the private stream of Trader A and the private stream of the Member to whom Trader A is subordinate. **NGES Trading System would ensure that: order return would be delivered to Member System ahead of the transaction return; "remaining unfilled lot" field in order return has already reflected the updated amount in the order book of the Trading System.**
   - **CShfeFtdcTraderSpi::OnRspOrderInsert:** The trading front-end provides a reply for the request, with contents that order entry is successful, and the order with Local ID 1 is numbered with System ID 2.
   - **CShfeFtdcTraderSpi::OnRtnOrder:** The trading front-end provides order return to the private stream of Trader B and the private stream of the Member to whom Trader B is subordinate; the order status is "All Filled".
   - **CShfeFtdcTraderSpi::OnRtnTrade:** The trading front-end provides transaction return to the private stream of Trader B and the private stream of the Member to whom Trader B is subordinate.
   - **CShfeFtdcTraderSpi::OnRtnOrder:** The trading front-end provides order return to the private stream of Trader A and the private stream of the Member to whom Trader A is subordinate; Order status is "Partially Filled and Still Queuing", and the "remaining unmatched lot" is 10.
   - **CShfeFtdcTraderSpi::OnRtnTrade:** The trading front-end provides transaction return to the private stream of Trader A and the private stream of the Member to whom Trader A is subordinate.
3) Trader A cancels the order
   - **CShfeFtdcTraderApi::ReqOrderAction:** Order operating request. This function is called by the Member System and sent to the front-end of the Trading System through dialog stream.
   - **Cancellation processing by the Trading System:** The remaining order with System ID 1 is canceled. The front-end of the Trading System send cancellation response to the dialog stream of Trader A; the delivered order is returned to the private stream of Trader A and the private stream of the member to whom Trader A is subordinate. Both the order response and the order return message are processed by ***TraderAPI*** working thread with the

calling of the SPI object methods.

- **CShfeFtdcTraderSpi::OnRspOrderAction:** A response to the request is given by the front-end of the Trading System, with the content being: The cancellation was successful. This function is called by ***TraderAPI*** working thread after receiving the reply from the front-end of the Trading System.
- **CShfeFtdcTraderSpi::OnRtnOrder:** This function is called by the ***TraderAPI*** working thread after receiving the order return from the front-end of the Trading System. If there are other traders of Member A who login into the Trading System and subscribe to the private stream of Member A, they will receive the same order return message.

The following chart describes the UML interaction among the Member System, ***TraderAPI*** and the Trading System.

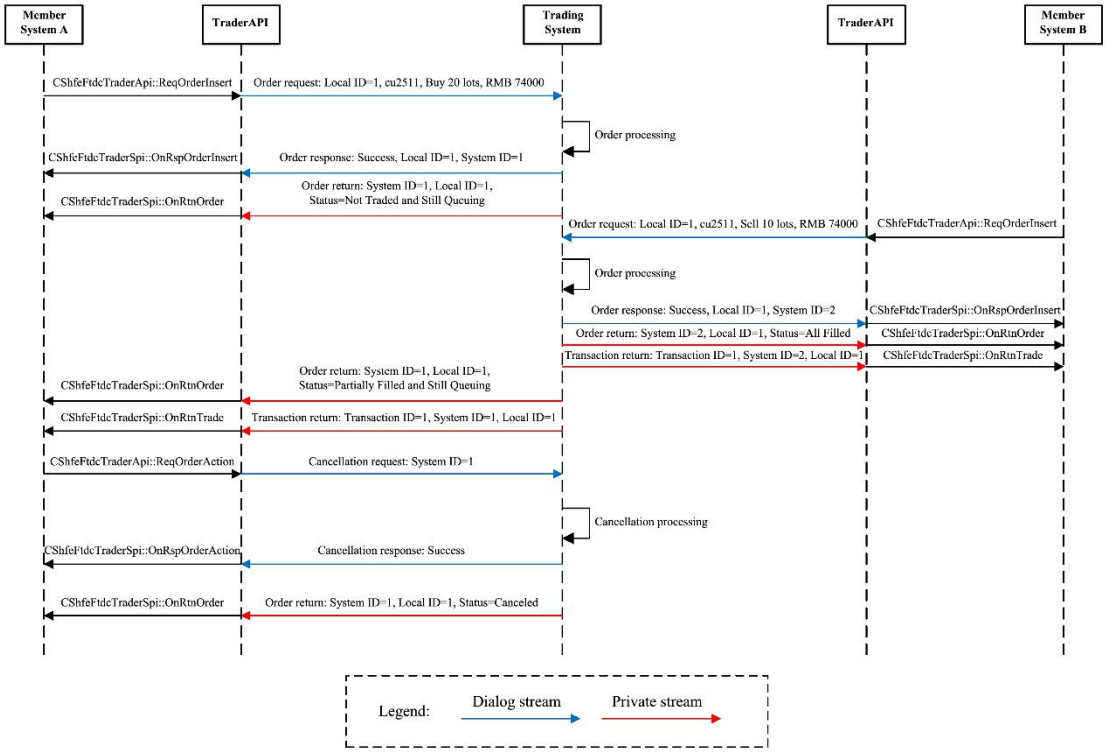


Chart 4: Illustration of the interaction between Member System and the Trading System

4.5. Interaction Between MduserAPI and the Market Data Front-end

The Market Data Receiving System interacts with the Market Data Front-end via ***MduserAPI***. Requests from the Market Data Receiving System are sent to the Market Data Front-end through ***MduserAPI***, and responses and returns from the front-end are returned to the receiving system through ***MduserAPI***.

Take a market data vendor's subscription to market data as an instance. The market data vendor subscribes to a market data topic 1001 in a snapshot mode via the Market Data Receiving System A, and the interaction process is as follows:

- **CShfeFtdcMduserApi::SubscribeMarketDataTopic:** Subscription to a market data topic. This function is called by the Market Data Receiving System.
- **CShfeFtdcMduserApi::ReqUserLogin:** Login request. This function is called by

the Market Data Receiving System, and sent to the front-end of the Market Data Front-end through dialog stream.

- **Request processing by the Trading System:** if the login request is valid, the Market Data Front-end will send a login request response to market data vendors through the dialog stream, and send market data notifications to market data vendors through the market data stream. Both the order response and the market data notification message are processed by ***MduserAPI*** working thread.
- **CShfeFtdcMduserSpi::OnRspUserLogin:** The response to the login request is given by the Market Data Front-end. This function is called by ***MduserAPI*** working thread after receiving the reply from the Market Data Front-end.
- **CShfeFtdcMduserSpi::OnRtnDepthMarketData:** The Market Data Front-end sends the latest snapshot for topic 1001 through the market data stream subscribed by market data vendors. This function is called by ***MduserAPI*** working thread after receiving the market data notification from the Market Data Front-end.
- **CShfeFtdcMduserApi::OnRtnDepthMarketData:** The Market Data Front-end sends incremental updates for topic 1001 through the market data stream subscribed by market data vendors.This function is called by ***MduserAPI*** working thread after receiving the market data notification from the Market Data Front-end.

The following chart describes the UML interaction among the Market Data Receiving System, ***MduserAPI*** and the Trading System.

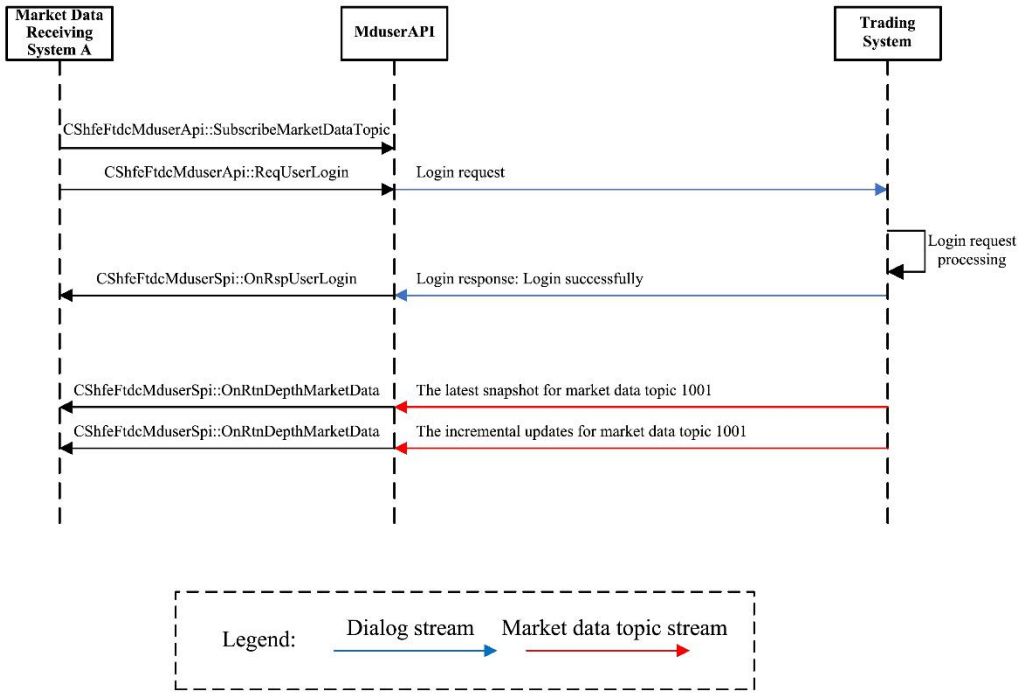


Chart 5: Illustration of the interaction between Market Data Receiving System and the Trading System

4.6. Local Files

During runtime, ***TraderAPI*** would write some data into local files. When calling the ***CreateFtdcTraderApi*** function, an input parameter can be passed to specify the local file path.

This path must be created before runtime. The file extension of all local files is “trade.con”.

MduserAPI works similarly as ***TraderAPI***, whereas the function called is ***CreateFtdcMduserApi***. The file extension of all local files is “md.con”.

4.7. Request and Response Log Files

TraderAPI offers two log interfaces for recording communication logs. ***OpenRequestLog*** is used to open the request log and ***OpenResponseLog*** is used to open the response log. If the logs are opened, all service requests would be written into the request log, and all service responses and returns would be recorded into the response log. Password fields in login requests, password change requests/responses, and the authentication token in terminal authentication requests are omitted from the logs.

Request Format:

Timestamp, request name, request parameter name = “request parameter content”

Response Format:

Timestamp, response name, response ID, response information, response parameter name = “response parameter content”

Return Format:

Timestamp, return name, return parameter name = “return parameter content”

4.8. Subscription Methods for Reliable Data Stream

In the ***FTD protocol***, the private stream, public stream and market data stream, etc, which can transmit data from one side to the other side in a reliable and orderly manner, are called reliable data streams. Reliable data streams are critical to ensure the correctness and completeness of the data in the Member End System. For instance, the Member System may obtain sufficient information through various return messages in the Member’s private stream, so that the Member System could complete its business processing at the Member’s end. In order to guarantee the correctness of business operations in the Member End System, messages in the private stream have to be received in a reliable, orderly and unique manner.

Reliable data stream relies on a re-transmission mechanism to guarantee the reliable and orderly delivery of data. The Member End System is responsible for managing the Sequence ID of the data stream. In case of transmission interruption, the system could re-subscribe to the data stream from a specified Sequence ID to ensure data integrity.

The dialog stream and query stream do not support re-transmission; therefore, they are unreliable streams.

API offers two methods for managing reliable data streams: re-transmitted message serial number managed by the API and re-transmitted message serial number managed by the Member End System.

4.8.1. Re-Transmission Sequence ID Maintained by API

The API periodically writes the sequence ID of received reliable data stream messages to local files ***trade.con*** and ***md.con***. If the Member System re-subscribes data stream after its logout, then the message sequence ID recorded in the local file can be used for subscription of

the data stream. If the Member-end System is unexpectedly disconnected, the sequence ID of the last received message may not have been written to the local file. After re-connection, the same message may be delivered twice to the Member End System, in which case the Member End System should perform de-duplication processing.

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

Subscription methods can be designated via interface parameters, which are classified into three modes: ***RESTART*** (retransmission), ***RESUME*** (resuming of a transmission) and ***QUICK*** (snapshot).

- ***RESTART*** mode starts the re-transmission from the 1st message in the stream, and in this case, the message Sequence ID recorded in the local file is ignored.
- ***RESUME*** mode starts the re-transmission following the Sequence ID recorded in the local file. If it is a market data stream, a snapshot of the thematic market data at that moment will be transmitted first, and then the market data transmission will be started from a specified Sequence ID. In order to maintain the integrity of members' trading data, SHFE recommends the "***RESUME***" mode for the private stream of the member or the trader.
- ***QUICK*** mode starts the re-transmission at the maximum Sequence ID at the moment of subscribing the data stream. If it is a market data stream, the latest market data snapshot of the topic will be transmitted first. The ***QUICK*** mode is mainly used for occasions in which there are no need to guarantee the data integrity, such as quick receiving and resuming of market data after breakdown of communication or software. As for the member's or trader's private stream, SHFE does not recommend the use of ***QUICK*** method.

Note: if a reliable data stream message has been notified to the Member End System through the callback function of SPI, but the corresponding message sequence ID has not been written in the file, the same message will be called back to the Member End System twice.

4.8.2. Re-Transmission Sequence ID Managed by Member End System

Whenever the API receives a message from the reliable data stream, it first calls the ***OnPackageStart*** function of the SPI to inform the Member End System that a message has been received, then calls the callback function of the SPI to notify the Member End System to process the business data, and finally calls the ***OnPackageEnd*** function of the SPI to inform the Member End System that the callback of the message is completed. From the interfaces ***OnPackageStart*** and ***OnPackageEnd***, the Member End System can obtain the Sequence ID of the current callback message, and record the Sequence ID if necessary. When re-transmitting the reliable data stream, the recorded Sequence ID would be used as the parameter for the ***ReqSubscribeTopic*** method (similar to the ***RESUME*** mode).

Via the ***ReqSubscribeTopic*** method, the Member End System can specify the message Sequence ID for data stream re-transmission. If the Sequence ID is 0, the entire data stream would be re-transmitted (similar to ***RESTART*** mode); and if the specified Sequence ID is -1, the message re-transmission would start from the largest Sequence ID at the moment of

subscription (similar to the ***QUICK*** mode).

As for the subscription of the market data stream, if the specified re-transmission Sequence ID is not 0, the market data snapshots before the generation of this Sequence ID message will be transmitted first.

4.9. Heartbeat Mechanism

Heartbeat message is added to check whether the connection is valid or not. If one side does not receive any heartbeat message within a specified timeout period, it could be considered that the TCP virtual link is invalid. In this case, this side should take the initiative to disconnect the link; if one side does not send any business message to the other side within a certain time interval, it should send heartbeat message to the other side to maintain the normal working status of the virtual link.

The API provided the ***SetHeartbeatTimeout*** method to set the timeout period for the Member End System to monitor the validity of the TCP virtual link. The Trading System regularly sends heartbeat messages to the API. If no message is received from the Trading System in more than timeout/2 seconds, the callback function ***OnHeartBeatWarning*** will be triggered; and if no message is received from the Trading System after timeout, TCP connection will be interrupted and the callback function ***OnFrontDisconnected*** will be triggered.

For instance, assuming that the member side sets the heartbeat timeout period to be 16 seconds. If API does not receive any message from the Trading System in 8 seconds, the callback function ***OnHeartBeatWarning*** would be triggered. If no message is received in 16 seconds, API would take the initiative to disconnect the network and trigger the callback function ***OnFrontDisconnected***.

The front-ends of both the Trading System and the Market Data Receiving System monitor the TCP connection of the Member End System via the heartbeat mechanism, and the timeout parameter is also used for the two front-ends to monitor the Member End System. The timeout parameter would be set to 10 seconds by default. The minimum allowable value of the timeout parameter is 4 seconds and the maximum is 181 seconds.

If the timeout parameter is set at a too high level, in the situation of link disruption, a much longer time would be taken for the Member End System to switch to the alternative link; and if the timeout parameter is set at a too low level, unexpected switching might occur. Therefore, the timeout setting requires a comprehensive consideration among the application of the Member End System and the status of the network.

A timeout value of 10-30 seconds is recommended for the Member End System.

4.10. Disaster Recovery Interface

SHFE has built three data centers: Zhangjiang data center, Shanghai Futures Tower data center, and Beijing data center. The three data centers use high-speed optical fiber to connect each other. Zhangjiang data center is the current main data center. The Trading System runs simultaneously at the three data centers: the main center is responsible for business processing, and the backup centers receive data from the main center in real time.

When data center switching occurs, the backup data center takes over the work of the

main data center and continues business processing. During the data center switching, a small amount of the business data might be lost. The Member End System needs to know the Sequence ID of the data stream to be canceled via the API interfaces.

1) The “Data Center ID” field in the API user login request message is used to identify the last logged-in data center. The Trading System returns the currently used data center in the user login response message.
2) The Member End System can obtain the Sequence ID of the data stream to be canceled according to the API “Data Stream Cancellation” (***OnRtnFlowMessageCancel***) interface.
3) The Sequence IDs of the data stream to be canceled include the starting Sequence ID and the ending Sequence ID of the cancellation. The data between the two Sequence IDs is considered invalid data. The Member End System needs to perform cancellation processing on the received data based on the Sequence IDs of the data stream to be canceled. For instance, if the current Member End System requests to subscribe to the data stream starting from Sequence ID 100, and the data stream cancellation notification returned by the Trading System indicates that the starting Sequence ID for cancellation is 95 and the ending Sequence ID is 100, then the Member End System needs to perform a cancellation operation on the data numbered 96 to 100. If the Member End System has the need to subscribe to subsequent data, the Exchange suggests resubscribing starting from the initial Sequence ID 95.

Part II TraderAPI Reference Manual

Part II is designed for Member's system developers, including:
Chapter 1 is the categories of ***TraderAPI*** interfaces.
Chapter 2 is the description of ***TraderAPI*** interfaces.
Chapter 3 is a development instance of ***TraderAPI*** interfaces.

1. Categories of TraderAPI Interfaces

1.1. Management Interfaces

TraderAPI management interfaces control the life cycle and operating parameter of API.

Interface Type	Interface Name	Explanation
Lifecycle Management Interfaces	CShfeFtdcTraderApi:: CreateFtdcTraderApi	Create a TraderApi instance
	CShfeFtdcTraderApi:: GetVersion	Gain API version
	CShfeFtdcTraderApi:: Release	Delete the instance of the interface
	CShfeFtdcTraderApi:: Init	Initialization
	CShfeFtdcTraderApi:: Join	Wait for the interface thread to end the run
	CShfeFtdcTraderApi:: GetTradingDay	Get the current trading day
Parameter Management Interfaces	CShfeFtdcTraderApi:: RegisterSpi	Register to callback interface
	CShfeFtdcTraderApi:: RegisterFront	Register to FEP network communication address
	CShfeFtdcTraderApi:: RegisterNameServer	Register to FENS address
	CShfeFtdcTraderApi:: SetHeartbeatTimeout	Set heartbeat timeout
Subscription Interfaces	CShfeFtdcTraderApi:: SubscribePrivateTopic	Subscribe to member private stream
	CShfeFtdcTraderApi:: SubscribePublicTopic	Subscribe to public stream
	CShfeFtdcTraderApi:: SubscribeUserTopic	Subscribe to trader's private stream
Logging interface	CShfeFtdcTraderApi:: OpenRequestLog	Open request log file
	CShfeFtdcTraderApi:: OpenResponseLog	Open response log file
Communication Status Interfaces	CShfeFtdcTraderSpi:: OnFrontConnected	The method is called when communication with the Trading System connection is established.
	CShfeFtdcTraderSpi:: OnFrontDisconnected	This method will be called when communication with the Trading System is disconnected.
	CShfeFtdcTraderSpi:: OnHeartBeatWarning	The method is called when no heartbeat message is received after a long time.
	CShfeFtdcTraderSpi:: OnPackageStart	Notification for start of message callback
	CShfeFtdcTraderSpi:: OnPackageEnd	Notification for end of the message callback
Disaster Recovery Interfaces	CShfeFtdcTraderSpi:: OnRtnFlowMessageCancel	Notification for data stream cancellation

1.2. Service Interfaces

Service Type	Service	Request Interface / Response Interface	Data Stream
Login	Login	CShfeFtdcTraderApi:: ReqUserLogin	Dialogue

Service Type	Service	Request Interface / Response Interface	Data Stream
		CShfeFtdcTraderSpi:: OnRspUserLogin	Stream
	Logout	CShfeFtdcTraderApi:: ReqUserLogout CShfeFtdcTraderSpi:: OnRspUserLogout	Dialogue Stream
	User Password Update	CShfeFtdcTraderApi:: ReqUserPasswordUpdate	Dialogue Stream
	User Password Update	CShfeFtdcTraderSpi:: OnRspUserPasswordUpdate	Dialogue Stream
	Terminal Authentication	CShfeFtdcTraderApi:: ReqAuthenticate CShfeFtdcTraderSpi:: OnRspAuthenticate	Dialogue Stream
Subscription	Topic/Theme/ Subject Subscription	CShfeFtdcTraderApi:: ReqSubscribeTopic CShfeFtdcTraderSpi:: OnRspSubscribeTopic	Dialogue Stream
	Topic/Theme/ Subject Query	CShfeFtdcTraderApi:: ReqQryTopic CShfeFtdcTraderSpi:: OnRspQryTopic	Query Stream
Trading	Order Entry	CShfeFtdcTraderApi:: ReqOrderInsert CShfeFtdcTraderSpi:: OnRspOrderInsert	Dialogue Stream
	Order Action	CShfeFtdcTraderApi:: ReqOrderAction CShfeFtdcTraderSpi:: OnRspOrderAction	Dialogue Stream
	Price Quotation Entry	CShfeFtdcTraderApi:: ReqQuoteInsert CShfeFtdcTraderSpi:: OnRspQuoteInsert	Dialogue Stream
	Price Quotation Action	CShfeFtdcTraderApi:: ReqQuoteAction CShfeFtdcTraderSpi:: OnRspQuoteAction	Dialogue Stream
	Declaration Entry	CShfeFtdcTraderApi:: ReqExecOrderInsert CShfeFtdcTraderSpi:: OnRspExecOrderInsert	Dialogue Stream
	Declaration Action	CShfeFtdcTraderApi:: ReqExecOrderAction CShfeFtdcTraderSpi:: OnRspExecOrderAction	Dialogue Stream
	Abandon Declaration Entry	CShfeFtdcTraderApi:: ReqAbandonExecOrderInsert CShfeFtdcTraderSpi:: OnRspAbandonExecOrderInsert	Dialogue Stream
	Abandon Declaration Action	CShfeFtdcTraderApi:: ReqAbandonExecOrderAction CShfeFtdcTraderSpi:: OnRspAbandonExecOrderAction	Dialogue Stream
	Quote Demand Entry	CShfeFtdcTraderApi:: ReqQuoteDemand CShfeFtdcTraderSpi:: OnRspQuoteDemand CShfeFtdcTraderSpi:: OnRtnQuoteDemandNotify	Dialogue Stream
	Option Self-Hedging Update	CShfeFtdcTraderApi:: ReqOptionSelfCloseUpdate CShfeFtdcTraderSpi:: OnRspOptionSelfCloseUpdate	Dialogue Stream
	Option Self-Hedging Action	CShfeFtdcTraderApi:: ReqOptionSelfCloseAction CShfeFtdcTraderSpi:: OnRspOptionSelfCloseAction	Dialogue Stream
Private	Trade Return	CShfeFtdcTraderSpi:: OnRtnTrade	Private

Service Type	Service	Request Interface / Response Interface	Data Stream
Return			Stream
	Order Return	CShfeFtdcTraderSpi:: OnRtnOrder	Private Stream
	Price Quotation Return	CShfeFtdcTraderSpi:: OnRtnQuote	Private Stream
	Order Execution Return	CShfeFtdcTraderSpi:: OnRtnExecOrder	Private Stream
	Order Entry Error Return	CShfeFtdcTraderSpi:: OnErrRtnOrderInsert	Private Stream
	Order Action Error Return	CShfeFtdcTraderSpi:: OnErrRtnOrderAction	Private Stream
	Price Quotation Entry Error Return	CShfeFtdcTraderSpi:: OnErrRtnQuoteInsert	Private Stream
	Price Quotation Action Error Return	CShfeFtdcTraderSpi:: OnErrRtnQuoteAction	Private Stream
	Declaration Entry Error Return	CShfeFtdcTraderSpi:: OnErrRtnExecOrderInsert	Private Stream
	Declaration Action Error Return	CShfeFtdcTraderSpi:: OnErrRtnExecOrderAction	Private Stream
	Abandon Declaration Return	CShfeFtdcTraderSpi:: OnRtnAbandonExecOrder	Private Stream
	Abandon Declaration Entry Error Return	CShfeFtdcTraderSpi:: OnErrRtnAbandonExecOrderInsert	Private Stream
	Abandon Declaration Action Error Return	CShfeFtdcTraderSpi:: OnErrRtnAbandonExecOrderAction	Private Stream
	Option Self-Hedging Update Return	CShfeFtdcTraderSpi:: OnRtnOptionSelfCloseUpdate	Private Stream
	Option Self-Hedging Update Error Return	CShfeFtdcTraderSpi:: OnErrRtnOptionSelfCloseUpdate	Private Stream

Service Type	Service	Request Interface / Response Interface	Data Stream
	Option Self-Hedging Action Error Return	CShfeFtdcTraderSpi:: OnErrRtnOptionSelfCloseAction	Private Stream
Public Notification	Contract/Instrument Trading Status Notification	CShfeFtdcTraderSpi:: OnRtnInstrumentStatus	Public Stream
	Instrument Addition Notification	CShfeFtdcTraderSpi:: OnRtnInsInstrument	Public Stream
	Bulletin Notification	CShfeFtdcTraderSpi:: OnRtnBulletin	Public Stream
Query	Fund Query	CShfeFtdcTraderApi:: ReqQryPartAccount CShfeFtdcTraderSpi:: OnRspQryPartAccount	Query Stream
	Order Query	CShfeFtdcTraderApi:: ReqQryOrder CShfeFtdcTraderSpi:: OnRspQryOrder	Query Stream
	Price Quotation Query	CShfeFtdcTraderApi:: ReqQryQuote CShfeFtdcTraderSpi:: OnRspQryQuote	Query Stream
	Trade Query (i.e.filled/matched order)	CShfeFtdcTraderApi:: ReqQryTrade CShfeFtdcTraderSpi:: OnRspQryTrade	Query Stream
	Client Query	CShfeFtdcTraderApi:: ReqQryClient CShfeFtdcTraderSpi:: OnRspQryClient	Query Stream
	Member Holding Position Query	CShfeFtdcTraderApi:: ReqQryPartPosition CShfeFtdcTraderSpi:: OnRspQryPartPosition	Query Stream
	Client Holding Position Query	CShfeFtdcTraderApi:: ReqQryClientPosition CShfeFtdcTraderSpi:: OnRspQryClientPosition	Query Stream
	Instrument/Contract Query	CShfeFtdcTraderApi:: ReqQryInstrument CShfeFtdcTraderSpi:: OnRspQryInstrument	Query Stream
	Instrument/Contract Trading 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
	Bulletin Query	CShfeFtdcTraderApi:: ReqQryBulletin CShfeFtdcTraderSpi:: OnRspQryBulletin	Query Stream
	Order Execution Query	CShfeFtdcTraderApi:: ReqQryExecOrder CShfeFtdcTraderSpi:: OnRspQryExecOrder	Query Stream

Service Type	Service	Request Interface / Response Interface	Data Stream
	Exchange Rate Query	CShfeFtdcTraderApi:: ReqQryExchangeRate CShfeFtdcTraderSpi:: OnRspQryExchangeRate	Query Stream
	Abandon Declaration Query	CShfeFtdcTraderApi:: ReqQryAbandonExecOrder CShfeFtdcTraderSpi:: OnRspQryAbandonExecOrder	Query Stream
	Option Self-Hedging Query	CShfeFtdcTraderApi:: ReqQryOptionSelfClose CShfeFtdcTraderSpi:: OnRspQryOptionSelfClose	Query Stream
Error Response	Error Response	CShfeFtdcTraderSpi:: OnRspError	Dialogue Stream Query Stream

2. TraderAPI Interface Description

2.1. CShfeFtdcTraderSpiInterface

CShfeFtdcTraderSpi implements event notification interface. User has to derive the ***CShfeFtdcTraderSpi*** interface, and writes event-handling methods to process the required events.

2.1.1. OnFrontConnected Method

After the TCP virtual link path connection between Member System and the NGES Trading System is established, the method is called. The mentioned connection is automatically established by the API.

Function Prototype:

```
void OnFrontConnected();
```

Note: The fact that the OnFrontConnected is called only implies that the TCP connection is successful; user must log in to the Member System by himself/herself to carry out any business operations afterwards.

2.1.2. OnFrontDisconnected Method

After the TCP virtual link path connection between Member System and the NGES Trading System is broken, the method is called. In this case, API would automatically reconnect, and the automatically re-connected address may be the originally registered address or other available communication addresses that are supported by the system, which is decided by the application.

Function Prototype:

```
void OnFrontDisconnected(int nReason);
```

Parameters:

nReason: disconnection reasons

- 0x1001 network reading failed
- 0x1002 network writing failure
- 0x2001 heartbeat receiving timeout
- 0X2002 message encryption failed
- 0X2003 message decryption failed
- 0x2004 the message of an unsubscribed topic has been received
- 0X2005 the received message serial number is discontinuous
- 0x2006 the length of the message is illegal
- 0x2007 message conversion error
- 0X2008 login failure (front server)

2.1.3. OnHeartBeatWarning Method

This is for heartbeat timeout warning. This method will be called when no message is received for an extended period of time. By default, the timeout warning is triggered after 5 seconds. If the method ***SetHeartbeatTimeout*** (unsigned int timeout) has been called to set a custom heartbeat timeout, the warning will be triggered at half of the specified timeout (i.e., timeout / 2).

Function Prototype:

```
void OnHeartBeatWarning(int nTimeLapse);
```

Parameters:

nTimeLapse: time lapse from last time receiving the message (in seconds).

2.1.4. OnPackageStart Method

This is the method for notification of start of message/packets callback. When the API receives a message, this method will be called if the message belongs to the public stream or a private stream (either the member private stream or the trader private stream). After this method will be called, callbacks for each data field are triggered, followed by a notification indicating the end of the message callback process.

Function Prototype:

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

Parameters:

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

nSequenceNo: Message Sequence Number

2.1.5. OnPackageEnd Method

This is the notification for end of message/packets callback. After the API receives a message, if the message belongs to the public stream, the private stream (member private stream, trader private stream) calls the message callback to start notification, then the callback of each data field, and finally calls this method.

Function Prototype:

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

Parameters:

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

nSequenceNo: Message Sequence Number.

2.1.6. OnRspUserLogin Method

After Member System sends out a login request, and when the Trading System sends back the response, the method is called to inform the Member System whether the login is successful.

Function Prototype:

```
void OnRspUserLogin(
        CShfeFtdcRspUserLoginField* pRspUserLogin,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspUserLogin: returns the address of user login information structure.

The structure:

```
struct CShfeFtdcRspUserLoginField {
    ///TradingDay
    TShfeFtdcDateType  TradingDay;
    ///Successful login time
    TShfeFtdcTimeType  LoginTime;
    ///Maximum order's local ID
    TShfeFtdcOrderLocalIDType  MaxOrderLocalID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Name of Trading System
    TShfeFtdcTradingSystemNameType  TradingSystemName;
    ///Data Center ID
    TShfeFtdcDataCenterIDType  DataCenterID;
    ///Current size of member's private stream
    TShfeFtdcSequenceNoType  PrivateFlowSize;
    ///Current size of private stream of trader/user
    TShfeFtdcSequenceNoType  UserFlowSize;
    ///action day
    TShfeFtdcDateType  ActionDay;
};
```

Note: When retrieving the date on which the business action occurred, use the ActionDay field; the same applies to all similar cases below.

pRspInfo: returns the address of user response information. Error ID 0 means successful operation; this is the same as below. Response information/message structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
3	Member not found	Login failed due to an incorrect member ID
45	Settlement Group not properly initialized	Trading System initialization incomplete, please try again later
59	Same user logged in	Same user logged in multiple times from different IP addresses
60	Username or password incorrect	Invalid username or password
62	User inactive	The Trading System does not permit login for this user
64	User does not belong to this member	The member ID of login is wrong

65	Invalid login IP address	Login attempt from an IP address not authorized by the Exchange
75	Front-End inactive	Trading System Front-End Inactive
100	Invalid user type	Non-Trading user attempting to log in
106	Duplicate session	Multiple logins with the same session
135	User authentication failed	User key verification failed
136	User does not have permission for direct Front-End connection	User unauthorized for direct Front-End access
150	Proprietary member not authenticated or authentication failed before login	Proprietary member terminal information not authenticated

nRequestID: returns the user login request ID; this ID is specified by the user upon login.

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

2.1.7. OnRspUserLogout Method

After Member System sends out logout request, the Trading System calls this method to send back the response to inform the Member System whether logout is successful.

Function Prototype:

```
void OnRspUserLogout(
        CShfeFtdcRspUserLogoutField* pRspUserLogout,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspUserLogout: returns the address of user logout information/message. User logout information/message structure:

```
struct CShfeFtdcRspUserLogoutField {
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
};
```

pRspInfo: returns the address of user response information/message. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
66	User not logged in	User not logged in
67	User not logged in with this account	Logout Attempt by a different user than the one logged in
68	Member Not Logged In with This	Logout Attempt by a Different Member than the One

Account	Logged In

nRequestID: returns the user logout request ID; this ID is specified by the user upon logout.

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

2.1.8. OnRspUserPasswordUpdate Method

This method is for the user password change reply. After Member System sends out password update request, the Trading System calls it to send back the response.

Function Prototype:

```
void OnRspUserPasswordUpdate(
        CShfeFtdcUserPasswordUpdateField* pUserPasswordUpdate,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pUserPasswordUpdate: pointer to the user password update structure, including the input data for user password update request. User password update structure:

```
struct CShfeFtdcUserPasswordUpdateField {
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Old password
    TShfeFtdcPasswordType  OldPassword;
    ///New password
    TShfeFtdcPasswordType  NewPassword;
};
```

pRspInfo: pointer to the response information/message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
23	Settlement group data not synchronized	Trading System not fully initialized. Please try again later
58	User mismatch	The user attempting to change the password is different from the Logged-In user
60	Wrong username or Password	Incorrect original password
62	User inactive	User does not have permission to log in, trade, or change Password
66	User not logged in	Not logged In
68	Member not logged in with this account	The member ID for password change does not match the logged-in member

147	New password does not meet requirements (Minimum 8 characters, must include numbers, uppercase and lowercase letters)	New password does not meet the password policy requirements

nRequestID: returns the user password modification request ID; this ID is specified by the user upon password modification

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

2.1.9. OnRspSubscribeTopic Method

This method is for the reply on topic/theme subscription. After Member System sends out topic subscription request, the Trading System calls this method to send back the response.

Function Prototype:

```
void OnRspSubscribeTopic(
        CShfeFtdcDisseminationField* pDissemination,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pDissemination: pointer to the subscription topic structure, including topic subscribed and sequence number of starting message. The structure:

```
struct CShfeFtdcDisseminationField {
    ///Sequence series
    TShfeFtdcSequenceSeriesTypeSequenceSeries;
    ///Sequence number
    TShfeFtdcSequenceNoTypeSequenceNo;
};
```

pRspInfo: pointer to the response information/message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
1	Invalid session or topic does not exist	The subscribed topic does not exist or the number of subscriptions has exceeded the limit

nRequestID: returns the user subscribed topic request ID; this ID is specified by the user upon subscription.

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

2.1.10. OnRspQryTopic Method

This method is for the reply to the query of topic. This method will be called when the

Trading System returns a response after the Member System issues topic query instruction.

Function Prototype:

```
void OnRspQryTopic(
        CShfeFtdcDisseminationField* pDissemination,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pDissemination: pointer to the topic query structure, including the topic to be queried and the number of messages in the topic. The structure:

```
struct CShfeFtdcDisseminationField {
    ///Sequence series
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
};
```

pRspInfo: pointer to the response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
Possible errors: None
```

nRequestID: returns the user topic query request ID; this ID is specified by the user upon querying topics.

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

2.1.11. OnRspError Method

This method is for error notification with respect to user request.

Function Prototype:

```
void OnRspError(
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspInfo: returns the address of response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
Possible errors:
```

Error ID	Error message	Possible cause
1	Not Login	Not logged in yet
	Too High FTD Version	Too high FTD version
	Unrecognized ftd tid	FTD Header Error
134	APIVerification failed	User session authentication failed
151	Version check failed	Trading API version verification failed
997	api authentication failure	Unauthorized API access
	api crypt info failure	Failed to query API encryption Information
998	query frequency is too high	Query Frequency Too High
999	the last query result is on way	Pending Query Response Exists

nRequestID: returns the user operating request ID; this ID is specified by the user upon sending request.

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

2.1.12. OnRspOrderInsert Method

This method is for the reply to the order entry. After Member System sends out order entry instruction, the Trading System calls this method to send back the response.

Function Prototype:

```
void OnRspOrderInsert(
        CShfeFtdcInputOrderField* pInputOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputOrder: pointer to the order insert structure, including input data upon submitting order insert as well as the order ID returned from the Trading System. The tructure:

```
struct CShfeFtdcInputOrderField {
    ///Order System ID*; this field is returned from the Trading System
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Trading user ID
    TShfeFtdcUserIDType  UserID;
    ///Contract ID/Instrument ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Order price type/condition
    TShfeFtdcOrderPriceTypeType  OrderPriceType;
    ///Buy/Sell direction
    TShfeFtdcDirectionType  Direction;
    ///Combination offset flag
    TShfeFtdcCombOffsetFlagType  CombOffsetFlag;
    ///Combination speculation hedge flag
    TShfeFtdcCombHedgeFlagType  CombHedgeFlag;
    ///Price
    TShfeFtdcPriceType  LimitPrice;
    ///Quantity
```

```
        TShfeFtdcVolumeType  VolumeTotalOriginal;
        ///Validity period type
        TShfeFtdcTimeConditionType  TimeCondition;
        ///GTDDate, not used
        TShfeFtdcDateType  GTDDate;
        ///Match volume type
        TShfeFtdcVolumeConditionType  VolumeCondition;
        ///Minimum Volume
        TShfeFtdcVolumeType  MinVolume;
        ///Trigger condition
        TShfeFtdcContingentConditionType  ContingentCondition;
        ///Stop Price, not used
        TShfeFtdcPriceType  StopPrice;
        ///Force close reasons
        TShfeFtdcForceCloseReasonType  ForceCloseReason;
        ///Local order ID
        TShfeFtdcOrderLocalIDType  OrderLocalID;
        ///Automatic suspend flag
        TShfeFtdcBoolType  IsAutoSuspend;
        ///Business unit
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IPAddress
        TShfeFtdcIPAddressType  IPAddress;
        ///MacAddress
        TShfeFtdcMacAddressType  MacAddress;
    };
```

* OrderSysID: This is a sequential identifier generated by the Trading System for both regular orders and quote-generated orders, managed under a unified numbering scheme. The value increases incrementally within the Trading System. The SysIDs used for different business types are managed independently. For instance, the OrderSysID for order placement and the QuoteSysID for quoting are assigned separately and are not correlated.

pRspInfo: pointer to the response information structure. The structure:

```
    struct CShfeFtdcRspInfoField {
        ///Error ID
        TShfeFtdcErrorIDType  ErrorID;
        ///Error message
        TShfeFtdcErrorMsgType  ErrorMsg;
    };
```

Possible errors:

Error ID	Error message	Possible cause
2	Unable to find instrument	Unable to find the instrument in the order
3	Unable to find the Member	Unable to find the Member in the order
4	Unable to find the client	Unable to find the client in the order
6	Fields error in the order	The order contains invalid field values (e.g., an out-of-range enumerated value), or a forced liquidation reason is specified for a non-forced-liquidation order when the order quantity is not an integer multiple of the required amount
12	Duplicated orders	Local ID in the order is duplicated
15	Client has no account under the exchange member	The client in the order has no account under the specified member
16	IOC has to be with the continuous trading session	IOC (immediately-or-cancel) order is tried to be entered at non-continuous trading session

17	GFA has to be with the auction session	GFA order is tried to be entered at non-auction session
19	Volume restriction should be with IOC order	The order whose volume restriction is not arbitrary does not have the IOC time condition
20	GTD order is expired	The GTD date in the GTD order is expired
21	The minimum volume is greater than the order volume	The order has minimum volume condition, but the order volume is less than this minimum volume
22	Exchange data is not synchronized	The Trading System is not completely initialized, try later
23	Settlement group data is not synchronized	Initialization of the Trading System is incomplete, try later
26	This operation is forbidden under current status	The trading status of the instrument is not continuous-trading or auction or auction balance
31	Insufficient client position	The client does not have enough position to place the close order
32	Client's position limit is exceeded	When submitting an open position order, the client's general position limit for the specified contract is exceeded
33	Member holding positions is not enough when close position	The submitted close order exceeds the member's available position
34	Member's position limit is exceeded	When entering open position order, the member's limit position is exceeded
35	Unable to find account	Unable to find the fund account used in the order
36	Fund not enough	There is not enough fund in the fund account
37	Invalid quantity	The order quantity is not a positive multiple of the minimum order quantity or exceeds the maximum
48	Price is not an integer multiple of the minimum unit	Order price is not an integer multiple of the minimum variable price unit
49	Price exceeds limit up	Order price exceed the upper limit of the contract
50	Price falls below limit down	Order price lower than the lower limit of the instrument
51	No trading permission	The member, client, or user does not have trading permission for the specified contract
52	Close position only	The member, client, or user is only authorized to close positions for the specified contract
53	No such trading role	Member, client or trader have no rights to trade specified contract
54	Session not found	User not logged in
57	Cannot operate for other members	The user operates on a member not to whom he belongs
58	User not match	The user in the quote does not match the user when logging in
72	Natural persons are not allowed to open positions	A client of the natural person type initiates a position opening request in the delivery month
78	GTD date not set in the GTD order	GTD order does not specify the GTD date
79	Order type not supported	SHFE does not support this type of order
83	Stop loss order is only used for continuous trading	Stop loss order is entered in non-continuous trading session
84	Stop loss order has to be IOC/GFD	Time condition is neither IOC nor GFD at stop loss order
95	Stop loss order should specify stop price	Stop loss order does not specify a stop price
96	Hedging amount not enough	When entering hedging order, client hedge amount is not enough
98	Forced-liquidation orders must be submitted by administrator-level users	The current user does not have the required permissions to submit a forced-liquidation order

101	Clearing members are not permitted to place orders	The submitting member is classified as a clearing member and therefore cannot trade
102	Unable to locate the corresponding clearing member	No clearing member is associated with the submitting member
103	Hedging position unable to close within the same day	Hedging position should not use close-today-position order to close the position
114	Best price order unable to queue	Best price order time condition is not IOC
131	The client's open position for the specified contract has exceeded the daily open limit	The client's number of order submissions for the specified product has exceeded the per-second order flow control
132	Exceeded the order limit per second for client products	The number of orders submitted by clients on a certain product within one second exceeds the limit
153	Market orders must use a time-in-force condition of GFD or IOC	The market order time condition is not
154	Market orders can only be submitted during the continuous trading session	A market order was submitted outside the continuous trading phase
155	Market orders are only supported for futures and options contracts	A market order was submitted for a non-futures-or-options contract
1005	No record	No contract record found for the order

nRequestID: returns the user order insertion request; this ID is specified by the user upon submitting the order.

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

Note:

CShfeFtdcRspInfoField.ErrorID is 0 implies that current order entry is successful. In ***CShfeFtdcInputOrderField* pInputOrderonly*** order ID (the system ID given by the Trading System) and local order ID are meaningful, which are used to relate the order between the Trading System and Member System. The detailed content of the order should be obtained from private stream.

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

2.1.13. OnRspOrderAction Method

This method is used to response to order operations, which includes order cancellation, order suspension, order activation and order modification. When Member System sent an order for order operation and Trading System needs to return a response, this method will be called.

Function Prototype:

```
void OnRspOrderAction(
        CShfeFtdcOrderActionField* pOrderAction,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pOrderAction: pointer to the order operation structure, including the input data when an order in submitted as well as the order number returned from the Trading System. Order

operation structure:

```
struct CShfeFtdcOrderActionField {
    ///Order number
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Local Order number
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of Order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member's ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client's ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Price, not used
    TShfeFtdcPriceType  LimitPrice;
    ///Quantity change, not used
    TShfeFtdcVolumeType  VolumeChange;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID
    ///Business unit, not used
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP Address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac Address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot not be found	Contract cannot be found in order operation
3	Member cannot not be found	Member cannot be found in the order operation
4	Client cannot be found	Client cannot be found in the order operation
8	Error field in the order operation	Illegal field values in the order operation (out-of-range of the enumerated value)
15	Client didn't open an account at this member	Client didn't open an account at the designated member
16	IOC orders must be placed during the continuous trading session	Attempted to operate an IOC order outside the continuous trading session
17	GFA orders must be submitted during the call auction session	Attempted to operate a GFA order outside the call auction session
20	The GTD order has expired	The GTD date specified in the GTD order has already expired
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later

23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
24	Order cannot be found	Order to be operated cannot be found
26	This operation is prohibited by current state	As for activation of operation, the contract's trading status is not the continuous trade, call auction order or call auction balancing As for other operation, the trading status is not the continuous trade or call auction order
28	The order has been fully filled	Order has already been fulfilled
29	The order has been canceled	Order has already been canceled
30	Insufficient quantity for modification	After modifying the order quantity, the remaining order quantity is less than zero
31	Insufficient client position	The client does not have enough position to place the close order
32	Exceeding client's position limit	The order cannot be activated because it exceeds the client's general position limit
33	The member does not hold a sufficient position to close	The member has insufficient position to place the close order
34	Exceeding member's position limit	The order cannot be activated because it exceeds the member's position limit
35	Account cannot be found	The fund account shall be used cannot be found
36	Insufficient fund	No sufficient funds in fund account
37	Invalid quantity	The modified order quantity is either not a positive integer multiple of the minimum order quantity, or it falls outside the valid quantity range
48	The price is not the integral multiple of the Min. unit	Price of order after modification is not the integral multiple of the contract's tick size
49	Price exceeds the upward limit	Price of order after modification is higher than the contract's upward price limit
50	Price exceeds the downward limit	Price of order after modification is lower than the contract's downward price limit
51	No trading permissions	The specified contract, the client for the specified contract, or the user does not have trading permission
52	Only closing positions is permitted	The member, the client for the specified contract, or the user only has permission to close positions
54	Session not found	User not logged in
57	Cannot operate for other members	The user is not authorized to operate on behalf of other members
58	Unmatched user	Trader in the order operation doesn't match with trader at the time of login
71	Operations on derivative orders are not allowed	The user attempted to operate on a derivative order
72	Natural person clients are not allowed to open positions	During the delivery month, natural person clients cannot activate or modify opening orders
76	The order has been suspended	The order has already been suspended when attempting to suspend it
77	The order has been activated	The order has already been activated when attempting to activate it
79	Unsupported order type	The exchange does not support this order type, for instance: order modification
83	Stop-loss orders can only be used during the continuous trading session	Attempted to operate a stop-loss order outside the continuous trading session
95	A stop-loss price must be specified for the stop-loss order	The stop-loss order does not contain a specified stop-loss price

96	Insufficient hedging quota	After activation or modification, the client's hedging quota is insufficient
98	Forced liquidation orders must be operated by an administrator	A non-administrator attempted to operate a forced liquidation order
99	Operation shall not be conducted by other users	Unauthorized trader operates order submitted by other traders of the same member
131	The client's open volume for the contract exceeds the daily limit	The client's open position in a specific contract has exceeded the daily open position limit
133	The client has exceeded the per-second order cancellation limit for the product	The client's number of order cancellations within one second for a specific product exceeds the allowed limit

nRequestID: returns the user order operation request ID; this ID is specified by the user upon order operating.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.14. OnRspQuoteInsert Method

This method is used to response to quote entry. When Member System gives the instructions for entry of order and Trading System returns a response, this method will be called.

Function Prototype:

```
void OnRspQuoteInsert(
        CShfeFtdcInputQuoteField* pInputQuote,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputQuote: pointer to the input quote structure, including the input data of quote entry operation and the quoto number returned from Trading System. The input quote structure:

```
struct CShfeFtdcInputQuoteField {
    ///Quotation number
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Quantity
    TShfeFtdcVolumeType  Volume;
    ///Contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    //Buyer's combination offset flag
    TShfeFtdcCombOffsetFlagType  BidCombOffsetFlag;
```

```
    ///Buyer's combination hedge flag
    TShfeFtdcCombHedgeFlagType  BidCombHedgeFlag;
    ///Buyer's price
    TShfeFtdcPriceType  BidPrice;
    ///Seller's combination offset flag
    TShfeFtdcCombOffsetFlagType  AskCombOffsetFlag;
    ///Seller's combination hedge flag
    TShfeFtdcCombHedgeFlagType  AskCombHedgeFlag;
    ///Seller's price
    TShfeFtdcPriceType  AskPrice;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
    ///Quote request number
    TShfeFtdcOrderSysIDType  QuoteDemandID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract cannot be found in the quote
3	Member cannot be found	Member cannot be found in the quote
4	Client cannot be found	Client cannot be found in the quote
7	Error field in the quote	Illegal field values in the quote (out-of-range of the enumerated value)
13	Duplicate Quote	Duplicate local quoto number in the quote
15	Client didn't open an account at this member	Client in the quote didn't open an account at the designated member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract's trading status is not the continuous trade, call auction order or call auction balancing
31	Insufficient client position	The client does not have enough position to place the close order
32	Exceeding client's position limit	The quote causes the client's general position to exceed the position limit
33	The member's position is insufficient at the time of closing-out	The member does not have enough position
34	Exceeding member's position limit	This quote caused the member's open interest exceeding position limit
35	Account cannot be found	The fund account used for quotation cannot be found
36	Inadequate fund	No sufficient funds in fund account

37	Invalid quantity	The quote quantity is not a positive multiple of the minimum order quantity or exceeds the maximum
48	A multiple of a non-smallest unit of price	The quoted price is not the integral multiple of the contract's tick size
49	Price exceeds the upward limit	The quoted price is higher than the contract's upward price limit
50	Price exceeds the downward limit	The quoted price is lower than the contract's downward price limit
51	Not authorized to trade	Not authorized to trade in the designated contract, or client or user is not authorized to trade in the designated contract
52	Only closing positions is permitted	The member, the client for the specified contract, or the user only has permission to close positions
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
54	Session not found	User not logged in
57	Operation shall not be conducted by other members	User conducts operation on behalf of member to whom he is not subordinate
58	Unmatched user	User in the quote doesn't match with user at the time of login
72	Natural persons are not allowed to open positions	A client of the natural person type initiates a position opening request in the delivery month
79	Unsupported quote type	The Exchange does not support this order type
96	Insufficient hedging quota	When entering the quotation, the client's hedging amount is insufficient
98	Forced liquidation orders must be submitted by an administrator	A non-administrator user attempted to place a forced liquidation order
101	Clearing members cannot make transactions	The quoting member is of clearing member type
102	Failed to locate the corresponding clearing member	No clearing member associated with the quoting member could be found
103	Today's hedging positions cannot be closed	Hedging positions should not be closed using the current position quotation
131	The client's opening volume for the contract exceeds the daily limit	The client's open position in a specific contract exceeds the allowed limit
132	The client's order submission rate for the product exceeds the per-second limit	The number of orders submitted by the client for a product within one second exceeds the limit
1005	No record found	Contract record corresponding to quote is missing

nRequestID: returns the user quote entry operation request; this ID is specified by the user upon quote entry.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.15. OnRspQuoteAction Method

This function is used to response to quote operation, including cancellation of quote, suspension of quote, activation of quote and modification to quote. When Member System gives the instructions for quote operation and Trading System returns a response, this method will be called.

Function Prototype:

```
void OnRspQuoteAction(
        CShfeFtdcQuoteActionField* pQuoteAction,
```

```
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pQuoteAction: pointer to the quote operation structure, including the input data of request for quote operation and quoto number returned from Trading System. Quote operation structure:

```
struct CShfeFtdcQuoteActionField {
    ///Quoto number
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    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: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract specified in the quote operation cannot be found
3	Member cannot be found	Member specified in the quote operation cannot be found
4	Client cannot be found	Client specified in the quote operation cannot be found
8	Error field in the quote operation	The derived order from the quote operation contains invalid field values (e.g., price is not a floating-point number or is outside the valid range)
9	Error field in the quote operation	The quote operation contains invalid field values (e.g., out-of-range enumeration values or unsupported

		operation flags such as modify, activate, or suspend)
15	Client didn't open an account at this member	Client didn't open an account at the designated member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
25	Quote cannot be found	Quote to be operated cannot be found
26	This operation is prohibited by current state	As for activation of operation, the contract's trading status is not the continuous trade, call auction order or call auction balancing As for other operations, the trading status is not the continuous trade or call auction order
28	Order has been fully filled	Order derived from quote has already been fulfilled
29	The order has been canceled	Order derived from the quote has already been canceled
35	Account cannot be found	The fund account shall be used cannot be found
36	Insufficient fund	No sufficient funds in fund account
51	No trading permission	No trading permission for the specified contract, client, or user
54	Session not found	User not logged in
57	Operation on behalf of another member is not allowed	User attempted to operate on a member they do not belong to
58	Unmatched user	The user in the quote operation does not match the user at login
70	Quote has already been canceled	Quote has already been canceled
99	Operation on behalf of another user is not permitted	Unauthorized user attempted to operate on a quote submitted by another user under the same member

nRequestID: returns the user quote operation request ID; this ID is specified by the user upon quote operation.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.16. OnRspExecOrderInsert Method

This method is used to response to option exercise entry. When Member System executed the entry of declaration and Trading System returned a response, this method will be called.

Function Prototype:

```
void OnRspExecOrderInsert(
        CShfeFtdcInputExecOrderField* pInputExecOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputExecOrder: pointer to the declaration entry structure. The structure of option exercise entry:

```
struct CShfeFtdcInputExecOrderField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
```

```
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local option exercise number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Quantity
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///position direction, i.e. whether buyer(long position) or seller (short position)
    made this application
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///flag for whether position is reserved after option exercrised, not used
    TShfeFtdcExecOrderPositionFlagType  ReservePositionFlag;
    ///flag for whether position is closed automatically after option exercrised
    TShfeFtdcExecOrderCloseFlagType  CloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP Address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac Address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract cannot be found in the option exercise
3	Member cannot be found	Member cannot be found in the option exercise
4	Client cannot be found	Client cannot be found in the option exercise
15	Client didn't open an account at this member	Client in the option exercise didn't open an account at the designated member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract is not in continuous trading or business processing status
31	Insufficient client positions at closing	The client has insufficient position quota for execution of declaration submission
33	Insufficient member positions at closing	The member has insufficient position quota for execution of declaration submission
35	Account cannot be found	The required fund account

36	Insufficient funding	No trading permission for the specified contract, client, or user found
37	Invalid quantity	Invalid quantity in option exercise
51	No trading permission	No trading permission for the specified contract, client, or user
54	Session not found	User not logged in
57	Operation on behalf of another member is not allowed	The user conducts operation on behalf of member to whom he is not subordinate
58	Unmatched user	User in the option exercise does not match user at the time of login
79	Unsupported order type	This order type is not supported by the exchange
89	Error field in the execution of declaration operation	Illegal field values in the execution of declaration operation (out-of-range of the enumerated value)
91	Duplicate option exercise	The local announcement execution number in option exercise is not unique
94	Option exercise is only used in option	The contract in option exercise is non-option contract
101	Clearing members cannot make transactions	The member in the execution of declaration is a clearing member
102	Corresponding clearing member not found	No clearing member associated with the execution of declaration member could be found
127	Not within the declaration period	Not within the contract's delivery period (exercise window)
129	Execution of declarations must not use the open position flag	The offset flag in the execution of declaration must indicate closing
146	Only holders of long positions are allowed to exercise	Only option buyers are allowed to exercise
1005	No record	The contract record referenced in the execution of declaration is missing

nRequestID: returns the user option exercise entry request ID; this ID is specified by the user upon option exercise entry.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.17. OnRspExecOrderAction Method

Response to execution of announcement operation. Execution of declaration operations include cancellation, suspension, activation, and modification of execution of declarations. When Member System executes the declaration operation and Trading System returns a response, this method will be called.

Function Prototype:

```
void OnRspExecOrderAction(
        CShfeFtdcExecOrderActionField* pExecOrderAction,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pExecOrderAction: pointer to the option exercise operation structure. The structure:

```
struct CShfeFtdcExecOrderActionField {
    ///Option exercise number
    TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
```

```
    ///Local execution announcement number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Order operation flag
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP Address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac Address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract cannot be found in the option exercise
3	Member cannot be found	Member cannot be found in the option exercise
4	Client cannot be found	Client cannot be found in the option exercise
15	The client has not opened an account with this member	Client in the in the option exercise didn't open an account at the designated member
22	Exchange data is not synchronized	Initialization of Trading System is not completed, please try later
23	Settlement group data is not synchronized	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract is not in continuous trading or business processing status
35	Account not found	The required fund account cannot be found
36	Insufficient funds	Insufficient funds in the fund account
51	No trading permission	No trading permission for the specified contract, the client under the contract, or the user
54	Session not found	User not logged in
57	Operation on behalf of another member is not allowed	The user conducts operation on behalf of member to whom he is not subordinate
58	User mismatch	The user in the execution announcement operation does not match the user at login
89	Field error in the execution of declaration	The execution of declaration contains invalid field values
90	Field error in the execution of	Illegal field values in the execution of declaration

	declaration operation	operation (out-of-range of the enumerated value)
92	The execution of declaration has been canceled	The declaration operation to be executed has been canceled
93	Execution of declaration cannot be found	The option exercise to be operated can not be found
127	Not within the declaration period	Not within the contract delivery period (exercise window)
1005	No record	The contract record corresponding to the declaration operation is missing

nRequestID: returns the user declaration operation execution request ID; this ID is specified by the user upon execution of declaration operation.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.18. OnRspQryPartAccount Method

This method is the response to query for member' funds. When Member System gives the instructions to query for member's funds and Trading System returns a response, this method will be called.

Function Prototype:

```
void OnRspQryPartAccount(
        CShfeFtdcRspPartAccountField* pRspPartAccount,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspPartAccount: pointer to the structure of response to member's funds. The structure:

```
Structure CShfeFtdcRspPartAccountField {
    ///Trading day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Last settlement reserve
    TShfeFtdcMoneyType  PreBalance;
    ///Total current margin
    TShfeFtdcMoneyType  CurrMargin;
    ///Profit & loss on closing-out of position
    TShfeFtdcMoneyType  CloseProfit;
    ///Option premium income and expenditure
    TShfeFtdcMoneyType  Premium;
    ///Deposit Amount
    TShfeFtdcMoneyType  Deposit;
    ///Withdrawal amount
    TShfeFtdcMoneyType  Withdraw;
    ///Reserve funds for futures settlement
    TShfeFtdcMoneyType  Balance;
    ///Withdrawable funds
    TShfeFtdcMoneyType  Available;
    ///Fund account
```

```
    TShfeFtdcAccountIDType  AccountID;
    ///Frozen margin
    TShfeFtdcMoneyType  FrozenMargin;
    ///Frozen premium
    TShfeFtdcMoneyType  FrozenPremium;
    ///Basic reserve funds
    TShfeFtdcMoneyType  BaseReserve;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session not found	User not logged in
57	Operation on behalf of another member is not allowed	Querying data under other members is not permitted
80	User does not have this permission	Only trading users are authorized to perform queries Queries are limited to a single member account

nRequestID: returns the user request ID for user's query for funds; this ID is specified by the user upon sending query instruction.

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

2.1.19. OnRspQryOrder Method

This method is for order query request. After Member System sends out order query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryOrder(
        CShfeFtdcOrderField* pOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pOrder: pointer to the order information/message structure. The structure:

```
struct CshfeFtdcOrderField {
    ///Business day
    TshfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Order ID
    TShfeFtdcOrderSysIDType  OrderSysID;
```

```
///Member ID
TShfeFtdcParticipantIDType  ParticipantID;
///Client ID
TShfeFtdcClientIDType  ClientID;
///Transaction user's ID
TShfeFtdcUserIDType  UserID;
///Instrument/contract ID
TShfeFtdcInstrumentIDType  InstrumentID;
///Order Price Type
TShfeFtdcOrderPriceTypeType  OrderPriceType;
///buy-sell direction
TShfeFtdcDirectionType  Direction;
///Combination offset flag
TShfeFtdcCombOffsetFlagType  CombOffsetFlag;
///Combination Hedge Flag
TShfeFtdcCombHedgeFlagType  CombHedgeFlag;
///Price
TShfeFtdcPriceType  LimitPrice;
///Volume
TShfeFtdcVolumeType  VolumeTotalOriginal;
///Expiry Type
TShfeFtdcTimeConditionType  TimeCondition;
///GTD Date, NOT USED
TShfeFtdcDateType  GTDDate;
///Match volume condition type
TShfeFtdcVolumeConditionType  VolumeCondition;
///Minimum Volume
TShfeFtdcVolumeType  MinVolume;
///Trigger/Contingent Condition
TShfeFtdcContingentConditionType  ContingentCondition;
///Stop loss Price, NOT USED
TShfeFtdcPriceType  StopPrice;
///Forced close reasons
TShfeFtdcForceCloseReasonType  ForceCloseReason;
///Local order ID
TShfeFtdcOrderLocalIDType  OrderLocalID;
///Auto Suspend flag
TShfeFtdcBoolType  IsAutoSuspend;
///Order Source
TShfeFtdcOrderSourceType  OrderSource;
///Order Status
TShfeFtdcOrderStatusType  OrderStatus;
///Order Type
TShfeFtdcOrderTypeType  OrderType;
///Today's trade volume
TShfeFtdcVolumeType  VolumeTraded;
///Remaining volume
TShfeFtdcVolumeType  VolumeTotal;
///order date
TShfeFtdcDateType  InsertDate;
///Entry time
TShfeFtdcTimeType  InsertTime;
///Activation time
TShfeFtdcTimeType  ActiveTime;
```

```
    ///Suspension time
    TShfeFtdcTimeType  SuspendTime;
    ///Last amendment time
    TShfeFtdcTimeType  UpdateTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Last modified trading user ID
    TShfeFtdcUserIDType  ActiveUserID;
    ///Priority
    TShfeFtdcPriorityType  Priority;
    ///Sequence number by time order
    TShfeFtdcTimeSortIDType  TimeSortID;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user request ID for order query; this ID is specified by the user upon sending query instruction.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.20. OnRspQryQuote Method

This function is the response to query for quote. When Member System gives the instructions to query for quote and Trading System returns a response, this method will be called.

Function Prototype:

```
void OnRspQryQuote(
```

```
        CShfeFtdcQuoteField* pQuote,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pQuote: pointer to the quote message structure. The structure:

```
struct CShfeFtdcQuoteField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Quoto number
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Buyer's combination offset flag
    TShfeFtdcCombOffsetFlagType  BidCombOffsetFlag;
    ///Buyer's combination hedge flag
    TShfeFtdcCombHedgeFlagType  BidCombHedgeFlag;
    ///Buyer's price
    TShfeFtdcPriceType  BidPrice;
    ///Seller's combination offset flag
    TShfeFtdcCombOffsetFlagType  AskCombOffsetFlag;
    ///Seller's combination hedge flag
    TShfeFtdcCombHedgeFlagType  AskCombHedgeFlag;
    ///Seller's price
    TShfeFtdcPriceType  AskPrice;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Transaction time
    TShfeFtdcTimeType  TradeTime;
    ///Buyer's order number
    TShfeFtdcOrderSysIDType  BidOrderSysID;
    ///Seller's order number
    TShfeFtdcOrderSysIDType  AskOrderSysID;
    ///Settlement member's number
```

```
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Action day
    TShfeFtdcDateType ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
    ///Quote request ID
    TShfeFtdcOrderSysIDType  QuoteDemandID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user request ID for quote request; this ID is specified by the user upon query for quote.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.21. OnRspQryTrade Method

This method is for the reply on matched order/ trade query. After Member System sends out matched order (i.e. trade) query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryTrade(
        CShfeFtdcTradeField* pTrade,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pTrade: pointer to the matched order information structure. The structure:

```
struct CShfeFtdcTradeField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
```

```
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Matched order ID
    TShfeFtdcTradeIDType  TradeID;
    ///buy-sell direction
    TShfeFtdcDirectionType  Direction;
    ///Order ID
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Trading Role
    TShfeFtdcTradingRoleType  TradingRole;
    ///Fund account
    TShfeFtdcAccountIDType  AccountID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Price
    TShfeFtdcPriceType  Price;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Transaction time
    TShfeFtdcTimeType  TradeTime;
    ///Trade Type / order matching type
    TShfeFtdcTradeTypeType  TradeType;
    ///Trade Price Source / Order Matching Price Source
    TShfeFtdcPriceSourceType  PriceSource;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local order ID
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user request ID for matched order query; this ID is specified by the user upon sending fund query instruction.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.22. OnRspQryClient Method

This method is for the reply on member client query. After Member System sends out client query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryClient(
        CShfeFtdcRspClientField* pRspClient,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspClient: pointer to the client information/message structure. The structure:

```
struct CShfeFtdcRspClientField {
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Client name
    TshfeFtdcPartyNameType  ClientName;
    ///ID Type
    TShfeFtdcIdCardTypeType  IdentifiedCardType;
    ///Original ID
    TShfeFtdcIdentifiedCardNoV1Type  UseLess;
    ///Trading Role
    TShfeFtdcTradingRoleType  TradingRole;
    ///Client type
    TShfeFtdcClientTypeType  ClientType;
    ///Active or not flag
    TShfeFtdcBoolType  IsActive;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///ID Number
    TShfeFtdcIdentifiedCardNoType  IdentifiedCardNo;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
```

```
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the request ID of the member-client query; this ID is specified by the user upon performing the member-client query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.23. OnRspQryPartPosition Method

This method is for the reply on member holding position query. After Member System sends out member holding position query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryPartPosition(
        CShfeFtdcRspPartPositionField* pRspPartPosition,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspPartPosition: pointer to the member holding position response information/message structure. The structure:

```
struct CShfeFtdcRspPartPositionField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Holding position over-under direction
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Previous day holding position
    TShfeFtdcVolumeType  YdPosition;
    ///Current day holding position
    TShfeFtdcVolumeType  Position;
    ///Long frozen
    TShfeFtdcVolumeType  LongFrozen;
    ///Short frozen
    TShfeFtdcVolumeType  ShortFrozen;
    ///Previous day long frozen
    TShfeFtdcVolumeType  YdLongFrozen;
    ///Previous day short frozen
    TShfeFtdcVolumeType  YdShortFrozen;
```

```
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Trading Role
    TShfeFtdcTradingRoleType  TradingRole;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the request ID of the member position query; this ID is specified by the user upon performing the member position query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.24. OnRspQryClientPosition Method

This method is for the reply on client holding position query. After Member System sends out client holding position query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryClientPosition(
        CShfeFtdcRspClientPositionField* pRspClientPosition,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspClientPosition: pointer to the member holding position response information/message structure. The structure:

```
struct CShfeFtdcRspClientPositionField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
```

```
    ///Holding position over-under direction
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Previous day holding position
    TShfeFtdcVolumeType  YdPosition;
    ///Current day holding position
    TShfeFtdcVolumeType  Position;
    ///Long frozen
    TShfeFtdcVolumeType  LongFrozen;
    ///Short frozen
    TShfeFtdcVolumeType  ShortFrozen;
    ///Previous day long frozen
    TShfeFtdcVolumeType  YdLongFrozen;
    ///Previous day short frozen
    TShfeFtdcVolumeType  YdShortFrozen;
    ///Buying volume on that day
    TShfeFtdcVolumeType  BuyTradeVolume;
    ///Selling volume on that day
    TShfeFtdcVolumeType  SellTradeVolume;
    ///Cost of carry
    TShfeFtdcMoneyType  PositionCost;
    ///Yesterday's cost of carry
    TShfeFtdcMoneyType  YdPositionCost;
    ///Margin used
    TShfeFtdcMoneyType  UseMargin;
    ///Frozen margin
    TShfeFtdcMoneyType  FrozenMargin;
    ///Margin frozen by the long
    TShfeFtdcMoneyType  LongFrozenMargin;
    ///Margin frozen by the short
    TShfeFtdcMoneyType  ShortFrozenMargin;
    ///Frozen premium
    TShfeFtdcMoneyType  FrozenPremium;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the request ID of the member position query; this ID is specified by the user upon performing the client position query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.25. OnRspQryInstrument Method

This method is for the reply on contract query. After Member System sends out contract query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryInstrument(
        CShfeFtdcRspInstrumentField* pRspInstrument,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspInstrument: pointer to the contract structure. The structure:

```
struct CShfeFtdcRspInstrumentField {
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Product ID
    TShfeFtdcProductIDType  ProductID;
    ///Product suite's ID
    TShfeFtdcProductGroupIDType  ProductGroupID;
    ///Basic commodity ID
    TShfeFtdcInstrumentIDType  UnderlyingInstrID;
    ///Product type
    TShfeFtdcProductClassType  ProductClass;
    ///Type of open interest
    TShfeFtdcPositionTypeType  PositionType;
    ///Strike price
    TShfeFtdcPriceType  StrikePrice;
    ///Option type
    TShfeFtdcOptionsTypeType  OptionsType;
    ///Contract multiplier
    TShfeFtdcVolumeMultipleType  VolumeMultiple;
    ///Contract multiplier for basic commodity
    TShfeFtdcUnderlyingMultipleType  UnderlyingMultipl
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Contract name
    TShfeFtdcInstrumentNameType  InstrumentName;
    ///Delivery year
    TShfeFtdcYearType  DeliveryYear;
    ///Delivery month
    TShfeFtdcMonthType  DeliveryMonth;
    ///Month in advance
    TShfeFtdcAdvanceMonthType  AdvanceMonth;
    ///Is trading right now?
    TShfeFtdcBoolType  IsTrading;
```

```
    ///Creation date
    TShfeFtdcDateType  CreateDate;
    ///Listing day
    TShfeFtdcDateType  OpenDate;
    ///Expiring date
    TShfeFtdcDateType  ExpireDate;
    ///Date of starting delivery
    TShfeFtdcDateType  StartDelivDate;
    ///The last delivery day
    TShfeFtdcDateType  EndDelivDate;
    ///Benchmark price for listing
    TShfeFtdcPriceType  BasisPrice;
    ///The Max. market order placement volume
    TShfeFtdcVolumeType  MaxMarketOrderVolume;
    ///Minimum Order Quantity for Market Orders
    TShfeFtdcVolumeType  MinMarketOrderVolume;
    ///The Max. limit order placemnt volume
    TShfeFtdcVolumeType  MaxLimitOrderVolume;
    ///The Min. limit order placement volume
    TShfeFtdcVolumeType  MinLimitOrderVolume;
    ///Minimum Price Fluctuation
    TShfeFtdcPriceType  PriceTick;
    ///Position opened by natural person during delvery month
    TShfeFtdcMonthCountType  AllowDelivPersonOpen;
    ///Currency ID
    TShfeFtdcCurrencyIDType  CurrencyID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
80	User is not authorized to do so	Only trading users are allowed to perform the query

nRequestID: returns the contract query request ID; this ID is specified by the user upon performing the contract query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.26. OnRspQryInstrumentStatus Method

This method is for the reply on contract trading status query. After Member System sends out contract trading status query instruction and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryInstrumentStatus(
        CShfeFtdcInstrumentStatusField* pInstrumentStatus,
```

```
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInstrumentStatus: pointer to the contract trading status structure. The structure:

```
struct CshfeFtdcInstrumentStatusField {
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Contract/Instrument Trading Status
    TShfeFtdcInstrumentStatusType  InstrumentStatus;
    ///Trading Phase/Stage/Segment ID
    TShfeFtdcTradingSegmentSNType  TradingSegmentSN;
    ///Time of entering current status
    TShfeFtdcTimeType  EnterTime;
    ///Reason for entering current status
    TShfeFtdcInstStatusEnterReasonType  EnterReason;
    ///Entry Date of Current Status
    TShfeFtdcDateType  EnterDate;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
80	User is not authorized to do so	Only trading users are allowed to perform the query

nRequestID: returns the request ID of the contract trading status query; this ID is specified by the user upon performing the contract trading status query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.27. OnRspQryBulletin Method

This method is for the reply on the Exchange bulletin/public announcement query. After Member System sends out the query instruction for the Exchange bulletin/public announcement and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryBulletin(
        CShfeFtdcBulletinField* pBulletin,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pBulletin: pointer to the Exchange bulletin/public announcement structure. The structure:

```
struct CShfeFtdcBulletinField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Bulletin number
    TShfeFtdcBulletinIDType  BulletinID;
    ///Sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
    ///Bulletin type
    TShfeFtdcNewsTypeType  NewsType;
    ///Urgency
    TShfeFtdcNewsUrgencyType  NewsUrgency;
    ///Transmission time
    TShfeFtdcTimeType  SendTime;
    ///Message digest
    TShfeFtdcAbstractType  Abstract;
    ///Source of message
    TShfeFtdcComeFromType  ComeFrom;
    ///Message body
    TShfeFtdcContentType  Content;
    ///WEB address
    TShfeFtdcURLLinkType  URLLink;
    ///Market ID
    TShfeFtdcMarketIDType  MarketID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
80	User is not authorized to do so	Only trading users are allowed to perform the query

nRequestID: returns the request ID of the exchange bulletin query; this ID is specified by the user upon performing the exchange bulletin query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.28. OnRspQryMarketData Method

This method is for the reply on general market data query. After Member System sends out the query instruction for market data and while the Trading System sends back the response, this method will be called.

Function Prototype:

```
void OnRspQryMarketData(
```

```
        CShfeFtdcMarketDataField* pMarketData,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pMarketData: pointer to the market data structure. The structure:

```
struct CShfeFtdcMarketDataField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///The latest price
    TShfeFtdcPriceType  LastPrice;
    ///Settlement of yesterday
    TShfeFtdcPriceType  PreSettlementPrice;
    ///Close of yesterday
    TShfeFtdcPriceType  PreClosePrice;
    ///Yesterday's open interest
    TShfeFtdcLargeVolumeType  PreOpenInterest;
    ///Today's open price
    TShfeFtdcPriceType  OpenPrice;
    ///The highest price
    TShfeFtdcPriceType  HighestPrice;
    ///The lowest price
    TShfeFtdcPriceType  LowestPrice;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Turnover
    TShfeFtdcMoneyType  Turnover;
    ///Open Interest
    TShfeFtdcLargeVolumeType  OpenInterest;
    ///Today's closing
    TShfeFtdcPriceType  ClosePrice;
    ///Today's settlement
    TShfeFtdcPriceType  SettlementPrice;
    ///Upward limit price
    TShfeFtdcPriceType  UpperLimitPrice;
    ///Downward limit price
    TShfeFtdcPriceType  LowerLimitPrice;
    ///Yesterday's delta value
    TShfeFtdcRatioType  PreDelta;
    ///Today's delta value
    TShfeFtdcRatioType  CurrDelta;
    ///Last amendment time
    TShfeFtdcTimeType  UpdateTime;
    ///The last modified millisecond
    TShfeFtdcMillisecType  UpdateMillisec;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Action day
```

```
    TShfeFtdcDateType  ActionDay;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
80	User is not authorized to do so	Only trading users are allowed to perform the query

nRequestID: returns the request ID of the standard market data query; this ID is specified by the user upon performing the standard market data query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.29. OnRspQryHedgeVolume Method

Hedging quota response. This method will be called when the Trading System returns a response after the Member System executes a hedging quota query.

Function Prototype:

```
void OnRspQryHedgeVolume(
        CShfeFtdcHedgeVolumeField* pHedgeVolume,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pHedgeVolume: points to the hedging quota volume structure. The structure:

```
struct CShfeFtdcHedgeVolumeField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Initial applied quantity for long hedging quota, in lots.
    TShfeFtdcVolumeType  LongVolumeOriginal;
    ///Initial applied quantity for short hedging quota, in lots.
    TShfeFtdcVolumeType  ShortVolumeOriginal;
    ///Long hedging quota, in lots.
    TShfeFtdcVolumeType  LongVolume;
```

```
    ///Short hedging quota, in lots.
    TShfeFtdcVolumeType  ShortVolume;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member. When the exchange enables cross-member joint hedging, only the administrator can perform this operation

nRequestID: returns the request ID of the hedge quota execution query; this ID is specified by the user upon performing the hedging quota execution query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.30. OnRtnTrade Method

Trade return. When a trade is executed, the Trading System will notify corresponding member systems, and this method will be called.

Function Prototype:

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

Parameters:

pTrade: pointer to the trade return structure. Note: some fields in the trade return are not used, and the Trading System returns null for those unused fields. The structure:

```
struct CShfeFtdcTradeField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Matched order ID
    TShfeFtdcTradeIDType  TradeID;
    ///buy-sell direction
    TShfeFtdcDirectionType  Direction;
    ///Order ID
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
```

```
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Trading role, not used
    TShfeFtdcTradingRoleType  TradingRole;
    ///Fund account, not used
    TShfeFtdcAccountIDType  AccountID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Price
    TShfeFtdcPriceType  Price;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Transaction time
    TShfeFtdcTimeType  TradeTime;
    ///Trade Type / order matching type
    TShfeFtdcTradeTypeType  TradeType;
    ///Source of transaction price, not used
    TShfeFtdcPriceSourceType  PriceSource;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local order ID
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Business unit, NOT USED
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
};
```

2.1.31. OnRtnOrder Method

Order return. When an order is inserted, executed or for other reasons (i.e. partial match) so that the order status changes, the Trading System will automatically inform Member System, and this method will be called.

Function Prototype:

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

Parameters:

pOrder: pointer to the order return structure. Note: some fields in the order return is not used, the Trading System will return a null value for those used fields. The structure:

```
struct CShfeFtdcOrderField {
    ///Business day, not used
    TShfeFtdcDateType  TradingDay;
```

```
///Settlement group's ID, not used
TShfeFtdcSettlementGroupIDType  SettlementGroupID;
///Settlement number, not used
TShfeFtdcSettlementIDType  SettlementID;
///Order ID
TShfeFtdcOrderSysIDType  OrderSysID;
///Member ID
TShfeFtdcParticipantIDType  ParticipantID;
///Client ID
TShfeFtdcClientIDType  ClientID;
///Transaction user's ID
TShfeFtdcUserIDType  UserID;
///Instrument/contract ID
TShfeFtdcInstrumentIDType  InstrumentID;
///Order Price Type
TShfeFtdcOrderPriceTypeType  OrderPriceType;
///buy-sell direction
TShfeFtdcDirectionType  Direction;
///Combination offset flag
TShfeFtdcCombOffsetFlagType  CombOffsetFlag;
///Combination Hedge Flag
TShfeFtdcCombHedgeFlagType  CombHedgeFlag;
///Price
TShfeFtdcPriceType  LimitPrice;
///Volume
TShfeFtdcVolumeType  VolumeTotalOriginal;
///Expiry Type
TShfeFtdcTimeConditionType  TimeCondition;
///GTD DATE
TShfeFtdcDateType  GTDDate;
///Match volume condition type
TShfeFtdcVolumeConditionType  VolumeCondition;
///Minimum Volume
TShfeFtdcVolumeType  MinVolume;
///Trigger/Contingent Condition
TShfeFtdcContingentConditionType  ContingentCondition;
///Stop-loss price
TShfeFtdcPriceType  StopPrice;
///Forced close reasons
TShfeFtdcForceCloseReasonType  ForceCloseReason;
///Local order ID
TShfeFtdcOrderLocalIDType  OrderLocalID;
///Auto Suspend flag
TShfeFtdcBoolType  IsAutoSuspend;
///Source of order, not used
TShfeFtdcOrderSourceType  OrderSource;
///Order Status
TShfeFtdcOrderStatusType  OrderStatus;
///Order Type
TShfeFtdcOrderTypeType  OrderType;
///Volume on that day, not used
TShfeFtdcVolumeType  VolumeTraded;
///Remaining volume
TShfeFtdcVolumeType  VolumeTotal;
```

```
        ///order date
        TShfeFtdcDateType  InsertDate;
        ///Entry time
        TShfeFtdcTimeType  InsertTime;
        ///activation time, NOT USED
        TShfeFtdcTimeType  ActiveTime;
        ///Suspension time, NOT USED
        TShfeFtdcTimeType  SuspendTime;
        ///Last amendment time
        TShfeFtdcTimeType  UpdateTime;
        ///Time of cancelation, not used
        TShfeFtdcTimeType  CancelTime;
        ///Last modified trading user ID
        TShfeFtdcUserIDType  ActiveUserID;
        ///Priority, NOT USED
        TShfeFtdcPriorityType  Priority;
        ///Sequence number by time order, NOT USED
        TShfeFtdcTimeSortIDType  TimeSortID;
        ///Settlement member ID, NOT USED
        TShfeFtdcParticipantIDType  ClearingPartID;
        ///Business unit, NOT USED
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///Action day
        TShfeFtdcDateType  ActionDay;
        ///IP Address, not used
        TShfeFtdcIPAddressType  IPAddress;
        ///MAC Address, not used
        TShfeFtdcMacAddressType  MacAddress;
    };
```

2.1.32. OnRtnQuote Method

Quote return. When an order is inserted or executed so that the price quote changes, the Trading System will automatically inform Member System, and this method will be called.

Function Prototype:

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

Parameters:

pQuote: pointer to the price quote return structure. The structure:

```
    struct CShfeFtdcQuoteField {
        ///Business day
        TShfeFtdcDateType  TradingDay;
        ///Settlement group's ID
        TShfeFtdcSettlementGroupIDType  SettlementGroupID;
        ///Settlement number
        TShfeFtdcSettlementIDType  SettlementID;
        ///Quoto number
        TShfeFtdcQuoteSysIDType  QuoteSysID;
        ///Member ID
```

```
        TShfeFtdcParticipantIDType  ParticipantID;
        ///Client ID
        TShfeFtdcClientIDType  ClientID;
        ///Transaction user's ID
        TShfeFtdcUserIDType  UserID;
        ///Volume
        TShfeFtdcVolumeType  Volume;
        ///Instrument/contract ID
        TShfeFtdcInstrumentIDType  InstrumentID;
        ///Local quoto number
        TShfeFtdcOrderLocalIDType  QuoteLocalID;
        ///Business unit
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Buyer's combination offset flag
        TShfeFtdcCombOffsetFlagType  BidCombOffsetFlag;
        ///Buyer's combination hedge flag
        TShfeFtdcCombHedgeFlagType  BidCombHedgeFlag;
        ///Buyer's price
        TShfeFtdcPriceType  BidPrice;
        ///Seller's combination offset flag
        TShfeFtdcCombOffsetFlagType  AskCombOffsetFlag;
        ///Seller's combination hedge flag
        TShfeFtdcCombHedgeFlagType  AskCombHedgeFlag;
        ///Seller's price
        TShfeFtdcPriceType  AskPrice;
        ///Entry time
        TShfeFtdcTimeType  InsertTime;
        ///Cancellation time
        TShfeFtdcTimeType  CancelTime;
        ///Transaction time
        TShfeFtdcTimeType  TradeTime;
        ///Buyer's order number
        TShfeFtdcOrderSysIDType  BidOrderSysID;
        ///Seller's order number
        TShfeFtdcOrderSysIDType  AskOrderSysID;
        ///Settlement member's number
        TShfeFtdcParticipantIDType  ClearingPartID;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///Action day
        TShfeFtdcDateType  ActionDay;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
        ///Quote request ID
        TShfeFtdcOrderSysIDType  QuoteDemandID;
    };
```

2.1.33. OnRtnExecOrder Method

Order exercise return. This method will be called when the Member System performs an option exercise entry or option exercise operation resulting in a change of option exercise

status, and the Trading System will proactively notify the Member System.

Function Prototype:

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

Parameters:

pExecOrder: pointer to the order execution return structure. The structure:

```
struct CShfeFtdcExecOrderField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local annoncement execution number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Position direction, i.e. whether buyer(long position) or seller(short position)
    made this application
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Flag indicating whether to retain futures positions after option exercise, not
    used
    TShfeFtdcExecOrderPositionFlagType  ReservePositionFlag;
    ///Whether the futures positions generated after option exercise are self-hedged
    TShfeFtdcExecOrderCloseFlagType  CloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Option exercise number
    TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
    ///Order date
    TShfeFtdcDateType  InsertDate;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Execution result
    TShfeFtdcExecResultType  ExecResult;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
```

```
    ///Action day
    TShfeFtdcDateType  ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

2.1.34. OnRtnInstrumentStatus Method

Contract return. When the contract status changes, the Trading System will automatically inform Member System, and this method will be called.

Function Prototype:

```
void OnRtnInstrumentStatus(
        CShfeFtdcInstrumentStatusField* pInstrumentStatus);
```

Parameters:

pInstrumentStatus: pointer to the contract status structure. The structure:

```
struct CShfeFtdcInstrumentStatusField {
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Contract/Instrument Trading Status
    TShfeFtdcInstrumentStatusType  InstrumentStatus;
    ///Trading Phase/Stage/Segment ID
    TShfeFtdcTradingSegmentSNType  TradingSegmentSN;
    ///Time of entering current status
    TShfeFtdcTimeType  EnterTime;
    ///Reason for entering current status
    TShfeFtdcInstStatusEnterReasonType  EnterReason;
    ///Entry Date of Current Status
    TShfeFtdcDateType  EnterDate;
};
```

2.1.35. OnRtnInsInstrument Method

New contract notification. After successfully logging into the Member System, the Trading System will notify the Member System of newly added contracts via the public stream.

Function Prototype:

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

Parameters:

pInstrument: pointer to the contract structure. The structure:

```
struct CShfeFtdcInstrumentField {
```

```
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Product ID
    TShfeFtdcProductIDType  ProductID;
    ///Product suite's ID
    TShfeFtdcProductGroupIDType  ProductGroupID;
    ///Basic commodity ID
    TShfeFtdcInstrumentIDType  UnderlyingInstrID;
    ///Product type
    TShfeFtdcProductClassType  ProductClass;
    ///Type of open interest
    TShfeFtdcPositionTypeType  PositionType;
    ///Strike price
    TShfeFtdcPriceType  StrikePrice;
    ///Option type
    TShfeFtdcOptionsTypeType  OptionsType;
    ///Contract multiplier
    TShfeFtdcVolumeMultipleType  VolumeMultiple;
    ///Contract multiplier for basic commodity
    TShfeFtdcUnderlyingMultipleType  UnderlyingMultiple;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Contract name
    TShfeFtdcInstrumentNameType  InstrumentName;
    ///Delivery year
    TShfeFtdcYearType  DeliveryYear;
    ///Delivery month
    TShfeFtdcMonthType  DeliveryMonth;
    ///Month in advance
    TShfeFtdcAdvanceMonthType  AdvanceMonth;
    ///Is trading right now?
    TShfeFtdcBoolType  IsTrading;
    ///Currency ID
    TShfeFtdcCurrencyIDType  CurrencyID;
};
```

2.1.36. OnRtnBulletin Method

Announcement. When the Exchange sends announcement through the Trading System, the Trading System will automatically inform Member System, and this method will be called.

Function Prototype:

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

Parameters:

pBulletin: pointer to the announcement structure. The structure:

```
struct CShfeFtdcBulletinField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Bulletin number
    TShfeFtdcBulletinIDType  BulletinID;
    ///Sequence number
```

```
    TShfeFtdcSequenceNoType  SequenceNo;
    ///Bulletin type
    TShfeFtdcNewsTypeType  NewsType;
    ///Urgency
    TShfeFtdcNewsUrgencyType  NewsUrgency;
    ///Transmission time
    TShfeFtdcTimeType  SendTime;
    ///Message digest
    TShfeFtdcAbstractType  Abstract;
    ///Source of message
    TShfeFtdcComeFromType  ComeFrom;
    ///Message body
    TShfeFtdcContentType  Content;
    ///WEB address
    TShfeFtdcURLLinkType  URLLink;
    ///Market ID
    TShfeFtdcMarketIDType  MarketID;
};
```

2.1.37. OnRtnFlowMessageCancel Method

Data stream rollback notification. After the Trading System performs a disaster recovery switch and when the user logs back into the Trading System and subscribes to a specific data stream (either private or public), the Trading System will proactively notify the Member System that certain messages in the data stream have been invalidated or canceled. At this time, this method will be called.

Function Prototype:

```
void OnRtnFlowMessageCancel(
        CShfeFtdcFlowMessageCancelField* pFlowMessageCancel);
```

Parameters:

pFlowMessageCancel: pointer to the data stream cancellation structure. The structure:

```
struct CShfeFtdcFlowMessageCancelField{
    ///Serial number in sequence
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Datacenter ID
    TShfeFtdcDataCenterIDType  DataCenterID;
    ///Starting sequence number of rollback
    TShfeFtdcSequenceNoType  StartSequenceNo;
    ///Ending sequence number of rollback
    TShfeFtdcSequenceNoType  EndSequenceNo;
};
SequenceSeries: Datastream ID of rollback occured (private stream or public stream)
Message range of rollback: (StartSequenceNo,EndSequenceNo]
```

2.1.38. OnErrRtnOrderInsert Method

Order entry error return: sent automatically by the Trading System to Member System. When the Member System sends an order entry instruction and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnOrderInsert(
        CShfeFtdcInputOrderField* pInputOrder,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pInputOrder: points to the order entry structure, including the input data submitted during the order entry. The structure:

```
struct CShfeFtdcInputOrderField {
    ///Order number, this field will be returned by Trading System.
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Order Price Type
    TShfeFtdcOrderPriceTypeType  OrderPriceType;
    ///buy-sell direction
    TShfeFtdcDirectionType  Direction;
    ///Combination Offset flag
    TShfeFtdcCombOffsetFlagType  CombOffsetFlag;
    ///Combination Hedge Flag
    TShfeFtdcCombHedgeFlagType  CombHedgeFlag;
    ///Price
    TShfeFtdcPriceType  LimitPrice;
    ///Volume
    TShfeFtdcVolumeType  VolumeTotalOriginal;
    ///Expiry Type
    TShfeFtdcTimeConditionType  TimeCondition;
    ///GTD DATE
    TShfeFtdcDateType  GTDDate;
    ///Match volume condition type
    TShfeFtdcVolumeConditionType  VolumeCondition;
    ///Minimum Volume
    TShfeFtdcVolumeType  MinVolume;
    ///Trigger/Contingent Condition
    TShfeFtdcContingentConditionType  ContingentCondition;
    ///Stop-loss price
    TShfeFtdcPriceType  StopPrice;
    ///Forced close reasons
    TShfeFtdcForceCloseReasonType  ForceCloseReason;
    ///Local order ID
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Auto Suspend flag
```

```
    TShfeFtdcBoolType  IsAutoSuspend;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract not found	The contract specified in the order could not be found
3	Member not found	The member specified in the order could not be found
4	Client not found	The client specified in the order could not be found
6	Invalid order fields	The order contains invalid field values (e.g., out-of-range enumerations) or a non-force-close order includes a force-close reason
12	Duplicate order	The local order ID in the order is duplicated
15	Client not registered with member	The client in the order is not registered under the specified member
16	IOC orders only allowed in continuous trading	An IOC order was submitted outside the continuous trading session
17	GFA orders only allowed in call auction	A GFA order was submitted outside the call auction phase
19	Quantity constraint must be IOC	The time-in-force for a non-any-quantity order must be IOC
20	GTD order expired	The GTD date in the GTD order has already expired
21	Minimum quantity exceeds order quantity	The minimum quantity specified exceeds the total order quantity
22	Exchange data not synchronized	The Trading System is not fully initialized. Please retry later
23	Clearing group data not synchronized	The Trading System is not fully initialized. Please retry later
26	Operation not allowed in current state	The contract is not in continuous trading, call auction, or call auction equilibrium state
31	Insufficient client position	The client does not have enough position to place the close order
32	Client position limit exceeded	The client exceeded their open position limit
33	Insufficient member position	The member does not have enough position to place the close order
34	Member position limit exceeded	The member exceeded their position limit
35	Account not found	The account specified in the order could not be found
36	Insufficient funds	The account does not have sufficient funds
37	Invalid quantity	The order quantity is not a positive multiple of the minimum order quantity or exceeds the maximum

48	Price not a multiple of tick size	The order price is not a valid multiple of the contract's tick size
49	Price exceeds upper limit	The order price exceeds the contract's upper limit
50	Price below lower limit	The order price is below the contract's lower limit
51	No trading permission	No trading permission for the contract, client, or user
52	Close-only permission	Only close orders are permitted for this member, client, or user
53	Trading role not assigned	The member does not hold the required trading role for the client on the contract
54	Session not found	The user is not logged in
57	Operation on another member not allowed	The user attempted an operation for a non-affiliated member
58	User mismatch	The user in the order does not match the logged-in user
72	Natural person cannot open positions	Natural person clients are not allowed to open positions in the delivery month
78	GTD date not specified	GTD order lacks a specified GTD date
79	Unsupported order type	The exchange does not support this type of order
83	Stop orders allowed only in continuous trading	Stop orders are not allowed outside the continuous trading phase
84	Stop orders must be IOC or GFD	Stop orders must have a time condition of IOC or GFD
95	Stop order must specify stop price	Stop price is missing in the stop order
96	Insufficient hedge quota	Client's hedge quota is insufficient for hedge order
98	Force-close orders require admin	Only admin users may submit force-close orders
101	Clearing members cannot trade	The member type of the order is a clearing member
102	Clearing member not found	Cannot find the clearing member corresponding to the order member
103	Intraday hedge position cannot be closed	Hedge positions opened today cannot be closed using close-today orders
114	Best price orders cannot queue	Best price orders must have time-in-force = IOC
131	Client exceeds intraday open limit	Client exceeded the intraday open limit for the contract
132	Client exceeds per-second order limit	Client exceeded order limit per second on the product
153	Market orders must be GFD or IOC	Market orders must have time-in-force = GFD or IOC
154	Market orders allowed only in continuous trading	Market orders are not allowed outside continuous trading
155	Market orders only supported for futures and options	Market orders not allowed on non-futures/options contracts
1005	No record	The contract record associated with the order is missing

2.1.39. OnErrRtnOrderAction Method

Order operation error return. When the Member System sends an order operation instruction and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnOrderAction(
        CShfeFtdcOrderActionField* pOrderAction,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pOrderAction: pointer to the order operation structure, including the input data while

submitting the order operation and the order ID returned from the Trading System. The structure:

```
struct CShfeFtdcOrderActionField {
    ///Order number, this field will be returned by Trading System.
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Local order ID
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Price, NOT USED
    TShfeFtdcPriceType  LimitPrice;
    ///Change in quantity, NOT USED
    TShfeFtdcVolumeType  VolumeChange;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit, NOT USED
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract not found	The contract specified in the order could not be found
3	Member not found	The member specified in the order could not be found
4	Client not found	The client specified in the order could not be found
8	Invalid field value in order operation	The order contains invalid field values (e.g., out-of-range enumerations)
15	Client not registered with member	The client in the order is not registered under the specified member
16	IOC orders only allowed in continuous trading	An IOC order was submitted outside the continuous trading session
17	GFA orders only allowed in call auction	A GFA order was submitted outside the call auction phase
20	GTD order expired	The GTD date in the GTD order has already expired
22	Exchange data not synchronized	The Trading System is not fully initialized. Please retry later

23	Clearing group data not synchronized	The Trading System is not fully initialized. Please retry later
24	Order not found	The specified order to be operated on cannot be found
26	Operation not allowed in current state	The contract is not in continuous trading, call auction, or call auction equilibrium state
28	Order fully filled	The order has already been fully executed
29	Order already canceled	The order has already been canceled
30	Insufficient quantity for modification	The remaining order quantity would be less than 0 after modification
31	Insufficient client position	The client does not have enough position to place the close order
32	Client position limit exceeded	The client exceeded their open position limit
33	Insufficient member position	The member does not have enough position to place the close order
34	Member position limit exceeded	The member exceeded their position limit
35	Account not found	The account specified in the order could not be found
36	Insufficient funds	The account does not have sufficient funds
37	Invalid quantity	The order quantity is not a positive multiple of the minimum order quantity or exceeds the maximum
48	Price not a multiple of tick size	The order price is not a valid multiple of the contract's tick size
49	Price exceeds upper limit	The order price exceeds the contract's upper limit
50	Price below lower limit	The order price is below the contract's lower limit
51	No trading permission	No trading permission for the contract, client, or user
52	Close-only permission	Only close orders are permitted for this member, client, or user
54	Session not found	The user is not logged in
57	Operation on another member not allowed	The user attempted an operation for a non-affiliated member
58	User mismatch	The user in the order does not match the logged-in user
71	Operation on derivative order not allowed	The user attempted to operate on a derivative order
72	Natural person cannot open positions	Natural person clients are not allowed to open positions in the delivery month
76	Order already suspended	The order has already been suspended at the time of the suspension request
77	Order already activated	The order has already been activated at the time of the activation request
79	Unsupported order type	The exchange does not support this type of order
83	Stop orders allowed only in continuous trading	Stop orders are not allowed outside the continuous trading phase
95	Stop order must specify stop price	Stop price is missing in the stop order
96	Insufficient hedge quota	Client's hedge quota is insufficient for hedge order
98	Force-close orders require admin	Only admin users may submit force-close orders
99	Operation on behalf of another user not permitted	The user attempted to operate on an order submitted by a different user under the same member without proper authorization
131	Client exceeds intraday open limit	Client exceeded the intraday open limit for the contract
133	Client exceeds per-second cancelation limit	Client exceeded the allowed number of cancellations for a specific product within one second

2.1.40. OnErrRtnQuoteInsert Method

Erroneous quote entry return. When the Member System sends a quote entry instruction

and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnQuoteInsert(
        CShfeFtdcInputQuoteField* pInputQuote,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pInputQuote: pointer to the input quote structure, including the input data for quote entry operation and the quote number returned from the Trading System. The input quote structure:

```
struct CShfeFtdcInputQuoteField {
    ///Quoto number,this field will be returned by Trading System.
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Buyer's combination offset flag
    TShfeFtdcCombOffsetFlagType  BidCombOffsetFlag;
    ///Buyer's combination hedge flag
    TShfeFtdcCombHedgeFlagType  BidCombHedgeFlag;
    ///Buyer's price
    TShfeFtdcPriceType  BidPrice;
    ///Seller's combination offset flag
    TShfeFtdcCombOffsetFlagType  AskCombOffsetFlag;
    ///Seller's combination hedge flag
    TShfeFtdcCombHedgeFlagType  AskCombHedgeFlag;
    ///Seller's price
    TShfeFtdcPriceType  AskPrice;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
    ///Quote request ID
    TShfeFtdcOrderSysIDType  QuoteDemandID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
```

```
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in quote not found
3	Member cannot be found	Member in quote not found
4	Client cannot be found	Client in quote not found
7	Quote field error	Invalid field value in quote (enum value out of range)
13	Duplicate quote	Duplicate local quote ID in the quote
15	Client didn't open an account at this member	Client in quote has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	Contract trading status is neither continuous trading, call auction order entry, nor call auction balancing
31	Insufficient client position for closing	Client's position insufficient
32	Exceeded client position limit	Quote causes client's general position to exceed the limit
33	Insufficient member position for closing	Member's position insufficient
34	Exceeded member position limit	Quote causes member's general position to exceed the limit
35	Account not found	Fund account used in quote not found
36	Insufficient funds	Insufficient funds in the fund account
37	Invalid quantity	The quote quantity is not a positive multiple of the minimum order quantity or exceeds the maximum
48	Price not a multiple of minimum price fluctuation	Quote price is not an integer multiple of the contract's minimum price fluctuation
49	Price exceeds upper limit price	Quote price exceeds contract's upper limit price
50	Price falls below lower limit price	Quote price falls below contract's lower limit price
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
52	Close-only	Only the member, client, or user has permission to close positions on the specified contract
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in quote does not match the logged-in user
72	Opening positions not allowed for natural persons	Natural person client initiates an opening order in the delivery month
79	Order type that is not supported	The Exchange does not support this order type
96	Insufficient hedge quota	Client's hedging quota insufficient when submitting quote
98	Forced liquidation orders must be used by administrators	Non-administrator user submitted a forced liquidation order
101	Clearing members are not allowed to trade	Member type in quote is a clearing member

102	Corresponding clearing member not found	Clearing member corresponding to the quote's member not found
103	Same-day hedging positions cannot be closed	Hedging positions should not use close-today quotes for closing
131	Exceeded client's intraday contract opening limit	Client's opening quantity on a contract exceeds the intraday opening limit
132	Exceeded client's per-second order limit for the product	Number of client orders on a product within one second exceeds the limit
1005	No record	Contract record corresponding to the quote is missing

2.1.41. OnErrRtnQuoteAction Method

Erroneous quote return. When the Member System sends a quote instruction and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnQuoteAction(
        CShfeFtdcQuoteActionField* pQuoteAction,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pQuoteAction: pointer to the quote structure, including the input data for quote request and the quote number returned from Trading System. The quote structure:

```
struct CShfeFtdcQuoteActionField {
    ///Quote number,this field will be returned by Trading System.
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Local quote number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
```

```
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in quote operation not found
3	Member cannot be found	Member in quote operation not found
4	Client cannot be found	Client in quote operation not found
8	Order operation field error	Invalid field value in derived orders from quote operation (e.g., price is not a valid float or not within the valid range)
9	Quote operation field error	Invalid field value in quote operation (enum value out of range or operation flag is modify, activate, or suspend)
15	Client didn't open an account at this member	Client has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
25	Quote not found	Quote to be operated on cannot be found
26	This operation is prohibited by current state	For activation operations, the contract trading status is neither continuous trading, auction order, nor auction equilibrium For other operations, the trading status is neither continuous trading nor auction order
28	Order already fully filled	Orders derived from the quote have been fully filled
29	Order already canceled	Orders derived from the quote have been canceled
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in quote operation does not match the logged-in user
70	Quote has already been canceled	Quote has already been canceled
99	Cannot operate on behalf of other users	Unauthorized user operating on quotes submitted by other users under the same member

2.1.42. OnErrRtnExecOrderInsert Method

Option exercise entry error return. When the Member System sends an option exercise and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnExecOrderInsert(
        CShfeFtdcInputExecOrderField* pInputExecOrder,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pInputExecOrder: pointer to the option exercise structure. The structure:

```
struct CShfeFtdcInputExecOrderField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local annoncement execution number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///position direction, i.e. whether buyer(long position) or seller(short position)
    made this application
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Flag indicating whether to retain futures positions after option exercise, not
    used
    TShfeFtdcExecOrderPositionFlagType  ReservePositionFlag;
    ///Whether the futures positions generated after option exercise are self-hedged
    TShfeFtdcExecOrderCloseFlagType  CloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract cannot be found in the option exercise
3	Member cannot be found	Member cannot be found in the option exercise
4	Client cannot be found	Client cannot be found in the option exercise
15	Client didn't open an account at this member	Client in the in the option exercise didn' open an account at the designated member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later

23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
31	Insufficient client position for closing	Client's position insufficient when entering option exercise
33	Insufficient member position for closing	Member's position insufficient when entering option exercise
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account
37	Invalid quantity	Invalid quantity in option exercise
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option exercise does not match the logged-in user
79	Order type that is not supported	The Exchange does not support this order type
89	Option exercise field error	Invalid field value in option exercise (enum value out of range)
91	Duplicate option exercise	Duplicate local option exercise ID in the option exercise
94	Option exercise is allowed only for options	Contract in option exercise is a non-option contract
101	Clearing members are not allowed to trade	Member type in option exercise is a clearing member
102	Corresponding clearing member not found	Clearing member corresponding to the option exercise's member not found
127	Not within declaration period	Not within the contract delivery period (exercisable period)
129	Option exercise or abandonment cannot be opening orders	The offset flag in the execution of declaration must indicate closing
146	Only holders of long positions can exercise	Only option buyers can exercise
1005	No record	Contract record corresponding to the option exercise is missing

2.1.43. OnErrRtnExecOrderAction Method

Option exercise operation error return. When the Member System sends an option exercise operation instruction and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnExecOrderAction(
        CShfeFtdcExecOrderActionField* pExecOrderAction,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pExecOrderAction: pointer to the option exercise operation structure. The structure:

```
struct CShfeFtdcExecOrderActionField {
    ///Option exercise number
```

```
    TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
    ///Local annoncement execution number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option exercise operation not found
3	Member cannot be found	Member in option exercise operation not found
4	Client cannot be found	Client in option exercise operation not found
15	Client didn't open an account at this member	Client in option exercise operation has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option exercise operation does not match the logged-in user
89	Option exercise field error	Invalid field value in option exercise
90	Error field in the execution of	Invalid field value in option exercise operation (enum

	declaration opration	value out of range or operation flag is modify, activate, or suspend)
92	The execution of declaration has been canceled	The declaration operation to be executed has been canceled
93	The execution of declaration can not be found	The declaration operation to be executed cann not be found
127	Not within declaration period	Not within the contract delivery period (exercisable period)
1005	No record	Contract record corresponding to the option exercise operation is missing

2.1.44. OnRspQryExecOrder Method

Option exercise query response. When the Trading System automatically informs the Member System, this method will be called.

Function Prototype:

```
void OnRspQryExecOrder(
        CShfeFtdcExecOrderField* pExecOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pExecOrder: pointer to the option exercise structure. The structure:

```
struct CShfeFtdcExecOrderField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local annoncement execution number
    TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///position direction, i.e. whether buyer(long position) or seller(short position)
    made this application
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Flag indicating whether to retain futures positions after option exercise, not
    used
```

```
    TShfeFtdcExecOrderPositionFlagType  ReservePositionFlag;
    ///Whether the futures positions generated after option exercise are self-hedged
    TShfeFtdcExecOrderCloseFlagType  CloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Option exercise number
    TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
    ///order date
    TShfeFtdcDateType  InsertDate;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Execution result
    TShfeFtdcExecResultType  ExecResult;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user option exercise request ID; this ID is specified by the user upon submitting an option exercise query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.45. OnRspQryExchangeRate Method

Exchange rate query response. This method will be called when the Member System sends an exchange rate query request and the Trading System returns a response.

Function Prototype:

```
void OnRspQryExchangeRate(
```

```
        CShfeFtdcRspExchangeRateField* pRspExchangeRate,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspExchangeRate: pointer to the exchange rate response information structure. The structure:

```
struct CShfeFtdcRspExchangeRateField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Currency ID
    TShfeFtdcCurrencyIDType  CurrencyID;
    ///foreign exchange unit
    TshfeFtdcRateUnitType  RateUnit;
    ///central parity rate
    TShfeFtdcExRatePriceType  RatePrice;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
80	User is not authorized to do so	Only trading users are allowed to perform the query

nRequestID: returns the user exchange rate query request ID; this ID is specified by the user when submitting an exchange rate query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.46. OnRspAbandonExecOrderInsert Method

Option abandonment response. This method will be called when the Member System sends an option abandonment entry request and the Trading System returns a response.

Function Prototype:

```
void OnRspAbandonExecOrderInsert(
        CShfeFtdcInputAbandonExecOrderField* pInputAbandonExecOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputAbandonExecOrder: pointer to the option abandonment structure. The structure:

```
struct CShfeFtdcInputAbandonExecOrderField {
    ///Contract number
```

```
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Option abandonment local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Direction of the position applying for abandon; only long positions can apply for
    abandon
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option abandonment not found
3	Member cannot be found	Member in option abandonment not found
4	Client cannot be found	Client in option abandonment not found
15	Client didn't open an account at this member	Client in option abandonment has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
31	Insufficient client position for closing	Client's position insufficient when entering option abandonment
33	Insufficient member position for closing	Member's position insufficient when entering option abandonment
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account

37	Invalid quantity	Invalid quantity in option abandonment
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option abandonment does not match the logged-in user
79	Order type that is not supported	The Exchange does not support this order type
101	Clearing members are not allowed to trade	Member type in option abandonment is a clearing member
102	Corresponding clearing member not found	Clearing member corresponding to the order's member not found
121	Option abandonment field error	Invalid field value in option abandonment
123	Duplicate option abandonment	Duplicate local option abandonment ID in the option abandonment
126	Option abandonment is allowed only for options	Contract in option abandonment is a non-option contract
128	Only holders of long positions can enjoy execution waiver	Only option buyers can enjoy execution waiver
129	Option exercise or abandonment cannot be opening orders	Offset flag in option abandonment must be limited to close
149	Option abandonment applications can only be submitted on option expiration day	Trading day is not option expiration day
1005	No record	Contract record corresponding to the option abandonment is missing

nRequestID: returns the user option abandonment request ID; this ID is specified by the user upon an option abandonment.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.47. OnRspAbandonExecOrderAction Method

Option abandonment operation response, including cancellation, suspension, activation, and modification of option abandonment. This method will be called when the Member System sends an option abandonment operation request and the Trading System returns a response.

Function Prototype:

```
void OnRspAbandonExecOrderAction(
        CShfeFtdcAbandonExecOrderActionField* pAbandonExecOrderAction,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pAbandonExecOrderAction: pointer to the option abandonment structure. The structure:

```
struct CShfeFtdcAbandonExecOrderActionField {
    ///Option abandonment system ID
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///Option abandonment local ID
```

```
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option abandonment operation not found
3	Member cannot be found	Member in option abandonment operation not found
4	Client cannot be found	Client in option abandonment operation not found
15	Client didn't open an account at this member	Client in option abandonment operation has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option abandonment operation does not match the logged-in user
121	Option abandonment field error	Invalid field value in option abandonment
122	Option abandonment operation field error	Invalid field value in option abandonment operation (enum value out of range or operation flag is modify, activate, or suspend)

124	The option abandonment has already been canceled	The option abandonment to be operated on has already been deleted
125	Option abandonment not found	The option abandonment to be operated on was not found
149	Option abandonment applications can only be submitted on option expiration day	Trading day is not option expiration day
1005	No record	The contract record corresponding to the option abandonment operation is missing

nRequestID: returns the user option abandonment request ID; this ID is specified by the user upon submitting an option abandonment.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.48. OnRspQryAbandonExecOrder Method

Option abandonment query response. This method will be called when the Member System sends an option abandonment query request and the Trading System returns a response.

Function Prototype:

```
void OnRspQryAbandonExecOrder(
        CShfeFtdcAbandonExecOrderField* pAbandonExecOrder,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pAbandonExecOrder: pointer to the option abandonment structure. The structure:

```
struct CShfeFtdcAbandonExecOrderField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Option abandonment Local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Direction of the position applying for abandon; only long positions can apply for
```

```
    abandon
    TshfeFtdcPosiDirectionType  PosiDirection;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Option abandonment system ID
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///order date
    TShfeFtdcDateType  InsertDate;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Result of abandon execution
    TShfeFtdcExecResultType  AbandonExecResult;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user option abandonment query request ID; this ID is specified by the user upon submitting an option abandonment query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.49. OnRtnAbandonExecOrder Method

Option abandonment return. When the Member System performs an option abandonment entry or operation resulting in a change of option abandonment status, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnRtnAbandonExecOrder(
```

```
CShfeFtdcAbandonExecOrderField* pAbandonExecOrder);
```

Parameters:

pAbandonExecOrder: pointer to the option abandonment structure.

```
struct CShfeFtdcAbandonExecOrderField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Option abandonment Local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Direction of the position applying for abandon; only long positions can apply for
    abandon
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///Option abandonment system ID
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///order date
    TShfeFtdcDateType  InsertDate;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Result of abandon execution
    TShfeFtdcExecResultType  AbandonExecResult;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

2.1.50. OnErrRtnAbandonExecOrderInsert Method

Option abandonment entry error return. When the Member System sends an option abandonment entry instruction and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnAbandonExecOrderInsert(
        CShfeFtdcInputAbandonExecOrderField* pInputAbandonExecOrder,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pInputAbandonExecOrder: pointer to the option abandonment entry structure.

```
struct CShfeFtdcInputAbandonExecOrderField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Option abandonment Local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge Flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Direction of the position applying for abandon; only long positions can apply for
    abandon
    TshfeFtdcPosiDirectionType  PosiDirection;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option abandonment not found
3	Member cannot be found	Member in option abandonment not found
4	Client cannot be found	Client in option abandonment not found
15	Client didn't open an account at this member	Client in option abandonment has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
31	Insufficient client position for closing	Client's position insufficient when entering option abandonment
33	Insufficient member position for closing	Member's position insufficient when entering option abandonment
35	Account not found	Required fund account not found
36	Insufficient funds	Insufficient funds in the fund account
37	Invalid quantity	Invalid quantity in option abandonment
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option abandonment does not match the logged-in user
79	Order type that is not supported	The Exchange does not support this order type
101	Clearing members are not allowed to trade	Member type in option abandonment is a clearing member
102	Corresponding clearing member not found	Clearing member corresponding to the option abandonment's member not found
121	Option abandonment field error	Invalid field value in option abandonment
123	Duplicate option abandonment	Duplicate local option abandonment ID in the option abandonment
126	Option abandonment is allowed only for options	Contract in option abandonment is a non-option contract
128	Only holders of long positions can enjoy execution waiver	Only option buyers can enjoy execution waiver
129	Option exercise or abandonment cannot be opening orders	The offset flag in the execution of declaration must indicate closing
149	Option abandonment applications can only be submitted on option expiration day	Trading day is not option expiration day
1005	No record	Contract record corresponding to the option abandonment is missing

2.1.51. OnErrRtnAbandonExecOrderAction Method

Option abandonment operation error return. When the Member System sends an option abandonment and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnAbandonExecOrderAction(
        CShfeFtdcAbandonExecOrderActionField* pAbandonExecOrderAction,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pAbandonExecOrderAction: pointer to the option abandonment structure. The structure:

```
struct CShfeFtdcAbandonExecOrderActionField {
    ///Option abandonment system ID
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///Option abandonment local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option abandonment operation not found
3	Member cannot be found	Member in option abandonment operation not found
4	Client cannot be found	Client in option abandonment operation not found
15	Client didn't open an account at this member	Client in option abandonment has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither continuous trading nor trading business processing
35	Account not found	Required fund account not found

36	Insufficient funds	Insufficient funds in the fund account
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option abandonment operation does not match the logged-in user
121	Option abandonment field error	Invalid field value in option abandonment
122	Option abandonment operation field error	Invalid field value in option abandonment operation (enum value out of range or operation flag is modify, activate, or suspend)
124	The option abandonment has already been canceled	The option abandonment to be operated on has already been deleted
125	Option abandonment not found	The option abandonment to be operated on was not found
149	Option abandonment applications can only be submitted on option expiration day	Trading day is not option expiration day
1005	No record	The contract record corresponding to the option abandonment operation is missing

2.1.52. OnRspQuoteDemand Method

Quote request entry response. This method will be called when the Member System sends a quote request and the Trading System returns a response.

Function Prototype:

```
void OnRspQuoteDemand(
        CShfeFtdcQuoteDemandInfoField* pQuoteDemandInfo,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pQuoteDemandInfo: pointer to the quote request entry response structure. The structure:

```
struct CShfeFtdcQuoteDemandInfoField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Instrument/contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///quote demand local input ID
    TShfeFtdcOrderLocalIDType  QuoteDemandLocalID;
    ///request time
    TShfeFtdcTimeType  DemandTime;
    ///Action day
    TShfeFtdcDateType  ActionDay;
```

```
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in quote request not found
3	Member cannot be found	Member in quote request not found
4	Client cannot be found	Client in quote request not found
15	Client didn't open an account at this member	Client in quote request has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	Contract is untradeable or contract is not in continuous trading status
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in quote request does not match the logged-in user
88	Target user to be operated on not found	User in quote request not found
101	Clearing members are not allowed to trade	Member type in quote request is a clearing member
148	Current market price is within reasonable spread range, and quote request is unnecessary	There are existing buy-side orders, and price has reached the upper limit; There are existing sell-side orders, and price has reached the lower limit; Orders exist on both buy and sell sides, and the price spread is within reasonable range

nRequestID: returns the user quotation entry request ID; this ID is specified by the user upon performing the quotation entry.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.53. OnRtnQuoteDemandNotify Method

Quote request distribution. This method will be called when the Trading System proactively notifies market maker users with corresponding permissions.

Function Prototype:

```
void OnRtnQuoteDemandNotify(
        CShfeFtdcQuoteDemandNotifyField* pQuoteDemandNotify);
```

Parameters:

pQuoteDemandNotify: pointer to the quote request notification. The structure:

```
struct CShfeFtdcQuoteDemandNotifyField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Quote request date
    TShfeFtdcDateType  DemandDay;
    ///Quote request time
    TShfeFtdcTimeType  DemandTime;
    ///Quote request ID
    TShfeFtdcOrderSysIDType  QuoteDemandID;
};
```

2.1.54. OnRspOptionSelfCloseUpdate Method

Option self-hedge update response. This method will be called when the Member System performs an option self-hedge update and the Trading System returns a response.

Function Prototype:

```
void OnRspOptionSelfCloseUpdate(
        CShfeFtdcInputOptionSelfCloseField* pInputOptionSelfClose,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputOptionSelfClose: pointer to the option self-hedge update structure. The structure:

```
struct CShfeFtdcInputOptionSelfCloseField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Whether the futures position generated after option exercise is self-hedged
    TShfeFtdcOptSelfCloseFlagType  SelfCloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract for option self-hedge update not found
3	Member cannot be found	Member for option self-hedge update not found
4	Client cannot be found	Client for option self-hedge update not found
15	Client didn't open an account at this member	The client for option self-hedge update is not opened under the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	Contract trading status is neither continuous trading nor processing trading business
37	Invalid quantity	Invalid quantity in option self-hedge update
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option self-hedge update does not match the logged-in user
79	Order type that is not supported	The Exchange does not support this order type
101	Clearing members are not allowed to trade	Member type requesting option self-hedge update is a clearing member
102	Corresponding clearing member not found	Clearing member corresponding to option self-hedge update's member not found
127	Not within declaration period	The futures position resulting from the option self-hedge exercised by the option seller can only be submitted during the delivery period (exercise window)
137	Option self-hedge field error	Option self-hedge update contains invalid field values (enumeration value out of range)
139	Duplicate option self-hedge update	Duplicate local option self-hedge ID in the option self-hedge update
141	Option self-hedge update is only applicable to options	The contract in the option self-hedge update is not an option contract
144	This client's SelfCloseFlag cannot be retain option position	Only market makers can submit retain option position requests
145	This client's SelfCloseFlag cannot be self-hedge option position	Market makers can only submit retain option position requests
1005	No record	Contract record corresponding to the option self-hedge update is missing

nRequestID: returns the user option self-hedge update request ID; this ID is specified by the user upon performing an option self-hedge update.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.55. OnErrRtnOptionSelfCloseUpdate Method

Option self-hedge update error return. When the Member System performs an option self-hedge update and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnOptionSelfCloseUpdate(
        CShfeFtdcInputOptionSelfCloseField* pInputOptionSelfClose,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pInputOptionSelfClose: pointer to the option self-hedge update structure. The structure:

```
struct CShfeFtdcInputOptionSelfCloseField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Whether the futures position generated after option exercise is self-hedged
    TShfeFtdcOptSelfCloseFlagType  SelfCloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract for option self-hedge update not found
3	Member cannot be found	Member for option self-hedge update not found
4	Client cannot be found	Client for option self-hedge update not found
15	Client didn't open an account at	The client for option self-hedge update is not opened

	this member	under the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	Contract trading status is neither continuous trading nor processing trading business
37	Invalid quantity	Invalid quantity in option self-hedge update
51	Not authorized to trade	No trading permission for specified contract or client for the specified contract or the user
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option self-hedge update does not match the logged-in user
79	Order type that is not supported	The Exchange does not support this order type
101	Clearing members are not allowed to trade	Member type requesting option self-hedge update is a clearing member
102	Corresponding clearing member not found	Clearing member corresponding to option self-hedge update's member not found
127	Not within declaration period	The futures position resulting from the option self-hedge exercised by the option seller can only be submitted during the delivery period (exercise window)
137	Option self-hedge field error	Option self-hedge update contains invalid field values (enumeration value out of range)
139	Duplicate option self-hedge update	Duplicate local option self-hedge ID in the option self-hedge update
141	Option self-hedge update is only applicable to options	The contract in the option self-hedge update is not an option contract
144	This client's SelfCloseFlag cannot be retain option position	Only market makers can submit retain option position requests
145	This client's SelfCloseFlag cannot be self-hedge option position	Market makers can only submit retain option position requests
1005	No record	Contract record corresponding to the option self-hedge update is missing

2.1.56. OnRtnOptionSelfCloseUpdate Method

Option self-hedge update return. When the Member System performs an option self-hedge update resulting in a change to the option self-hedge table, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnRtnOptionSelfCloseUpdate(
        CShfeFtdcOptionSelfCloseField* pOptionSelfClose);
```

Parameters:

pOptionSelfClose: pointer to the option self-hedge structure. The structure:

```
struct CShfeFtdcOptionSelfCloseField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
```

```
        TShfeFtdcSettlementGroupIDType  SettlementGroupID;
        ///Settlement number
        TShfeFtdcSettlementIDType  SettlementID;
        ///Contract number
        TShfeFtdcInstrumentIDType  InstrumentID;
        ///Member ID
        TShfeFtdcParticipantIDType  ParticipantID;
        ///Client ID
        TShfeFtdcClientIDType  ClientID;
        ///Transaction user's ID
        TShfeFtdcUserIDType  UserID;
        ///Local option self-hedge ID
        TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
        ///Volume
        TShfeFtdcVolumeType  Volume;
        ///Whether the futures position generated after option exercise is self-hedged
        TShfeFtdcOptSelfCloseFlagType  SelfCloseFlag;
        ///Business unit
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
        ///Option self-hedge ID
        TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
        ///Option self-hedge result
        TShfeFtdcExecResultType  SelfCloseResult;
        ///order date
        TShfeFtdcDateType  InsertDate;
        ///Entry time
        TShfeFtdcTimeType  InsertTime;
        ///Cancellation time
        TShfeFtdcTimeType  CancelTime;
        ///Settlement member's number
        TShfeFtdcParticipantIDType  ClearingPartID;
        ///Action day
        TShfeFtdcDateType  ActionDay;
    };
```

2.1.57. OnRspOptionSelfCloseAction Method

Option self-hedge operation response, including the cancellation, suspension, activation, and modification of option self-hedge. This method will be called when the Member System performs an option self-hedge operation and the Trading System returns a response.

Function Prototype:

```
    void OnRspOptionSelfCloseAction(
            CShfeFtdcOptionSelfCloseActionField* pOptionSelfCloseAction,
            CShfeFtdcRspInfoField* pRspInfo,
            int nRequestID,
```

```
        bool bIsLast);
```

Parameters:

pOptionSelfCloseAction: pointer to the option self-hedge operation structure. The structure:

```
struct CShfeFtdcOptionSelfCloseActionField{
    ///Option self-hedge ID
    TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Option self-hedge operation flag
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option self-hedge operation not found
3	Member cannot be found	Member in option self-hedge operation not found
4	Client cannot be found	Client in option self-hedge operation not found
15	Client didn't open an account at this member	Client in option self-hedge operation has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither in continuous trading nor in trading business processing state
51	Not authorized to trade	No trading permission for the specified contract, or the client on the specified contract, or the trader
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client

57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option self-hedge operation does not match the logged-in user
138	Option self-hedge operation field error	Invalid field value in option self-hedge operation (enum value out of range or operation flag is modify, activate, or suspend)
140	Option self-hedge update has been canceled	Option self-hedge to be operated on has been deleted
142	Option self-hedge not found	Option self-hedge to be operated on cannot be found
143	Option self-hedge operation must be deletion	Option self-hedge operation type error

nRequestID: returns the user option self-hedge request ID; this ID is specified by the user upon performing an option self-hedge operation.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.58. OnErrRtnOptionSelfCloseAction Method

Option self-hedge operation error return. When the Member System performs an option self-hedge operation and an error occurs, the Trading System will proactively notify the Member System. At this time, this method will be called.

Function Prototype:

```
void OnErrRtnOptionSelfCloseAction(
        CShfeFtdcOptionSelfCloseActionField* pOptionSelfCloseAction,
        CShfeFtdcRspInfoField* pRspInfo);
```

Parameters:

pOptionSelfCloseAction: pointer to the option self-hedge operation structure. The structure:

```
struct CShfeFtdcOptionSelfCloseActionField{
    ///Option self-hedge ID
    TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Option self-hedge operation flag
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Operation of local number
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
```

```
    TShfeFtdcMacAddressType  MacAddress;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
2	Contract cannot be found	Contract in option self-hedge operation not found
3	Member cannot be found	Member in option self-hedge operation not found
4	Client cannot be found	Client in option self-hedge operation not found
15	Client didn't open an account at this member	Client in option self-hedge operation has not opened an account with the specified member
22	The exchange's data is not in the synchronized state	Initialization of Trading System is not completed, please try later
23	The settlement group's data is not in synchronized date	Initialization of Trading System is not completed, please try later
26	This operation is prohibited by current state	The contract trading status is neither in continuous trading nor in trading business processing state
51	Not authorized to trade	No trading permission for the specified contract, or the client on the specified contract, or the trader
53	No such trading role	On the designated contract, member doesn't has the trading role corresponding to such client
57	Operation shall not be conducted by other members	User operating on behalf of a member not associated with them
58	Unmatched user	User in option self-hedge operation does not match the logged-in user
138	Option self-hedge operation field error	Invalid field value in option self-hedge operation (enum value out of range or operation flag is modify, activate, or suspend)
140	Option self-hedge update has been canceled	Option self-hedge to be operated on has been deleted
142	Option self-hedge not found	Option self-hedge to be operated on cannot be found
143	Option self-hedge operation must be deletion	Option self-hedge operation type error

2.1.59. OnRspQryOptionSelfClose Method

Option self-hedge query response. This method will be called when the Member System sends an option self-hedge query request and the Trading System returns a response.

Function Prototype:

```
void OnRspQryOptionSelfClose(
        CShfeFtdcOptionSelfCloseField* pOptionSelfClose,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pOptionSelfClose: pointer to the option self-hedge structure. The structure:

```
struct CShfeFtdcOptionSelfCloseField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Volume
    TShfeFtdcVolumeType  Volume;
    ///Whether the futures position generated after option exercise is self-hedged
    TShfeFtdcOptSelfCloseFlagType  SelfCloseFlag;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
    ///Option self-hedge ID
    TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
    ///Option self-hedge result
    TShfeFtdcExecResultType  SelfCloseResult;
    ///order date
    TShfeFtdcDateType  InsertDate;
    ///Entry time
    TShfeFtdcTimeType  InsertTime;
    ///Cancellation time
    TShfeFtdcTimeType  CancelTime;
    ///Settlement member's number
    TShfeFtdcParticipantIDType  ClearingPartID;
    ///Action day
    TShfeFtdcDateType  ActionDay;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
54	Session Not Found	User Not Logged In
57	Operation shall not be conducted by other members	The conditions under other members cannot be queried
80	User is not authorized to do so	Only trading users are allowed to perform the query; the query can only be performed for a single member

nRequestID: returns the user option self-hedge query request ID; this ID is specified by the user upon submitting the option self-hedge query.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.1.60. OnRspAuthenticate Method

This method is only for proprietary members and is used for authentication before proprietary members collect trading terminal information.

Terminal authentication response. This method will be called when the proprietary Member System performs terminal authentication and the Trading System returns a response.

Function Prototype:

```
void OnRspAuthenticate(
        CShfeFtdcProductAuthField* pProductAuth,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pProductAuth: pointer to the terminal product authentication information structure. The structure:

```
struct CShfeFtdcProductAuthField
{
    ///Trading terminal name
    TShfeFtdcProductInfoType  AppID;
    ///Terminal authentication authorization ID
    TShfeFtdcAuthIDType  AuthID;
};
```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
-1	Authentication failed	Unable to find the authorization ID corresponding to the trading terminal or the authorization ID does not match

nRequestID: returns the user terminal authentication request ID; this ID is specified by the user upon performing terminal authentication.

bIsLast: indicates whether or not this return is the last return regarding nRequestID.

2.2. CShfeFtdcTraderApi Interfaces

The ***CShfeFtdcTraderApi*** interface provides users with functions including order and quote entry, order and quote cancellation, order suspension and activation, order and quote request, trade query, member-client query, member position query, client position query, contract query, contract trading status query, and exchange announcement query.

The Trading System imposes limits on the instruction speed (number of instructions sent per second and number of in-transit instructions) for each seat; and exceeding the limit will result in instruction sending failures. Please consult the relevant department of the Exchange for specific quota number.

2.2.1. CreateFtdcTraderApi Method

This is to create an instance of the ***CShfeFtdcTraderApi***; while this cannot be created with a "new".

Function Prototype:

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

Parameters:

pszFlowPath: Constant character pointer, used to point to a file catalog/directory that stores the status of the bulletin/news sent by the Trading System. The default value is the current catalog/location/directory.

Return Value:

This returns a pointer that point to an instance of the ***CShfeFtdcTraderApi***.

2.2.2. GetVersion Method

This is to get the API version.

Function Prototype:

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

Parameters:

nMajorVersion: returns the main/primary version number

nMinorVersion: returns the minor version number

Return Value:

This returns a constant pointer that points to the versioning identification string.

2.2.3. Release Method

Release the internal resources of the current API instance, exit the API working thread, and set the API exit signal (only sets the exit signal, does not release the instance).

Function Prototype:

```
int Release();
```

Return Value:

0, success

-9 indicates uninitialized.

2.2.4. Init Method

This is to establish the connection between the Member System and the Trading System. After the connection is established, user can proceed to login.

Function Prototype:

```
int Init();
```

Return Value:

0, success

-5 indicates already logged in or repeated invocation.

2.2.5. Join Method

Blocks the API working thread. After the API exit signal is triggered, the current API instance will be released.

Function Prototype:

```
int Join();
```

Return Value:

0, success

2.2.6. GetTradingDay Method

This is to get the current trading day. A correct value will only be retrieved after a successful login to the Trading System.

Function Prototype:

```
const char* GetTradingDay();
```

Return Value:

This returns a constant pointer that points to the date information character string.

2.2.7. RegisterSpi Method

This is to register to an instance derived from ***CShfeFtdcTraderSpi*** instance class. This instance would be used to complete events handling.

Function Prototype:

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

Parameters:

pSpi: realizes/implements the pointer for ShfeFtdcTraderSpi interface instance.

2.2.8. RegisterFront Method

Set the network communication address of the trading front server. The Trading System has multiple trading front servers, and users can register multiple trading front server network communication addresses simultaneously.

Function Prototype:

```
int RegisterFront(const char* pszFrontAddress);
```

Parameters:

pszFrontAddress: pointer to the trading front server's network communication address. The server address is in the format "protocol://ipaddress:port", e.g. "tcp://127.0.0.1:17001". "tcp" in the instance is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represent s the server port number.

Return Value:

0, success

-8, indicates the number of registered front addresses exceeds the maximum value;

-10, indicates already initialized.

2.2.9. RegisterNameServer Method

Set the network communication address of the Trading System's FENS service. The Trading System has multiple FENS services, and users can register multiple FENS service network communication addresses simultaneously.

Function Prototype:

```
int RegisterNameServer(const char* pszNsAddress);
```

Parameters:

pszNsAddress: pointer to the Trading System FENS service network communication address. The network communication address is in the format "protocol://ipaddress:port", e.g. "tcp://127.0.0.1:17001". "tcp" in the instance is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represents the server port number.

Return Value:

0, success

-8, indicates the number of registered FENS service addresses exceeds the maximum value;

-10, indicates already initialized.

2.2.10. SetHeartbeatTimeout Method

This is to set heartbeat timeout limit for network communication. After the connection between ***TraderAPI*** and the TCP of the Trading System is established, it will send regular heartbeat to detect whether the connection is functioning well. This method is used to set the

time for detecting heartbeat timeout. **The Exchange recommends that member systems set the timeout value between 10 and 30 seconds.**

Function Prototype:

```
int SetHeartbeatTimeout(unsigned int timeout);
```

Parameters:

timeout: heartbeat timeout time limit (in seconds). If no information is received from the Trading System for more than timeout/2 seconds, the ***OnHeartBeatWarning*** callback will be triggered. If no information is received from the Trading System for more than timeout seconds, the connection will be disconnected, triggering the ***OnFrontDisconnected*** callback.

Please refer to Part I Section 4.9 for the heartbeat mechanism

Return Value:

0, success

-10, indicates already initialized.

2.2.11. OpenRequestLog Method

This is to open the request log file. After this method is called, all request information sent to the Trading System will be recorded in the specified log files.

Function Prototype:

```
int OpenRequestLog(const char* pszReqLogFileName);
```

Parameters:

pszReqLogFileName: the request log file name.

Return Value:

0, success

-4, indicates failure to open log file.

2.2.12. OpenResponseLog Method

This is to open the reply log file. After the method is called, all information returned from the Trading System will be recorded in the specified log file, including reply message and return message.

Function Prototype:

```
int OpenResponseLog(const char* pszRspLogFileName);
```

Parameters:

pszRspLogFileName: reply log file name.

Return Value:

0, success

-4, indicates failure to open log file.

2.2.13. SubscribePrivateTopic Method

This is to subscribe to member-specific private stream. After a successful subscription, the Trading System will proactively send the member private stream or trader private stream to the Member System based on subscription permissions.

Function Prototype:

```
int SubscribePrivateTopic(TERESUMETYPE nResumeType);
```

Parameters:

nResumeType: Member private stream retransmission mode:

TERT_RESTART: to re-transmit from current trading day

TERT_RESUME: resume from where it last left off. **To ensure the integrity of member trading data, the Exchange recommends using this mode to receive the member private stream, and to process subsequent order business only after restoring the member's trading data for the day.**

TERT_QUICK: to only transmit those post-current-login member-specific private stream contents. **To ensure the integrity of member trading data, the Exchange does not recommend using this method to receive the private stream.**

Return Value:

0, success

-10, indicates already initialized.

2.2.14. SubscribePublicTopic Method

This is to subscribe to public stream. After a successful subscription, the Trading System will proactively send the public stream to the Member System.

Function Prototype:

```
int SubscribePublicTopic(TE_RESUME_TYPE nResumeType);
```

Parameters:

nResumeType: public stream re-transmission method types:

TERT_RESTART: to re-transmit from current trading day

TERT_RESUME: to re-transmit by resuming and continuing from last transmission

TERT_QUICK: to only transmit those post-current-login member-specific private stream contents

Return Value:

- 0, success
- -10, indicates already initialized.

2.2.15. SubscribeUserTopic Method

This is to subscribe to trader-specific private stream. After a successful subscription, the Trading System will proactively send the trader private stream to the Member System.

Function Prototype:

```
int SubscribeUserTopic(TE_RESUME_TYPE nResumeType);
```

Parameters:

nResumeType: Trader private stream retransmission mode,

TERT_RESTART: to re-transmit from current trading day

TERT_RESUME: resume from where it last left off. **To ensure the integrity of member trading data, the Exchange recommends using this mode to receive the trader private stream, and to process subsequent order business only after restoring both the member's and the trader's trading data for the day.**

TERT_QUICK: only sends trader private stream content after login. **The Exchange does not recommend using this mode to receive trader private streams to ensure the integrity of members' trading data.**

Return Value:

0, success

-10, indicates already initialized.

2.2.16. ReqUserLogin Method

User login request.

Function Prototype:

```
int ReqUserLogin(
        CShfeFtdcReqUserLoginField* pReqUserLoginField,
        int nRequestID);
```

Parameters:

pReqUserLoginField: pointer to the login request structure. The structure:

```
struct CShfeFtdcReqUserLoginField {
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Password
    TShfeFtdcPasswordType  Password;
    ///The user-end product information
    TShfeFtdcProductInfoType  UserProductInfo;
    ///The interface-port product information
    TShfeFtdcProductInfoType  InterfaceProductInfo;
    ///Protocol information
    TShfeFtdcProtocolInfoType  ProtocolInfo;
    ///Datacenter ID
    TShfeFtdcDataCenterIDType  DataCenterID;
};
```

User is required to fill the field of "UserProductInfo", i.e., product information of Member System such as software developer and version number For instance, "SFIT Trader V100" represents the trading program and version number developed by technology firm.

nRequestID: returns the user login request ID; this ID is specified and managed by the user.

Return Value:

0, success
-2, indicates exceeding in-transit transaction flow control;
-3, indicates exceeding transaction request flow control;
-5, indicates already logged in or repeated invocation;
-6, indicates a required field is empty[1] (UserProductInfo not filled in);
-7, indicates authentication is enabled but authentication failed;
-9, indicates uninitialized;
-12, indicates connection to the front server has not yet been established.

2.2.17. ReqUserLogout Method

User logout request.

Function Prototype:

```
int ReqUserLogout(
        CShfeFtdcReqUserLogoutField* pReqUserLogout,
        int nRequestID);
```

Parameter:

pReqUserLogout: pointer to the logout request structure. The structure:

```
struct CShfeFtdcReqUserLogoutField {
    ///Trading User ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
};
```

nRequestID: returns the user logout request ID; this ID is specified and managed by the user.

Returned Value:

0, success
-1, indicates not logged in;
-2, indicates exceeding in-transit transaction flow control;
-3, indicates exceeding the transaction request flow control.

2.2.18. ReqUserPasswordUpdate Method

This is the user password update request.

Function Prototype:

```
int ReqUserPasswordUpdate(
        CShfeFtdcUserPasswordUpdateField* pUserPasswordUpdate,
        int nRequestID);
```

Parameter:

pUserPasswordUpdate: pointer to the user password modification structure. The structure:

[1] Note: Empty string definition: Strings containing only spaces or no characters; the same applies below.

```
struct CShfeFtdcUserPasswordUpdateField {
    ///Trading User ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Old Password
    TShfeFtdcPasswordType  OldPassword;
    ///New Password
    TShfeFtdcPasswordType  NewPassword;
};
```

nRequestID: returns the user password modification request ID; this ID is specified and managed by the user.

Returned Value:

0, success

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding the transaction request flow control;

-13, indicates a member ID mismatch;

-14, indicates a user ID mismatch.

2.2.19. ReqSubscribeTopic Method

Topic subscription request. It will be called after successful login.

Function Prototype:

```
int ReqSubscribeTopic(
        CShfeFtdcDisseminationField* pDissemination,
        int nRequestID);
```

Parameter:

pDissemination: pointer to the subscribed topic structure, including topic to be subscribed as well as the starting message sequence number. The structure:

```
struct CShfeFtdcDisseminationField {
    ///Sequence series number
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
};
SequenceSeries: topics to be subscribed
SequenceNo: A value less than 0 indicates using TERT_QUICK mode; other values
specify the sequence number to resume from
```

nRequestID: returns the user topic subscription request ID; this ID is specified and managed by the user.

Returned Value:

0, success

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding the transaction request flow control.

2.2.20. ReqQryTopic Method

This is the request for querying topic/theme. It will be called after successful login.

Function Prototype:

```
int ReqQryTopic(
        CShfeFtdcDisseminationField* pDissemination,
        int nRequestID);
```

Parameter:

pDissemination: pointer to the topic query structure, including topic to be queried. The structure:

```
struct CShfeFtdcDisseminationField {
    ///Serial series number: Fill in the topic number to query
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Sequence number, unused field
    TShfeFtdcSequenceNoType  SequenceNo;
};
```

nRequestID: returns the user topic query request ID; this ID is specified and managed by the user.

Returned Value:

0, success

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.21. ReqOrderInsert Method

Order entry request.

Function Prototype:

```
int ReqOrderInsert(
        CShfeFtdcInputOrderField* pInputOrder,
        int nRequestID);
```

Parameter:

pInputOrder: pointer to the order entry structure. The structure:

```
struct CShfeFtdcInputOrderField {
    ///Order number; this field will be returned by Trading System.
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Conditions of order price
    TShfeFtdcOrderPriceTypeType  OrderPriceType;
```

```
        ///Buy-sell direction
        TShfeFtdcDirectionType  Direction;
        ///Combination offset flag
        TShfeFtdcCombOffsetFlagType  CombOffsetFlag;
        ///Combination hedge flag
        TShfeFtdcCombHedgeFlagType  CombHedgeFlag;
        ///Price
        TShfeFtdcPriceType  LimitPrice;
        ///Quantity
        TShfeFtdcVolumeType  VolumeTotalOriginal;
        ///Type of valid period
        TShfeFtdcTimeConditionType  TimeCondition;
        ///GTD DATE, not used
        TShfeFtdcDateType  GTDDate;
        ///Volume type;
        TShfeFtdcVolumeConditionType  VolumeCondition;
        ///The Min.volume, used when the VolumeCondition is set as “minimum quality”
        TShfeFtdcVolumeType  MinVolume;
        ///Trigger conditions
        TShfeFtdcContingentConditionType  ContingentCondition;
        ///Stop-loss price, not used
        TShfeFtdcPriceType  StopPrice;
        ///Reasons for forced closing-out
        TShfeFtdcForceCloseReasonType  ForceCloseReason;
        ///Local order number*
        TShfeFtdcOrderLocalIDType  OrderLocalID;
        ///Flag of auto-suspension
        TShfeFtdcBoolType  IsAutoSuspend;
        ///Business unit, not used
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
    };
    * OrderLocalID: Local order identifier, must increment sequentially (compared as
strings). After each successful login, the maximum local order ID used for the seat on the
current day can be obtained from the MaxOrderLocalID field of the
CShfeFtdcRspUserLoginField output parameter in OnRspUserLogin.
```

nRequestID: returns the user order entry request ID; this ID is specified and managed by the user. This ID can be reused within the same session.

Returned Value:

0, success

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (OrderLocalID is empty);

-11, indicates duplicate ID (OrderLocalID not incrementing as required[2]).

Business Description:

The current Trading System supports the following order types:

Price Condition OrderPriceType	Time Condition TimeCondition	Volume Condition VolumeCondition	Trigger Condition ContingentCondition
Limit Price	Day Order	Any Volume (AV)	Immediate
Market Price			
Limit Price	Complete immediately, otherwise cancel	Any Volume (AV)	Immediate
		Minimum Volume (MV)	
Market Price		Fill-Or-Kill (CV)	

2.2.22. ReqOrderAction Method

Order action requests, including order cancellation, suspension, activation, and modification.

Function Prototype:

```
int ReqOrderAction(
        CShfeFtdcOrderActionField* pOrderAction,
        int nRequestID);
```

Parameter:

pOrderAction: pointer to the order operation structure. The structure:

```
struct CShfeFtdcOrderActionField {
    ///Order number*
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Local order number*
    TShfeFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Price, not used
    TShfeFtdcPriceType  LimitPrice;
    ///Local number of operation*
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Change in quantity, not used
    TShfeFtdcVolumeType  VolumeChange;
    ///Business unit, not used
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Local business ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
```

[2] Explanation of Incrementing: Requests involving local IDs share a common LocalID sequence. For non-empty local ID strings, the LocalID sequence must increment; the same applies below.

```
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
    };
    * OrderSysID and OrderLocalID means that either of the target order to operated can
be filled.
    * ActionLocalID: Local operation ID; if non-empty, must increment sequentially.
```

nRequestID: returns the user order action request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required fields are empty (both OrderSysID and OrderLocalID are empty);

-11, indicates duplicate ID (ActionLocalID not incrementing as required).

Business Description:

Order modification functionality is not currently supported.

2.2.23. ReqQuoteInsert Method

Quote entry request.

Function Prototype:

```
int ReqQuoteInsert(
        CShfeFtdcInputQuoteField* pInputQuote,
        int nRequestID);
```

Parameter:

pInputQuote: pointer to the quote entry structure. The structure:

```
struct CShfeFtdcInputQuoteField {
    ///Quote number,this field will be returned by Trading System.
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Quantity
    TShfeFtdcVolumeType  Volume;
    ///Contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Business unit, not used
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Buyer's combination offset flag
```

```
        TShfeFtdcCombOffsetFlagType  BidCombOffsetFlag;
        ///Buyer's combination hedge flag
        TShfeFtdcCombHedgeFlagType  BidCombHedgeFlag;
        ///Buyer's price
        TShfeFtdcPriceType  BidPrice;
        ///Seller's combination offset flag
        TShfeFtdcCombOffsetFlagType  AskCombOffsetFlag;
        ///Seller's combination hedge flag
        TShfeFtdcCombHedgeFlagType  AskCombHedgeFlag;
        ///Seller's price
        TShfeFtdcPriceType  AskPrice;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
        ///Quote request ID
        TShfeFtdcOrderSysIDType  QuoteDemandID;
    };
```

nRequestID: returns the user quote request ID; this ID is designated and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (QuoteLocalID is empty);

-11, indicates duplicate ID (QuoteLocalID not incrementing as required).

2.2.24. ReqQuoteAction Method

Quote action requests, including quote cancellation, suspension, activation, and modification.

Function Prototype:

```
int ReqQuoteAction(
        CShfeFtdcQuoteActionField* pQuoteAction,
        int nRequestID);
```

Parameter:

pQuoteAction: pointer to the quote operation structure. The structure:

```
struct CShfeFtdcQuoteActionField {
    ///Quoto number
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Local quoto number
    TShfeFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
```

```
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    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;
};
```

nRequestID: returns the user quote request ID; this ID is designated and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required fields are empty (both QuoteSysID and QuoteLocalID are empty);

-11, indicates duplicate ID (ActionLocalID not incrementing as required).

Business Description:

Currently, only quote cancellation is supported.

2.2.25. ReqExecOrderInsert Method

Execution declaration entry request. Only option buyers are allowed to submit option exercise requests.

Function Prototype:

```
int ReqExecOrderInsert(
        CShfeFtdcInputExecOrderField* pInputExecOrder,
        int nRequestID);
```

Parameter:

pInputExecOrder: pointer to the input option exercise structure. The structure:

```
struct CShfeFtdcInputExecOrderField {
    ///Contract number
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
```

```
        ///Transaction user's ID
        TShfeFtdcUserIDType  UserID;
        ///Local option exercise number
        TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
        ///Quantity
        TShfeFtdcVolumeType  Volume;
        ///Offset flag
        TShfeFtdcOffsetFlagType  OffsetFlag;
        ///Hedge flag
        TShfeFtdcHedgeFlagType  HedgeFlag;
        ///position direction, i.e. whether buyer(long position) or seller(short position)
        made this application
        TShfeFtdcPosiDirectionType  PosiDirection;
        ///Flag indicating whether to retain futures positions after option exercise, not
        used
        TShfeFtdcExecOrderPositionFlagType  ReservePositionFlag;
        ///Whether the futures positions generated after option exercise are self-hedged
        TShfeFtdcExecOrderCloseFlagType  CloseFlag;
        ///Business unit
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///Local business ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
    };
```

nRequestID: returns the user option exercise entry request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (ExecOrderLocalID is empty);

-11, indicates duplicate ID (ExecOrderLocalID not incrementing as required).

2.2.26. ReqExecOrderAction Method

Option exercise request.

Function Prototype:

```
int ReqExecOrderAction(
        CShfeFtdcExecOrderActionField* pExecOrderAction,
        int nRequestID);
```

Parameter:

pExecOrderAction: pointer to the option exercise operation structure. The structure:

```
struct CShfeFtdcExecOrderActionField {
    ///Option exercise number
```

```
        TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
        ///Local option exercise number
        TShfeFtdcOrderLocalIDType  ExecOrderLocalID;
        ///Flag of order operation
        TShfeFtdcActionFlagType  ActionFlag;
        ///Member ID
        TShfeFtdcParticipantIDType  ParticipantID;
        ///Client ID
        TShfeFtdcClientIDType  ClientID;
        ///Transaction user's ID
        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;
    };
```

nRequestID: returns the user option exercise action request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required fields are empty (both ExecOrderSysID and ExecOrderLocalID are empty);

-11, indicates duplicate ID (ActionLocalID not incrementing as required).

Business Description:

Currently, only option exercise cancellation is supported.

2.2.27. ReqQryPartAccount Method

Member funds query request.

Function Prototype:

```
int ReqQryPartAccount(
        CShfeFtdcQryPartAccountField* pQryPartAccount,
        int nRequestID);
```

Parameter:

pQryPartAccount: pointer to the member fund query structure. The structure:

```
struct CShfeFtdcQryPartAccountField {
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
```

```
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Fund account, optional
    TShfeFtdcAccountIDType  AccountID;
};
```

nRequestID: returns the user member funds query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.28. ReqQryOrder Method

This is for order query request.

Function Prototype:

```
int ReqQryOrder(
        CShfeFtdcQryOrderField* pQryOrder,
        int nRequestID);
```

Parameter:

pQryOrder: pointer to the order query structure. The query conditions are related. If an optional query condition is empty, that query condition will be ignored. The structure:

```
struct CShfeFtdcQryOrderField {
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Order number, optional
    TShfeFtdcOrderSysIDType  OrderSysID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Client ID, optional
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID, optional
    TShfeFtdcUserIDType  UserID;
    ///The starting time, optional
    TShfeFtdcTimeType  TimeStart;
    ///The finishing time, optional
    TShfeFtdcTimeType TimeEnd;
};
```

nRequestID: returns the user order query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.29. ReqQryQuote Method

Quote request request.

Function Prototype:

```
int ReqQryQuote(
        CShfeFtdcQryQuoteField* pQryQuote,
        int nRequestID);
```

Parameter:

pQryQuote: pointer to the quote request structure. The structure:

```
struct CShfeFtdcQryQuoteField {
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Quote No, optional
    TShfeFtdcQuoteSysIDType  QuoteSysID;
    ///Client ID, optional
    TShfeFtdcClientIDType  ClientID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Transaction user's ID, optional
    TShfeFtdcUserIDType UserID;
};
```

nRequestID: returns the user quote request ID; this ID is designated and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.30. ReqQryTrade Method

This is the request for trade query (matched/filled order query).

Function Prototype:

```
int ReqQryTrade(
        CShfeFtdcQryTradeField* pQryTrade,
        int nRequestID);
```

Parameter:

pQryTrade: pointer to the trade query (i.e. filled/matched order) structure. The structure:

```
struct CShfeFtdcQryTradeField {
```

```
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///The starting contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDStart;
    ///The ending contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDEnd;
    ///Transaction number, optional
    TShfeFtdcTradeIDType  TradeID;
    ///Client ID, optional
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID, optional
    TShfeFtdcUserIDType  UserID;
    ///The starting time, optional
    TShfeFtdcTimeType  TimeStart;
    ///The finishing time, optional
    TShfeFtdcTimeType  TimeEnd;
};
```

NRequestID: returns the user trade query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.31. ReqQryClient Method

This is for member client query request.

Function Prototype:

```
int ReqQryClient(
        CShfeFtdcQryClientField* pQryClient,
        int nRequestID);
```

Parameter:

pQryClient: pointer to the client query structure. The structure:

```
struct CShfeFtdcQryClientField {
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///The starting client ID, optional
    TShfeFtdcClientIDType  ClientIDStart;
    ///The ending client ID, optional
    TShfeFtdcClientIDType  ClientIDEnd;
};
```

nRequestID: returns the user client query request ID; this ID is specified and managed

by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.32. ReqQryPartPosition Method

Member position query request.

Function Prototype:

```
int ReqQryPartPosition(
        CShfeFtdcQryPartPositionField* pQryPartPosition,
        int nRequestID);
```

Parameter:

pQryPartPosition: pointer to the member position query structure. The structure:

```
struct CShfeFtdcQryPartPositionField {
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///The starting contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDStart;
    ///The ending contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDEnd;
};
```

nRequestID: returns the user member position query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.33. ReqQryClientPosition Method

Client position query request.

Function Prototype:

```
int ReqQryClientPosition(
        CShfeFtdcQryClientPositionField* pQryClientPosition,
        int nRequestID);
```

Parameter:

pQryClientPosition: pointer to the client position query structure. The structure:

```
struct CShfeFtdcQryClientPositionField {
```

```
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///The starting client ID, optional
    TShfeFtdcClientIDType  ClientIDStart;
    ///The ending client ID, optional
    TShfeFtdcClientIDType  ClientIDEnd;
    ///The starting contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDStart;
    ///The ending contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDEnd;
    ///Type of client, optional
    TShfeFtdcClientTypeType  ClientType;
};
```

nRequestID: returns the user client position query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.34. ReqQryInstrument Method

Instrument/contract query request.

Function Prototype:

```
int ReqQryInstrument(
        CShfeFtdcQryInstrumentField* pQryInstrument,
        int nRequestID);
```

Parameter:

pQryInstrument: pointer to the contract query structure. The structure:

```
struct CShfeFtdcQryInstrumentField {
    ///Settlement group's ID, optional
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Product suite's ID, optional
    TShfeFtdcProductGroupIDType  ProductGroupID;
    ///Product ID, optional
    TShfeFtdcProductIDType  ProductID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
};
```

nRequestID: returns the user contract query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;
-2, indicates exceeding the pending query flow control;
-3, indicates exceeding the query request flow control.

2.2.35. ReqQryInstrumentStatus Method

Contract trading status query request.

Function Prototype:

```
int ReqQryInstrumentStatus(
        CShfeFtdcQryInstrumentStatusField* pQryInstrumentStatus,
        int nRequestID);
```

Parameter:

pQryInstrumentStatus: pointer to the contract trading status query structure. The structure:

```
struct CShfeFtdcQryInstrumentStatusField {
    ///The starting contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDStart;
    ///The ending contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDEnd;
};
```

nRequestID: returns the user contract status query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful
-1, indicates not logged in;
-2, indicates exceeding the pending query flow control;
-3, indicates exceeding the query request flow control.

2.2.36. ReqQryMarketData Method

General market data query request.

Function Prototype:

```
int ReqQryMarketData(
        CShfeFtdcQryMarketDataField* pQryMarketData,
        int nRequestID);
```

Parameter:

pQryMarketData: pointer to the market data query structure. The structure:

```
struct CShfeFtdcQryMarketDataField {
    ///Product ID, optional
    TShfeFtdcProductIDType  ProductID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID
};
```

nRequestID: returns the user general market data query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.37. ReqQryBulletin Method

Exchange bulletin query request.

Function Prototype:

```
int ReqQryBulletin(
        CShfeFtdcQryBulletinField* pQryBulletin,
        int nRequestID);
```

Parameter:

pQryBulletin: pointer to the Exchange announcement query structure. The structure:

```
struct CShfeFtdcQryBulletinField {
    ///Trading Day, Optional
    TShfeFtdcDateType  TradingDay;
    ///market ID, optional
    TShfeFtdcMarketIDType  MarketID;
    ///bulletin ID, optional
    TShfeFtdcBulletinIDType  BulletinID;
    ///bulletin type, optional
    TShfeFtdcNewsTypeType  NewsType;
    ///urgency level, optional
    TShfeFtdcNewsUrgencyType  NewsUrgency;
};
```

nRequestID: returns the user announcement query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.38. ReqQryHedgeVolume Method

Hedge quota query request.

Function Prototype:

```
int ReqQryHedgeVolume(
        CShfeFtdcQryHedgeVolumeField* pQryHedgeVolume,
        int nRequestID);
```

Parameter:

pQryHedgeVolume: pointer to the hedge quota query structure. The structure:

```
struct CshfeFtdcQryHedgeVolumeField {
{
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///The starting client ID, optional
    TShfeFtdcClientIDType  ClientIDStart;
    ///The ending client ID, optional
    TShfeFtdcClientIDType  ClientIDEnd;
    ///The starting contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDStart;
    ///The ending contract ID, optional
    TShfeFtdcInstrumentIDType  InstIDEnd;
};
```

nRequestID: returns the user hedge quota query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

Business Description:

This feature is currently unsupported.

2.2.39. ReqQryExecOrder Method

Execution declaration query request.

Function Prototype:

```
int ReqQryExecOrder(
        CShfeFtdcQryExecOrderField* pQryExecOrder,
        int nRequestID);
```

Parameter:

pQryExecOrder: pointer to the option exercise query structure. The structure:

```
struct CShfeFtdcQryExecOrderField
{
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Option exercise number, optional
    TShfeFtdcExecOrderSysIDType  ExecOrderSysID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///client ID, optional
```

```
        TShfeFtdcClientIDType  ClientID;
        ///transaction user's ID, optional
        TShfeFtdcUserIDType  UserID;
        ///The starting time, optional
        TShfeFtdcTimeType  TimeStart;
        ///The finishing time, optional
        TShfeFtdcTimeType  TimeEnd;
};
```

nRequestID: returns the user option exercise query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.40. ReqQryExchangeRate Method

This function is used to perform the exchange rate query.

Function Prototype:

```
int ReqQryExchangeRate(
        CShfeFtdcQryExchangeRateField* pQryExchangeRate,
        int nRequestID);
```

Parameter:

pQryExchangeRate: pointer to the exchange rate query structure. The structure:

```
struct CShfeFtdcQryExchangeRateField
{
    ///Currency ID
    TShfeFtdcCurrencyIDType  CurrencyID;
};
```

nRequestID: returns the user exchange rate query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.41. ReqAbandonExecOrderInsert Method

Option abandonment entry request. Only option buyers have the right to abandon exercise.

Function Prototype:

```
int ReqAbandonExecOrderInsert(
```

```
        CShfeFtdcInputAbandonExecOrderField* pInputAbandonExecOrder,
        int nRequestID);
```

Parameter:

pInputAbandonExecOrder: pointer to the option abandonment entry structure. The structure:

```
struct CShfeFtdcInputAbandonExecOrderField {
    ///Contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Option abandonment local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Quantity
    TShfeFtdcVolumeType  Volume;
    ///Offset flag
    TShfeFtdcOffsetFlagType  OffsetFlag;
    ///Hedge flag
    TShfeFtdcHedgeFlagType  HedgeFlag;
    ///Position direction that apply for abandon, only long position could apply for
    abandon actually
    TShfeFtdcPosiDirectionType  PosiDirection;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Business local ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

nRequestID: returns the user option abandonment entry request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (AbandonExecOrderLocalID is empty);

-11, indicates duplicate ID (AbandonExecOrderLocalID not incrementing as required).

2.2.42. ReqAbandonExecOrderAction Method

Option abandonment request.

Function Prototype:

```
int ReqAbandonExecOrderAction(
        CShfeFtdcAbandonExecOrderActionField* pAbandonExecOrderAction,
        int nRequestID);
```

Parameter:

pAbandonExecOrderAction: pointer to the option abandonment structure. The structure:

```
struct CShfeFtdcAbandonExecOrderActionField {
    ///Option abandonment ID
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///Option abandonment local ID
    TShfeFtdcOrderLocalIDType  AbandonExecOrderLocalID;
    ///Flag of order operation
    TShfeFtdcActionFlagType  ActionFlag;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Client ID
    TShfeFtdcClientIDType  ClientID;
    ///Transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local number of operation
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///Business local ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

nRequestID: returns the user option abandonment action request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required fields are empty (both AbandonExecOrderLocalID and AbandonExecOrderSysID are empty);

-11, indicates duplicate ID (ActionLocalID not incrementing as required).

Business Description:

Currently, only abandon cancellation is supported.

2.2.43. ReqQryAbandonExecOrder Method

Request of option abandonment query.

Function Prototype:

```
int ReqQryAbandonExecOrder(
        CShfeFtdcQryAbandonExecOrderField* pQryAbandonExecOrder,
        int nRequestID);
```

Parameter:

pQryAbandonExecOrder: pointer to the option abandonment query structure. The structure:

```
struct CShfeFtdcQryAbandonExecOrderField
{
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Option abandonment ID, optional
    TShfeFtdcExecOrderSysIDType  AbandonExecOrderSysID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///client ID, optional
    TShfeFtdcClientIDType  ClientID;
    ///transaction user's ID, optional
    TShfeFtdcUserIDType  UserID;
    ///The starting time, optional
    TShfeFtdcTimeType  TimeStart;
    ///The finishing time, optional
    TShfeFtdcTimeType  TimeEnd;
};
```

nRequestID: returns the user option abandonment query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.44. ReqQuoteDemand Method

Request of quote request entry.

Function Prototype:

```
int ReqQuoteDemand(
        CShfeFtdcInputQuoteDemandField* pInputQuoteDemand,
        int nRequestID);
```

Parameter:

pInputQuoteDemand: pointer to the quote demand entry request structure. The structure:

```
struct CShfeFtdcInputQuoteDemandField
```

```
{
    ///member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///client ID
    TShfeFtdcClientIDType  ClientID;
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///quote demand local input ID
    TShfeFtdcOrderLocalIDType  QuoteDemandLocalID;
};
```

nRequestID: returns the user quote demand entry request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (QuoteDemandLocalID is empty).

2.2.45. ReqOptionSelfCloseUpdate Method

Option self-hedge update request. Ordinary clients can apply for self-hedge option positions; option sellers can apply for self-hedge futures positions arising from exercise; and option market makers can apply to retain option positions. For option self-hedge updates, only the latest request is kept for identical member, client, contract, and self-hedge type combinations.

Function Prototype:

```
int ReqOptionSelfCloseUpdate(
        CShfeFtdcInputOptionSelfCloseField* pInputOptionSelfClose,
        int nRequestID);
```

Parameter:

pInputOptionSelfClose: pointer to the option self-hedge update structure. The structure:

```
struct CShfeFtdcInputOptionSelfCloseField {
    ///contract ID
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///client ID
    TShfeFtdcClientIDType  ClientID;
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Quantity
    TShfeFtdcVolumeType  Volume;
```

```
        ///Whether the futures position generated after option exercise is self-hedged
        TShfeFtdcOptSelfCloseFlagType  SelfCloseFlag;
        ///business unit
        TShfeFtdcBusinessUnitType  BusinessUnit;
        ///business local ID
        TShfeFtdcBusinessLocalIDType  BusinessLocalID;
        ///IP address
        TShfeFtdcIPAddressType  IPAddress;
        ///Mac address
        TShfeFtdcMacAddressType  MacAddress;
    };
```

nRequestID: returns the user option self-hedge update request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required field is empty (***OptionSelfCloseLocalID*** is empty);

-11, indicates duplicate ID (***OptionSelfCloseLocalID*** not incrementing as required).

2.2.46. ReqOptionSelfCloseAction Method

Option self-hedge action request.

Function Prototype:

```
int ReqOptionSelfCloseAction(
        CShfeFtdcOptionSelfCloseActionField* pOptionSelfCloseAction,
        int nRequestID);
```

Parameter:

pOptionSelfCloseAction: pointer to the option self-hedge operation structure. The structure:

```
struct CShfeFtdcOptionSelfCloseActionField {
    ///Option self-hedge ID
    TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
    ///Local option self-hedge ID
    TShfeFtdcOrderLocalIDType  OptionSelfCloseLocalID;
    ///Option self-hedge operation flag
    TShfeFtdcActionFlagType  ActionFlag;
    ///member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///client ID
    TShfeFtdcClientIDType  ClientID;
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Local number of operation
    TShfeFtdcOrderLocalIDType  ActionLocalID;
    ///business unit
```

```
    TShfeFtdcBusinessUnitType  BusinessUnit;
    ///business local ID
    TShfeFtdcBusinessLocalIDType  BusinessLocalID;
    ///IP address
    TShfeFtdcIPAddressType  IPAddress;
    ///Mac address
    TShfeFtdcMacAddressType  MacAddress;
};
```

nRequestID: returns the user option self-hedge action request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-6, indicates required fields are empty (both OptionSelfCloseLocalID and OptionSelfCloseSysID are empty);

-11, indicates duplicate ID (ActionLocalID not incrementing as required).

Business Description:

Currently, only option self-hedge cancellations are supported.

2.2.47. ReqQryOptionSelfClose Method

Option self-hedge query request.

Function Prototype:

```
int ReqQryOptionSelfClose(
        CShfeFtdcQryOptionSelfCloseField* pQryOptionSelfClose,
        int nRequestID);
```

Parameter:

pQryOptionSelfClose: pointer to the option self-hedge query structure. The structure:

```
struct CShfeFtdcQryOptionSelfCloseField
{
    ///The starting member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDStart;
    ///The ending member ID can only represent this member
    TShfeFtdcParticipantIDType  PartIDEnd;
    ///Option self-hedge ID, optional
    TShfeFtdcOptionSelfCloseSysIDType  OptionSelfCloseSysID;
    ///Contract ID, optional
    TShfeFtdcInstrumentIDType  InstrumentID;
    ///client ID, optional
    TShfeFtdcClientIDType  ClientID;
    ///transaction user's ID, optional
    TShfeFtdcUserIDType  UserID;
    ///The starting time, optional
    TShfeFtdcTimeType  TimeStart;
    ///The finishing time, optional
```

```
    TShfeFtdcTimeType  TimeEnd;
};
```

nRequestID: returns the user option self-hedge query request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding the pending query flow control;

-3, indicates exceeding the query request flow control.

2.2.48. ReqAuthenticate Method

This method is only for proprietary members and is used for authentication before proprietary members collect trading terminal information.

Terminal authentication request.

Function Prototype:

```
int ReqAuthenticate(
        CShfeFtdcProductAuthField* pProductAuth,
        int nRequestID);
```

Parameter:

pProductAuth: pointer to the terminal product authentication information structure. The structure:

```
struct CShfeFtdcProductAuthField
{
    ///Trading terminal name
    TShfeFtdcProductInfoType  AppID;
    ///Terminal authentication authorization ID
    TShfeFtdcAuthIDType  AuthID;
};
```

nRequestID: returns the user terminal product authentication information request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-2, indicates exceeding in-transit transaction flow control;

-3, indicates exceeding transaction request flow control;

-5, indicates already logged in or duplicate invocation (not allowed after login or repeated invocation);

-9, indicates uninitialized;

-12, indicates connection to the front server has not yet been established.

3. TraderAPI Interface Development Instances

```
// tradeapitest.cpp :
```

```
// A simple instance that describes the use of interface for CShfeFtdcTraderApi and
CShfeFtdcTraderSpi.
// This instance shows the process of order entry operation

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "FtdcTraderApi.h"

class CSimpleHandler : public CShfeFtdcTraderSpi
{
public:
    // Constructed function that needs an effective pointer pointing to the
CShfeFtdcMduserApi instance
    CSimpleHandler(CShfeFtdcTraderApi *pTraderApi) : m_pTraderApi(pTraderApi){}

    ~CSimpleHandler() {}

    // Member System needs to complete the login step when it has created
communication connection with Trading System
    virtual void OnFrontConnected()
    {
      CShfeFtdcReqUserLoginField reqUserLogin;
      memset(&reqUserLogin, 0, sizeof(reqUserLogin));
      strcpy(reqUserLogin.ParticipantID, "0888");
      strcpy(reqUserLogin.UserID, "0888c1c" );
      strcpy(reqUserLogin.Password, "1" );
      strcpy(reqUserLogin.UserProductInfo, "Test TraderAPI v2.00");
      // Send the login request
      int ret = m_pTraderApi ->ReqUserLogin(&reqUserLogin, 0);
      if (ret != 0)
      {
        printf("ReqUserLogin Fail ret = %d\n", ret);
        exit(-1);
      }
    }

    // This method will be called when Member System disconnect its communication with
Trading System
    virtual void OnFrontDisconnected(int nReason)
    {
      // In this case, API will automatically conduct reconnection while Member System
may do nothing
      printf("OnFrontDisconnected Reason = %#x.\n", nReason);
    }

    // After Member System sent the login request, this method will be called to notify
Member System of whether this login is successful or not
    virtual void OnRspUserLogin(CShfeFtdcRspUserLoginField *pRspUserLogin,
CShfeFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
    {
      printf("OnRspUserLogin:\n");
      printf("ErrorID=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
      printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
```

```
        if (pRspInfo->ErrorID != 0) {
            // In case of login failure, Member System will be required to conduct the error-
processing
            printf("Failed to login, errorID=%d errormsg=%s requestid=%d chain=%d",
pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
            exit(-1);
        }

        // In case of successful login, the order entry  request will be sent
        CShfeFtdcInputOrderField ord;
        memset(&ord, 0, sizeof(ord));

        // Member ID
        strcpy(ord.ParticipantID, "0888" );
        // Client ID
        strcpy(ord.ClientID, "08880001");
        // Transaction user's ID
        strcpy(ord.UserID, "0888c1c" );
        // Contract ID
        strcpy(ord.InstrumentID, "cu2511");
        // Conditions of order price
        ord.OrderPriceType = SHFE_FTDC_OPT_LimitPrice;
        // Buy-sell direction
        ord.Direction = SHFE_FTDC_D_Buy;
        // Combination offset flag
        strcpy(ord.CombOffsetFlag, "0");
        // Combination hedge flag
        strcpy(ord.CombHedgeFlag, "1");
        // Price
        ord.LimitPrice = 74000;
        // Quantity
        ord.VolumeTotalOriginal = 10;
        // Type of valid period
        ord.TimeCondition = SHFE_FTDC_TC_GFD;
        // GTD date
        strcpy(ord.GTDDate, "");
        // Volume type
        ord.VolumeCondition = SHFE_FTDC_VC_AV;
        // The Min.volume
        ord.MinVolume = 0;
        // Trigger conditions
        ord.ContingentCondition = SHFE_FTDC_CC_Immediately;
        // Stop-loss price
        ord.StopPrice = 0;
        // Reasons for forced closing-out
        ord.ForceCloseReason = SHFE_FTDC_FCC_NotForceClose;
        // Local order number
        strcpy(ord.OrderLocalID, "0000000001");
        // Flag of auto-suspension
        ord.IsAutoSuspend = 0;

        int ret = m_pTraderApi ->ReqOrderInsert(&ord, 1);
        if (ret != 0)
```

```
        {
            printf("ReqOrderInsert Fail ret = %d\n", ret);
            exit(-1);
        }

    }

    // Response to order entry
    virtual void OnRspOrderInsert(CShfeFtdcInputOrderField *pInputOrder,
CShfeFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
    {
        // Output of order entry result
        printf("ErrorID=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);

        // Order Entry Completed
        exit(0);
    };

    ///Return on order
    virtual void OnRtnOrder(CShfeFtdcOrderField *pOrder)
    {
        printf("OnRtnOrder:\n");
        printf("OrderSysID=[%s]\n", pOrder->OrderSysID);
    }

    // Notification on erroneous user request
    virtual void OnRspError(CShfeFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
{
        printf("OnRspError:\n");
        printf("ErrorID=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
        // Member System is required to conduct error-processing
        exit(-1);
    }

private:
    // Pointer pointing to the instance of CShfeFtdcMduserApi
    CShfeFtdcTraderApi *m_pTraderApi;
};

int main()
{
    // Generate an instance of CShfeFtdcTraderApi
    CShfeFtdcTraderApi *pTraderApi =
CShfeFtdcTraderApi::CreateFtdcTraderApi();
    // Generate an incident-handling instance
    CSimpleHandler sh(pTraderApi);
    // Register an incident-handling instance
    pTraderApi->RegisterSpi(&sh);

    // Subscription of private stream
    pTraderApi->SubscribePrivateTopic(TERT_RESUME);

    // Subscription of public stream
```

```
    pTraderApi->SubscribePublicTopic(TERT_RESUME);

    //Set the heartbeat timout period
    pTraderApi->SetHeartbeatTimeout(19);

    // Set the address of NameServer of Trading System front-end
    pTraderApi->RegisterNameServer("tcp://172.16.0.31:17001");

    // Enable Member System to create connection with Trading System
    pTraderApi->Init();

    // Releas of API instance
    pTraderApi->Join();

    return 0;
}
```

Part III MduserAPI Reference Manual

This section is primarily intended for market data receiving system developers, and includes:

Chapter 1 ***MduserAPI*** Interface Categories.

Chapter 2 ***MduserAPI*** Interface Description.

Chapter 3 ***MduserAPI*** Interface Development Instances.

1. Categories of MduserAPI Interfaces

1.1. Management Interfaces

The ***MduserAPI*** management interface is used to control the API lifecycle and runtime parameters.

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 end the run
	CShfeFtdcMduserApi::GetTradingDay	Register to callback interface
Parameter Management Interfaces	CShfeFtdcMduserApi::RegisterSpi	Register Front Address
	CShfeFtdcMduserApi::RegisterFront	Register to NameServer Network address
	CShfeFtdcMduserApi::RegisterNameServer	Register to NameServer Network address
	CShfeFtdcMduserApi::SetHeartbeatTimeout	Set the heartbeat timeout
Subscription Interfaces	CShfeFtdcMduserApi::SubscribeMarketDataTopic	Subscribe to market data
Logging Interface	CShfeFtdcMduserApi::OpenRequestLog	This is to open the request log file
	CShfeFtdcMduserApi::OpenResponseLog	This is to open the reply log file
Communication Status Interfaces	CShfeFtdcMduserSpi::OnFrontConnected	When communication with the Trading System is established, this method will be called
	CShfeFtdcMduserSpi::OnFrontDisconnected	This method will be called when communication with the Trading System is disconnected
	CShfeFtdcMduserSpi::OnHeartBeatWarning	The method is called when no heartbeat message is received after a long time
	CShfeFtdcMduserSpi::OnPackageStart	Notification at the start of message callbacks
	CShfeFtdcMduserSpi::OnPackageEnd	Notification at the end of message callbacks
Disaster Recovery Interface	CShfeFtdcMduserSpi::OnRtnFlowMessageCancel	Notification for data stream cancellation

1.2. Service Interfaces

Service Type	Service	Request Interface / Response Interface	Data Stream
Login-Logout	Login	CShfeFtdcMduserApi::ReqUserLogin CShfeFtdcMduserSpi::OnRspUserLogin	Dialog Stream
	Logout	CShfeFtdcMduserApi::ReqUserLogout CShfeFtdcMduserSpi::OnRspUserLogout	Dialog Stream
	Change user password	CShfeFtdcMduserApi::ReqUserPasswordUpdate CShfeFtdcMduserSpi::OnRspUserPasswordUpdate	Dialog Stream
Subscription	Subscribe Topics	CShfeFtdcMduserApi::ReqSubscribeTopic	Dialog Stream

Service Type	Service	Request Interface / Response Interface	Data Stream
		CShfeFtdcMduserSpi::OnRspSubscribeTopic	
	Query Topics	CShfeFtdcMduserApi::ReqQryTopic CShfeFtdcMduserSpi::OnRspQryTopic	Query Streams
Market Data	Market Data Notification	CShfeFtdcMduserSpi::OnRtnDepthMarketData	Market Data Stream
Error Response	Error Response	CShfeFtdcMduserSpi::OnRspError	Dialogue Streams

2. MduserAPI Interface Description

2.1. CShfeFtdcMduserSpi Interface

The ***CShfeFtdcMduserSpi*** implements the event notification interface. Users must derive the ***CShfeFtdcMduserSpi*** interface and write event-handling methods to process the required events.

2.1.1. OnFrontConnected Method

When the market data receiving system establishes a TCP virtual link (connection) with the Trading System, this method will be called. The connection is automatically established by the API.

Function Prototype:

```
void OnFrontConnected();
```

Note: Calling ***OnFrontConnected*** only indicates that the TCP connection is successful. The market data receiving system must perform login operations to conduct subsequent business activities.

2.1.2. OnFrontDisconnected Method

When the communication link between the market data receiving system and the Trading System is disconnected, this method will be called. Upon disconnection, the API will automatically reconnect. The reconnection address may be the originally registered address or another available communication address supported by the system, which is selected automatically by the program.

Function Prototype:

```
void OnFrontDisconnected(int nReason);
```

Parameter:

nReason: disconnection reasons

0x1001, indicates network read failure;
0x1002, indicates network write failure;
0x2001, indicates heartbeat timeout;
0x2002, indicates message encryption failure;
0x2003, indicates message decryption failure;
0x2004, indicates receipt of a message from an unsubscribed topic;
0x2005, indicates discontinuity in received message sequence numbers;
0x2006, indicates illegal message length;
0x2007, indicates message conversion error;
0x2008, indicates login error with front-end service.

2.1.3. OnHeartBeatWarning Method

Heartbeat timeout warning. This method is invoked when no message is received for a prolonged period. The default timeout warning threshold is 5 seconds. If SetHeartbeatTimeout (unsigned int timeout) has been called to set a custom heartbeat timeout, the warning time is timeout/2.

Function Prototype:

```
void OnHeartBeatWarning(int nTimeLapse);
```

Parameter:

nTimeLapse: time lapse from last time receiving the message (in seconds)

2.1.4. OnPackageStart Method

Message callback start notification. When the API receives a message belonging to the market data stream, this method will be called first, followed by individual data field callbacks, and finally the message callback end notification.

Function Prototype:

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

Parameter:

nTopicID: Topic ID (e.g., market data topic 1001).

nSequenceNo: Message Sequence Number

2.1.5. OnPackageEnd Method

Message callback end notification. When the API receives a message belonging to the market data stream, it first calls the message callback start notification, followed by individual data field callbacks, and finally this method.

Function Prototype:

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

Parameter:

nTopicID: Topic (e.g., market data topic 1001).

nSequenceNo: Message Sequence Number

2.1.6. OnRspUserLogin Method

After the Market Data Receiving System sends out login request, and when the Trading System sends back the response, the Trading System will call this method to inform the Market Data Receiving System whether the login is successful.

Function Prototype:

```
void OnRspUserLogin(
        CShfeFtdcRspUserLoginField* pRspUserLogin,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
```

```
        bool bIsLast);
```

Parameter:

pRspUserLogin: pointer to the user login information structure. The structure:

```
struct CShfeFtdcRspUserLoginField {
    ///trading day
    TShfeFtdcDateType  TradingDay;
    ///successful login time
    TShfeFtdcTimeType  LoginTime;
    ///Maximum local order number, not used
    TShfeFtdcOrderLocalIDType  MaxOrderLocalID;
    ///Trading User ID
    TShfeFtdcUserIDType  UserID;
    ///Exchange Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Trading System Name
    TShfeFtdcTradingSystemNameType  TradingSystemName;
    ///Data Center ID
    TShfeFtdcDataCenterIDType  DataCenterID;
    ///Current length of member private stream, not used
    TShfeFtdcSequenceNoType  PrivateFlowSize;
    ///Current length of trader private stream, not used
    TShfeFtdcSequenceNoType  UserFlowSize;
    ///action day
    TShfeFtdcDateType  ActionDay;};
```

pRspInfo: returns the user response information structure. Error ID of 0 indicates success, and the same for subsequent descriptions. The response information structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
3	Member cannot be found	Member ID is wrong when logging in
45	Settlement group initialization status is incorrect	Trading System initialization is not completed, may try later
59	User multiple login	The trading user has logged in already
60	Wrong user ID or password	User ID or password is wrong
62	User account locked	Trading System locked the trader's account
64	User is not belong to the Member	Member ID is wrong
65	Wrong login IP address	The computer used to login does not have the IP address allowed by SHFE
75	Front-end inactive	Trading System front-end inactive
106	Duplicate session	Multiple logins using the same session
135	User authentication failed	User key verification failed
136	User has no permission for direct front-end connection	User has no permission for direct front-end connection

nRequestID: returns the user login request ID; this ID is specified by the user upon login

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

2.1.7. OnRspUserLogout Method

This method will be called when the Trading System returns a response after the Market Data Receiving System sends a logout request, indicating whether logout was successful.

Function Prototype:

```
void OnRspUserLogout(
        CShfeFtdcRspUserLogoutField* pRspUserLogout,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pRspUserLogout: returns the user logout information structure. The structure:

```
struct CShfeFtdcRspUserLogoutField {
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Memebr ID
    TShfeFtdcParticipantIDType  ParticipantID;
};
```

pRspInfo: returns the user response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
45	Settlement group initialization status incorrect	Initialization of Trading System is not completed, please try later
66	User not logged in yet	User has not logged in yet
67	Not logged in with this user ID	User logging out is not the same as the one logged in
68	Not logged in with this Memebr ID	Member logging out is not the same as the one logged in

nRequestID: returns the user logout request ID; this ID is specified by the user upon logout

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

2.1.8. OnRspSubscribeTopic Method

Subscription topic response. This method will be called when the Trading System returns a response after the Market Data Receiving System sends a subscription topic instruction.

Function Prototype:

```
void OnRspSubscribeTopic(
        CShfeFtdcDisseminationField* pDissemination,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pDissemination: pointer to the subscription topic structure, including topic subscribed and starting message sequence number. The structure:

```
struct CShfeFtdcDisseminationField {
    ///sequence series
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
};
```

pRspInfo: pointer to the response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
1	Invalid session or topic does not exist	The topic does not exist or the user lacks the necessary subscription permission

nRequestID: returns the user subscription topic request ID; this ID is specified by the user upon subscription.

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

2.1.9. OnRspQryTopic Method

Query topic response. This method will be called when the Trading System returns a response after the Market Data Receiving System issues a query topic instruction.

Function Prototype:

```
void OnRspQryTopic(
        CShfeFtdcDisseminationField* pDissemination,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pDissemination: pointer to the query topic structure, including the topic to be queried and the number of messages related to that topic. The structure:

```
struct CShfeFtdcDisseminationField {
    ///sequence series
    TShfeFtdcSequenceSeriesType  SequenceSeries;
```

```
    ///sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
};
```

pRspInfo: pointer to the response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error Message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

nRequestID: returns the user query topic request ID; this ID is specified by the user upon subscription to the topic.

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

2.1.10. OnRspError Method

Error notification for user requests.

Function Prototype:

```
void OnRspError(
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pRspInfo: pointer to the response information structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

Possible errors:

Error ID	Error message	Possible cause
1	Not Login	Not yet logged in
	Too High FTD Version	FTD version too high
	Unrecognized ftd tid	FTD message header error
151	Version verification failed	Market data API version verification failed
997	api authentication failure	Illegal API access
	api crypt info failure	Query for encrypted information not completed

nRequestID: returns the user operation request ID; this ID is specified by the user upon making an operation request.

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

2.1.11. OnRtnDepthMarketData Method

Market data notification. This method will be called when the Trading System notifies the Market Data Receiving System of market data changes.

Function Prototype:

```
void OnRtnDepthMarketData(
        CShfeFtdcDepthMarketDataField* pDepthMarketData);
```

Parameter:

pDepthMarketData: pointer to the market data structure. Note: For level-1 quotes, some fields (such as bid2-bid5, ask2-ask5) are meaningless. The market data structure:

```
struct CShfeFtdcDepthMarketDataField
{
    ///Business day
    TShfeFtdcDateType  TradingDay;
    ///Settlement group's ID
    TShfeFtdcSettlementGroupIDType  SettlementGroupID;
    ///Settlement number
    TShfeFtdcSettlementIDType  SettlementID;
    ///Latest price
    TShfeFtdcPriceType  LastPrice;
    ///Yesterday's settlement
    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
    TShfeFtdcRatioType  PreDelta;
    ///Today's Delta value
    TShfeFtdcRatioType  CurrDelta;
    ///Last modification time
    TShfeFtdcTimeType  UpdateTime;
    ///The last modified millisecond
    TShfeFtdcMillisecType  UpdateMillisec;
```

```
        ///Contract ID
        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;
        ///Action day
        TShfeFtdcDateType  ActionDay;
    };
```

2.1.12. OnRtnFlowMessageCancel Method

Data stream rollback notification. After the Trading System undergoes a disaster recovery switch and when the user logs back into the Trading System and subscribes to a specific data stream, the Trading System will proactively notify the Market Data Receiving System that certain messages in the data stream have been invalidated or canceled. At this

time, this method will be called.

Function Prototype:

```
void OnRtnFlowMessageCancel(
        CShfeFtdcFlowMessageCancelField* pFlowMessageCancel);
```

Parameter:

pFlowMessageCancel: pointer to the data stream rollback structure. The structure:

```
struct CShfeFtdcFlowMessageCancelField
{
    ///Sequence Series
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Trading Day
    TShfeFtdcDateType  TradingDay;
    ///Data Center ID
    TShfeFtdcDataCenterIDType  DataCenterID;
    ///Start Sequence number
    TShfeFtdcSequenceNoType  StartSequenceNo;
    ///End Sequence number
    TShfeFtdcSequenceNoType  EndSequenceNo;
};
SequenceSeries: the data stream series to be canceled (private stream or public
stream)
The messages to be canceled is between: (StartSequenceNo,EndSequenceNo)
```

2.1.13. OnRspUserPasswordUpdate Method

User password update response. This method will be called when the Trading System returns a response after the Market Data Receiving System issues a user password update command.

Function Prototype:

```
void OnRspUserPasswordUpdate(
        CShfeFtdcUserPasswordUpdateField* pUserPasswordUpdate,
        CShfeFtdcRspInfoField* pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pUserPasswordUpdate: pointer to the user password update structure, including the input data of the user password update request. The user password update structure:

```
struct CShfeFtdcUserPasswordUpdateField {
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Old Password
    TShfeFtdcPasswordType  OldPassword;
    ///New Password
    TShfeFtdcPasswordType  NewPassword;
};
```

```

```

pRspInfo: pointer to the response message structure. The structure:

```
struct CShfeFtdcRspInfoField {
    ///Error ID
    TShfeFtdcErrorIDType  ErrorID;
    ///Error Message
    TShfeFtdcErrorMsgType  ErrorMsg;
};
```

nRequestID: returns the user password update request ID; this ID is specified by the user during the update.

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

2.2. CShfeFtdcMduserApi Interfaces

Functions offered by ***CShfeFtdcMduserApi*** interfaces include login/logout, market data subscription, etc.

2.2.1. CreateFtdcMduserApi Method

This is to create an instance of the ***CShfeFtdcMduserApi***; this cannot be created with a "new".

Function Prototype:

```
static CShfeFtdcMduserApi* CreateFtdcMduserApi(const char* pszFlowPath = "");
```

Parameter:

pszFlowPath: constant character pointer, used to point to a file catalog/directory that stores the status of the bulletin/news sent by the Trading System. The default value is the current directory.

Returned Value:

returns a pointer to an instance of the ***CShfeFtdcMduserApi***.

2.2.2. GetVersion Method

This is to get the API version.

Function Prototype:

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

Parameter:

nMajorVersion: returns the major version number

nMinorVersion: returns the minor version number

Returned Value:

returns a constant pointer to the version identifier string.

2.2.3. Release Method

Release the internal resources of the current API instance, exit the API working thread, and set the API exit signal (only sets the exit signal, does not release the instance).

Function Prototype:

```
int Release();
```

Returned Value:

0, successful

-9 indicates uninitialized.

2.2.4. Init Method

This is to establish the connection between Market Data Receiving System and the Trading System. After the connection is established, users can proceed to login.

Function Prototype:

```
int Init();
```

Returned Value:

0, successful

-5 indicates already logged in or repeated invocation.

2.2.5. Join Method

Blocks the API working thread. After the API exit signal is triggered, the current API instance will be released.

Function Prototype:

```
int Join();
```

Returned Value:

0, successful

2.2.6. GetTradingDay Method

This is to get the current trading day. Only after successfully login to the Trading System, the correct value would be obtained.

Function Prototype:

```
const char* GetTradingDay();
```

Returned Value:

Returns a constant pointer to the date information character string.

2.2.7. RegisterSpi Method

This is to register an instance derived from ***CShfeFtdcMduserSpi*** instance class. This instance would be used to complete events handling.

Function Prototype:

```
void RegisterSpi(CShfeFtdcMduserSpi* pSpi);
```

Parameter:

pSpi: pointer to an instance that implements the CShfeFtdcMduserSpi interface.

2.2.8. RegisterFront Method

Set the network communication address of market data front-ends. The Trading System supports multiple market data front-ends, and users can register the network communication addresses of multiple front-ends simultaneously.

Function Prototype:

```
int RegisterFront(const char* pszFrontAddress);
```

Parameter:

pszFrontAddress: pointer to the network communication address of market data front-ends. The server address is in the format "protocol://ipaddress:port", e.g. "tcp://127.0.0.1:17001". "tcp" in the instance is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represents the server port number.

Returned Value:

0, successful

-8, indicates the number of registered front addresses exceeds the maximum value;

-10, indicates already initialized.

2.2.9. RegisterNameServer Method

Set the network communication address of the Trading System's FENS service. The Trading System has multiple FENS services, and users can register multiple FENS service network communication addresses simultaneously.

Function Prototype:

```
int RegisterNameServer(const char* pszNsAddress);
```

Parameter:

pszNsAddress: pointer to the Trading System FENS service network communication address. The network communication address is in the format "protocol://ipaddress:port", e.g. "tcp://127.0.0.1:17001". "tcp" in the instance is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represents the server port number.

Returned Value:

0, successful

-8, indicates the number of registered FENS service addresses exceeds the maximum value;

-10, indicates already initialized.

2.2.10. SetHeartbeatTimeout Method

Set the heartbeat timeout limit for network communication. When the ***MduserAPI*** establishes a TCP connection with the Trading System, the connection will periodically send heartbeats to check the connection status. This method is used to set the time for the detecting heartbeat timeout. **The Exchange recommends that the Market Data Receiving System set the timeout value to between 10 and 30 seconds.**

Function Prototype:

```
int SetHeartbeatTimeout(unsigned int timeout);
```

Parameter:

timeout: heartbeat timeout time limit (in seconds). If no information is received from the Trading System for more than timeout/2 seconds, the ***OnHeartBeatWarning*** callback will be triggered. If no information is received from the Trading System for more than timeout seconds, the connection will be disconnected, triggering the ***OnFrontDisconnected*** callback.

Returned Value:

0, successful

-10, indicates already initialized.

2.2.11. OpenRequestLog Method

Open the request log file. After this method is called, all request messages sent to the Trading System will be recorded in the specified log file.

Function Prototype:

```
int OpenRequestLog(const char* pszReqLogFileName);
```

Parameter:

pszReqLogFileName: the request log file name.

Returned Value:

0, successful

-4, indicates log file opening failed

2.2.12. OpenResponseLog Method

Open the reply log file. After this method is called, all information returned from the Trading System will be recorded in the specified log file, including reply message and return message.

Function Prototype:

```
int OpenResponseLog(const char* pszRspLogFileName);
```

Parameter:

pszRspLogFileName: reply log file name.

Returned Value:

0, successful

-4, indicates log file opening failed;

2.2.13. SubscribeMarketDataTopic Method

Subscribe to market data. After subscription, the Trading System will proactively send market data notifications to the Market Data Receiving System.

Function Prototype:

```
int SubscribeMarketDataTopic(int nTopicID, TE_RESUME_TYPE nResumeType);
```

Parameter:

nTopicID: The topic ID of the market data to be subscribed, as published by the Exchange.

NResumeType: Market data re-transmission method type:

TERT_RESTART: to re-transmit from current trading day

TERT_RESUME: to re-transmit by resuming and continuing from last transmission

TERT_QUICK: first transmit the market data snapshot, and then transmit all market data after that. **The Exchange recommends that members use this method to recover market data quickly.**

Returned Value:

0, successful

-8, indicates the number of subscribed market data topics exceeds the maximum limit;

-10, indicates already initialized.

2.2.14. ReqUserLogin Method

User login request.

Function Prototype:

```
int ReqUserLogin(
        CShfeFtdcReqUserLoginField* pReqUserLoginField,
        int nRequestID);
```

Parameter:

pReqUserLoginField: pointer to the user login request structure. The structure:

```
struct CShfeFtdcReqUserLoginField {
    ///trading day
    TShfeFtdcDateType  TradingDay;
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Password
    TShfeFtdcPasswordType  Password;
    ///The user-end product information
    TShfeFtdcProductInfoType  UserProductInfo;
    ///The interface-port product information
```

```
        TShfeFtdcProductInfoType  InterfaceProductInfo;
        ///Protocol information
        TShfeFtdcProtocolInfoType  ProtocolInfo;
        ///Datacenter ID
        TShfeFtdcDataCenterIDType  DataCenterID;
    };
```

Users must fill in the UserProductInfo field, which specifies the market data receiving system's product information (e.g., software developer, version number). For instance: "SFIT Mduser V100" represents a market data receiving program and version developed by a technology company.

nRequestID: returns the user login request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-2, Exceeded in-transit market data flow control;

-3, Exceeded market data request flow control;

-5, Already logged in;

-6, Mandatory field is empty (UserProductInfo not provided);

-9, indicates uninitialized;

-12, Connection to front-end not yet established.

2.2.15. ReqUserLogout Method

User logout request.

Function Prototype:

```
int ReqUserLogout(
        CShfeFtdcReqUserLogoutField* pReqUserLogout,
        int nRequestID);
```

Parameter:

pReqUserLogout: pointer to the user logout request structure. The structure:

```
struct CShfeFtdcReqUserLogoutField {
    ///Trading User ID
    TShfeFtdcUserIDType  UserID;
    ///Member ID
    TShfeFtdcParticipantIDType  ParticipantID;
};
```

nRequestID: returns the user logout request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit market data flow control;

-3, indicates exceeding market data request flow control.

2.2.16. ReqSubscribeTopic Method

Subscribed topic request.

Function Prototype:

```
int ReqSubscribeTopic(
        CShfeFtdcDisseminationField* pDissemination,
        int nRequestID);
```

Parameter:

pDissemination: pointer to the subscribed topic structure, including topic to be subscribed as well as the starting message sequence number. The structure:

```
struct CShfeFtdcDisseminationField {
    ///sequence series
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///sequence number
    TShfeFtdcSequenceNoType  SequenceNo;
};
SequenceSeries: topics to be subscribed
SequenceNo: <0 to re-transmit using the “QUICK” method
```

nRequestID: returns the user subscribed topic request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit market data flow control;

-3, indicates exceeding market data request flow control;

-8, indicates the number of subscribed market data topics exceeding the limit.

2.2.17. ReqQryTopic Method

This is the request for querying topic.

Function Prototype:

```
int ReqQryTopic(
        CShfeFtdcDisseminationField* pDissemination,
        int nRequestID);
```

Parameter:

pDissemination: pointer to the query topic structure, including topic to be queried. The structure:

```
struct CShfeFtdcDisseminationField {
    ///Serial series number: Fill in the topic number to query
    TShfeFtdcSequenceSeriesType  SequenceSeries;
    ///Sequence number, unused field
    TShfeFtdcSequenceNoType SequenceNo;
};
```

nRequestID: returns the user query topic request ID; this ID is specified and managed

by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit market data flow control;

-3, indicates exceeding market data request flow control.

2.2.18. ReqUserPasswordUpdate Method

User password update request.

Function Prototype:

```
int ReqUserPasswordUpdate(
        CShfeFtdcUserPasswordUpdateField* pUserPasswordUpdate,
        int nRequestID);
```

Parameter:

pUserPasswordUpdate: pointer to the user password update structure. The structure:

```
struct CShfeFtdcUserPasswordUpdateField {
    ///transaction user's ID
    TShfeFtdcUserIDType  UserID;
    ///member ID
    TShfeFtdcParticipantIDType  ParticipantID;
    ///Old Password
    TShfeFtdcPasswordType  OldPassword;
    ///New Password
    TShfeFtdcPasswordType  NewPassword;
};
```

nRequestID: returns the user password update request ID; this ID is specified and managed by the user.

Returned Value:

0, successful

-1, indicates not logged in;

-2, indicates exceeding in-transit market data flow control;

-3, indicates exceeding market data request flow control;

-13, indicates a member ID mismatch;

-14, indicates a user ID mismatch.

Business Description:

This feature is not supported in the current version.

3. MduserAPI Interface Development Instance

```
// mdusertest.cpp :
// A simple instance that describes the use of interface for CShfeFtdcTraderApi and
CShfeFtdcTraderSpi.
// When a market data field value equals DBL_MAX (1.7976931348623157e+308), it
actually indicates a null field
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <float.h>
#include "FtdcMduserApi.h"

class CSimpleHandler : public CShfeFtdcMduserSpi
{
public:
    // Constructed function that needs an effective pointer pointing to the
CShfeFtdcMduserApi instance
    CSimpleHandler(CShfeFtdcMduserApi *pMduserApi) : m_pMduserApi(pMduserApi) {}

    ~CSimpleHandler() {}

    // After the market data receiving system establishes a communication connection with
the Trading System, it must log in
    void OnFrontConnected()
    {
        CShfeFtdcReqUserLoginField reqUserLogin;
        memset(&reqUserLogin, 0, sizeof(reqUserLogin));
        strcpy(reqUserLogin.ParticipantID, "0888");
        strcpy(reqUserLogin.UserID, "0888c1c");
        strcpy(reqUserLogin.Password, "1");
        strcpy(reqUserLogin.UserProductInfo, "TestMduserAPI V2.00");
        //Send login request
        int ret = m_pMduserApi ->ReqUserLogin(&reqUserLogin, 0);
        if (ret != 0)
        {
            printf("ReqUserLogin Fail ret = %d\n", ret);
        }
    }

    // When the communication connection between Market Data Receiving System and
the Trading System is interrupted, this method will be called
    void OnFrontDisconnected(int nReason) {
        // When disconnection happens, API would re-connect automatically, and the
Market Data Receiving System does not need to handle
        printf("OnFrontDisconnected Reason = %#x.\n", nReason);
    }

    // This method will be called after the Trading System returns a login response to
indicate whether the login was successful
    void OnRspUserLogin(CShfeFtdcRspUserLoginField *pRspUserLogin,
CShfeFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
        printf("OnRspUserLogin: ErrorID=[%d], ErrorMsg=[%s]\n",
    pRspInfo->ErrorID, pRspInfo->ErrorMsg);
```

```
        printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
        if (pRspInfo->ErrorID != 0) {
            // Login failed. Error handling is required
            printf("Failed to login, errorID=%d errormsg=%s requestid=%d chain=%d",
pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
        }
    }

    // Depth market data notification, and the Trading System would inform automatically
    void OnRtnDepthMarketData(CShfeFtdcDepthMarketDataField *pMarketData) {
        // Quotion Receiving System would deal with the returned data based on its own
need
        if(pMarketData->OpenPrice!=DBL_MAX)
        {
            printf("OpenPrice=%.2f\n",pMarketData->OpenPrice);
        }
    }

    // Error notification with respect to user request
    void OnRspError(CShfeFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
    printf("OnRspError:\n");
    printf("ErrorID=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
    printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
    // Market Data Receiving System would need to do error handling
    }

private:
    // Pointer to an instance of CShfeFtdcMduserApi
    CShfeFtdcMduserApi *m_pMduserApi;
};

int main()
{
    // Create an instance of CShfeFtdcTraderApi
    CShfeFtdcMduserApi *pMduserApi = CShfeFtdcMduserApi::CreateFtdcMduserApi();
    // Create an instance of event handling
    CSimpleHandler sh(pMduserApi);
    // Register to an instance of event handling
    pMduserApi->RegisterSpi(&sh);
    // Register to required depth market data topic
    pMduserApi-> SubscribeMarketDataTopic (1001, TERT_RESUME);
    // Set the timeout for heartbeat
    pMduserApi->SetHeartbeatTimeout(19);
    // Set the Exchange FEP NameServer address
    pMduserApi->RegisterNameServer("tcp://192.168.1.1:17011");

    // Starts connection with market data FEP of the Trading System
    pMduserApi->Init();
    // Wait for API instance to exit
    pMduserApi->Join();

    return 0;
}
```

Part IV Appendix

1. Error ID List

Error number	Error message	Reasons for error
-1	Authentication failed	Unable to find the authorization ID corresponding to the trading terminal or the authorization ID does not match
1	Invalid session or topic does not exist	Subscribed topic does not exist, the number of subscribed topics exceeds the upper limit, or the user lacks the corresponding subscription permission
	Not Login	User hasn't logged in yet
	Too High FTD Version	FTD version too high
	Unrecognized ftd tid	FTD message header error
2	Contract cannot be found	Contract not found during operations
3	Member cannot be found	Member cannot be found in each operation
4	Client cannot be found	Client cannot be found in each operation
6	Incorrect Order field	Illegal field value was found on the order when inserting the order (out-of-range of the enumerated value)
		Forced closing-out reasons was set in non-forced closing-out order when inserting the order
7	Erroneous quote field	Illegal field value was found in the quote when inserting the quote (out-of-range of the enumerated value)
8	Incorrect field in order operation	Illegal field value was found in the order operation at the time of order operation (out-of-range of the enumerated value)
		Fields in the order derived from a quote operation are invalid (e.g., price not a floating-point number or outside the valid range)
9	Incorrect field in quote operation	Invalid field values detected during a quote operation (enumeration value out of range, or operation flag set to modify or suspend)
12	Duplicate order	Duplicate local order ID detected when inserting an order
13	Duplicate quote	Local quoto number was duplicate when inserting quote
15	Client didn't open an account at this member	It was fount during each operation that the designated client didn't open an account at the designated member
16	IOC to be conducted in continuous trade session.	Attempting to insert an IOC order outside continuous trading phase
17	GFA to be conducted in call auction session	Attempting to insert a GFA order outside the call auction phase
19	Quantity restriction shall be put on IOC	It was found in inserting the order with a quantity restriction of non-arbitrary quantity that time conditions are not IOC
20	GTD order had expired	It was found in inserting the GTD order that GTD data had expired
21	The Min. number exceeds the number of order	It was found in inserting the order with a Min. number requirement that the Min. number exceeds the number of order

22	The Exchange's data is not in the synchronized state	It was found during operation of each business that the Exchange's data is not in the synchronized state
23	The settlement group's data is not in the synchronized date	It was found during operation of each business that the settlement group's data is not in the synchronized state
24	Order cannot be found	It was found during order operation that order to be operated cannot be found
25	Quote cannot be found	It was found during quote operation that quote to be operated cannot be found
		While inserting orders/quotes, contract trading status is not continuous trading, call auction order entry, or call auction balancing
26	This operation is prohibited by current state	During order/quote operations, for activation, contract trading status is not continuous trading, call auction order entry, or call auction balancing; For other operations, contract trading status is not continuous trading or call auction order entry
		During option exercise/abandonment insertion or operations, or option self-hedge update/operations, contract trading status is not continuous trading or trade processing state
		During quote demand insertion, contract is not tradable or not in continuous trading status
28	Order already fully filled	During order/quote operations, the order has already been fully filled
29	Order already canceled	During order/quote operations, the order has already been canceled
30	Not enough quantity to modify	Remaining quantity after modifying order would be less than 0
31	The client's open interest is insufficient at the time of closing-out	It was found during each operation that may cause closing out that client's open interest is insufficient
32	Exceeding client's position limit	It was found during each operation that is likely to open a position that it has exceeded client's speculative position
33	Insufficient member position for closing	During operations potentially resulting in position closing, member's position is insufficient
34	Exceeding member's position limit	It was found during each operation that is likely to open a position that it has exceeded member's position limit
35	Account cannot be found	It was found during each operation that the account shall be used for such operation cannot be found
36	Inadequate fund	It was found during each operation that there is no sufficient fund in the account
37	Invalid quantity	When inserting orders, performing order operations, inserting quotes, entering option exercise, submitting option abandonment, or updating option self-hedge, the quantity is not a positive integer multiple of the minimum order quantity or exceeds the maximum
45	The settlement group's initialization state is not correct	Trading System is not fully initialized
48	Price not a multiple of minimum price fluctuation	It was found during each operation that price is not the integral mutiple of the contract's tick size
49	Price exceeds the upward limit	It was found during each operation that the price is higher than the contract's upward price limit
50	Price exceeds the downward limit	It was found during each operation that the price is lower than the contract's downward price limit
51	Not authorized to trade	During operations, member, client, or user lacks trading permissions for the specified contract

52	Only can close out position	During operations that may open a position, member, client, or user only has permission for closing positions on the specified contract
53	No such trading role	During order insertion, quote insertion, option self-hedge updates/operations, member does not hold the corresponding client's trading role for the specified contract
54	Session Not Found	Session not found during operations
57	Operation shall not be conducted by other members	It was found during each operation that user conduct operation on behalf member to whom he is not subordinate
58	Unmatched user	It was found during each operation that user for operation doesn't match with user for dialogue
59	duplicate login by user	Duplicate login detected from different IP addresses by the same user
60	Incorrect username or password	It was found during user's login or password modification that username cannot be found or password is incorrect
62	User is not active	During user login or password modification, user lacks permission for login, trading, or password modification
64	User doesn't belong to this member	It was found during user's login that user doesn't belong to the designated member
65	Incorrect IP address of login	It was found during user's login that user' IP address is illegal
66	User hasn't logged in yet	During logout or password modification, user has not yet logged in
67	User not logged in under the specified account	User logging out differs from user who logged in
68	User hasn't logged in yet	User not logged in under the specified member during logout or password modification
70	Quote has been canceled	It was found during quote operation that quote has been canceled
71	Cannot operate on derived orders	During order operations, user attempts to operate on derived orders
72	Opening positions not allowed for natural persons	In the delivery month, natural person type clients initiating open positions or performing activation or modification operations on open position orders
75	Front-end inactive	Trading System front-end inactive
76	Order has been suspended	It was found during suspension of order that order has already been suspended
77	Order has been activated	It was found during activation of order that order has already been activate
78	Date is not set on GTD order	It was found in inserting GTD order that GTD date hasn't been designated
79	Unsupported order type	It was found in inserting various orders that this trade at this moment doesn't support this order type
80	User is not authorized to do so	User lacks permission for the requested operation
83	Stop-loss order is used for continuous trade only	Attempting to insert or operate on stop orders outside continuous trading phase
84	Stop-loss order is required to be IOC or GFD	It was found in inserting stop-loss order that time condition is neither IOC nor GFD
88	Target user to be operated on not found	User for quote demand not found during quote operation
89	Incorrect option exercise field	Invalid field detected in option exercise insertion/operation (enumeration value out of range)
90	Incorrect field in option exercise operation	Illegal field value was found in option exercise operation when operating declaration (out-of-range of the enumerated value)
91	Duplicate option exercise	At the time of inserting option exercise, local option exercise number is duplicate
92	Option exercise has been canceled	It was found during option exercise operation that declaration has already been canceled

93	Option exercise cannot be found	It was found during option exercise operation that to-be-operated declaration cannot be found
94	Option exercise can only be used for option	It was found in inserting the option exercise that the contract is non-option contract
95	The stop-loss price shall be specified on stop-loss order	No stop price specified during stop order insertion/operation
96	Insufficient hedge quota	During hedge orders or quote insertion, client's hedge quota is insufficient
98	Forced liquidation orders must be used by administrators	Non-administrator user submitted a forced liquidation order
99	Operation cannot be conducted by other users	Unauthorized user attempting to operate on orders/quotes inserted by another user of the same member
100	Incorrect user type	User identified as a market data user during login
101	Clearing members are not allowed to trade	Attempt to perform trading-related operations using a settlement member account
102	Corresponding clearing member not found	Settlement member not found for the specified member during operations
103	Hedge position on that day cannot be closed out	Attempt to insert the order for closing out today's position into hedge position
106	Duplicate session	Two login attempts issued in the same session
114	The best price orders are unable to queue	It was found in inserting the best price order that time condition is not IOC
121	Erroneous option abandonment field	Invalid fields found during option abandonment insertion/operation
122	Erroneous option abandonment operation field	Illegal field value was found in the option abandonment at the time of option abandonment operation
123	Duplicate option abandonment	Duplicate local option abandonment ID during insertion
124	Option abandonment canceled	Option abandonment has been canceled at the time of option abandonment operation
125	Option abandonment cannot be found	Option abandonment cannot be found at the time of option abandonment operation
126	Option abandonment can only be used in futures option	The contract is non-option contract when inserting the option abandonment
127	Not in declaration period	Option exercise is not in definitive period when insert or option abandonment
128	Only holders of long positions can enjoy execution waiver	Option sellers are not allowed to enjoy execution waiver
129	Option exercise or abandonment cannot be open position	Flag of open or closing position is open position when inserting option exercise or abandonment
131	Exceeded client's intraday contract opening limit	Client's cumulative intraday open volume on a contract exceeds the limit
132	Exceeded client's per-second order limit for the product	Number of client orders on a product within one second exceeds the limit
133	Exceeded client's per-second cancel limit for the product	Number of client cancellations on a product within one second exceeds the limit
134	API validation failed	Non-official API library used
135	User authentication failed	Developer software not certified by the exchange
136	User has no permission for direct front-end connection	User required to obtain front-end addresses through FENS server used direct connection mode
137	Option self-hedge field error	Option self-hedge update contains invalid field values (enumeration value out of range)

138	Option self-hedge operation field error	Invalid field detected in option self-hedge operation (enumeration value out of range)
139	Duplicate option self-hedge update	Duplicate local option self-hedge ID in the option self-hedge update
140	Option self-hedge update has been canceled	Targeted option self-hedge update already canceled
141	Option self-hedge update is only applicable to options	The contract in the option self-hedge update is not an option contract
142	Option self-hedge not found	Option self-hedge to be operated on cannot be found
143	Option self-hedge operation must be deletion	Option self-hedge operation type error
144	This client's SelfCloseFlag cannot be reserved option position	SelfCloseFlag in option self-hedge update does not match client type
145	This client's SelfCloseFlag cannot be self-hedge option position	SelfCloseFlag in option self-hedge update does not match client type
146	Only holders of long positions can exercise	Only option buyers can submit option exercise insertion requests
147	User's new password does not meet requirements (at least 8 characters, must include digits, uppercase and lowercase letters)	New password must meet complexity requirements during modification (minimum 8 characters with digits, uppercase and lowercase letters)
148	Market price is within a reasonable spread range, and quote request is unnecessary	If market price is within a reasonable spread, the client's quote request will not be sent to market makers, namely, the client's quote request is meaningless
149	Option abandonment applications can only be submitted on option expiration day	Option abandonment events for an option can only be submitted on the option's expiry date
150	Proprietary member has not authenticated or authentication failed before login	Proprietary members must complete terminal authentication before login
151	Version verification failed	API version verification failed
153	Market orders must be GFD or IOC orders	Validity type of market order is neither IOC nor GFD during insertion
154	Market orders must be entered during continuous trading	Contract status is not in continuous trading phase during market order insertion
155	Market orders are supported only for futures and options	Product type is neither futures nor options during market order insertion
997	Api authentication failure	Illegal API access
	Api crypt info failure	API encryption information query failed
998	Query frequency is too high	Query frequency is too high
999	The last query result is on way	There are pending query response data yet to be sent
1005	No record	During various operations, the record corresponding to the contract is missing

2. Enumeration Value List

Serial No.	Description of enumeration	Prefix of enumera-tion	Name of enumeration	Code description	Code name	Code No.
1	Trading role	ER	TradingRole	Broker	Broker	1
				Proprietary trading	Host	2
2	Product type	PC	ProductClass	Futures	Futures	1
				Option	Options	2
				Portfolio	Combination	3
				Spot	Spot	4
				EFP	EFP	5
				Settlement price trading; trading at settlement	TAS	6
				Arbitrage	Spread	7
3	Option type	OT	OptionsType	Non-option	NotOptions	0
				Bullish (call)	CallOptions	1
				Bearish (put)	PutOptions	2
4	Trading status of contract	IS	InstrumentStatus	Pre-opening	BeforeTrading	0
				Non-trading	NoTrading	1
				Continuous trade	Continous	2
				Call autction order	AuctionOrdering	3
				Call autction balancing	AuctionBalance	4
				Matching of call auction	AuctionMatch	5
				Close	Closed	6
				Transaction processing	TransactionProcessing	7
5	Buy-sell direction	D	Direction	Bid	Buy	0
				Ask	Sell	1
6	Type of open interest	PT	PositionType	Net position	Net	1
				Gross position	Gross	2
7	Direction of long and short open interest	PD	PosiDirection	Net	Net	1
				Long	Long	2
				Short	Short	3
8	Hedge flag	HF	HedgeFlag	General	Speculation	1
				Hedge	Hedge	3
				none	None	N
9	Type of client	CT	ClientType	Natural person	Person	0
				Legal person	Company	1
				Investment fund	Fund	2
10	Reasons for contract to enter the trading status	IER	InstStatusEnterReason	Auto-switch	Automatic	1
				Manual switch	Manual	2
				Fusing	Fuse	3
				Fuse mannually	FuseManual	4
11	Conditions of order price	OPT	OrderPriceType	Arbitrary price	AnyPrice	1
				Price limit	LimitPrice	2
				Best price	BestPrice	3
12	Offset flag	OF	OffsetFlag	Position opening	Open	0
				Closing-out of position	Close	1
				Forced closing-out	ForceClose	2
				Closing out today’s position	CloseToday	3

Serial No.	Description of enumeration	Prefix of enumera-tion	Name of enumeration	Code description	Code name	Code No.
				Closing out yesterday's position	CloseYesterday	4
				none	None	N
13	Reasons for forced closing-out	FCC	ForceCloseReason	Non-forced closing out	NotForceClose	0
				Insufficient fund	LackDeposit	1
				Client exceeded the position limit	ClientOverPositionLimit	2
				Member exceeded the position limit	MemberOverPositionLimit	3
				Position is not the integral multiple	NotMultiple	4
				Market abuse	Violation	5
				Others	Other	6
				Person near the delivery day	PersonDeliv	7
				Hedge volume over position limit	HedgeOverPositionLimit	8
14	Status of order	OST	OrderStatus	Fulfilled	AllTraded	0
				Part of transaction is still in the queue	PartTradedQueueing	1
				Part of transaction is not in the queue	PartTradedNotQueueing	2
				The unfulfilled is still in the queue	NoTradeQueueing	3
				The unfulfilled is not in the queue	NoTradeNotQueueing	4
				Order cancellation	Canceled	5
15	Type of order	ORDT	OrderType	Normal	Normal	0
				Quote derivatives	DeriveFromQuote	1
				Portfolio derivatives	DeriveFromCombination	2
16	Type of valid period	TC	TimeCondition	Immediate or cancel order	IOC	1
				Good for this session	GFS	2
				Good for the day	GFD	3
				Good till date	GTD	4
				Good till canceled	GTC	5
				Good for call auction	GFA	6
17	Volume type	VC	VolumeCondition	Any quantity	AV	1
				The Min. quantity	MV	2
				Total number	CV	3
18	Trigger conditions	CC	ContingentCondition	Immediately	Immediately	1
				Stop-loss	Touch	2
19	Operation flag	AF	ActionFlag	Deletion	Delete	0
				Suspension	Suspend	1
				Activation	Active	2
				Modification	Modify	3

Serial No.	Description of enumeration	Prefix of enumera-tion	Name of enumeration	Code description	Code name	Code No.
20	Source of orde	OSRC	OrderSource	From participants	Participant	0
				From administrator	Administrator	1
21	Transaction type	TRDT	TradeType	Common transaction	Common	0
				Option execution	OptionsExecution	1
				Transaction of OTC	OTC	2
				Transaction of EFP derivatives	EFPDerived	3
				Transaction of portfolio derivatives	CombinationDerived	4
				Block trade execution	BlockTrade	5
				Arbitrage-derived execution	SpreadDerived	6
22	Source of transaction price	PSRC	PriceSource	Previous transaction price	LastPrice	0
				Bid price	Buy	1
				Ask price	Sell	2
				Derived price	Imply	3
23	Execution result	OER	ExecResult	Not executed	NoExec	n
				Already canceled	Canceled	c
				Execution sucessful	OK	0
				Position of option is inadequate	NoPosition	1
				Fund is inadequate	NoDeposit	2
				Member doesn’t exist	NoParticipant	3
				Client doesn’t exist	NoClient	4
				Contract doesn’t exist	NoInstrument	6
				No authorization to execute	NoRight	7
				Unreasonable quantity	InvalidVolume	8
				No adequate historical transaction	NoEnoughHistoryTrade	9
24	Whether to keep the position mark after the option is exercised	EOPF	ExecOrderPositionFlag	Reserved	Reserve	0
				Not reserved	UnReserve	1
25	Whether position is closed automatically after option exercrised	EOCF	ExecOrderCloseFlag	Close position automatically	AutoClose	0
				Not closed	NotToClose	1
26	Whether option exercise is of	OSCF	OptSelfCloseFlag	Self-hedge option position	CloseSelfOptionPosition	0

Serial No.	Description of enumeration	Prefix of enumera-tion	Name of enumeration	Code description	Code name	Code No.
	self-hedge type			Retained option position	ReserveOptionPosition	1
				Self-hedge futures position generated after option seller exercise	SellCloseSelfFuturePosition	2

3. Data Type List

Name of data type	Basic data type	Description of data type
TShfeFtdcErrorIDType	int	Error ID
TShfeFtdcPriorityType	int	Priority
TShfeFtdcSettlementIDType	int	Settlement number
TShfeFtdcMonthCountType	int	Number of month
TShfeFtdcTradingSegmentSNType	int	numberof trading sessions
TShfeFtdcVolumeType	int	Quantity
TShfeFtdcTimeSortIDType	int	Sequence numberof queue by time
TShfeFtdcSequenceNoType	int	Sequence number
TShfeFtdcBulletinIDType	int	Bulletin number
TShfeFtdcMillisecType	int	Time (millisecond)
TShfeFtdcVolumeMultipleType	int	Contract multiplier
TShfeFtdcParticipantIDType	char[11]	Member ID
TShfeFtdcUserIDType	char[16]	Transaction user’s ID
TShfeFtdcPasswordType	char[41]	Password
TShfeFtdcClientIDType	char[11]	Client ID
TShfeFtdcInstrumentIDType	char[31]	Contract ID
TShfeFtdcProductIDType	char[9]	Product ID
TShfeFtdcDateType	char[9]	Date
TShfeFtdcTimeType	char[9]	Time
TShfeFtdcInstrumentNameType	char[21]	Contract name
TShfeFtdcProductGroupIDType	char[9]	Product suite’s ID
TShfeFtdcMarketIDType	char[9]	Market ID
TShfeFtdcSettlementGroupIDType	char[9]	Settlement group’s ID
TShfeFtdcOrderSysIDType	char[13]	Order number
TShfeFtdcExecOrderSysIDType	char[13]	System number of option exercise
TShfeFtdcQuoteSysIDType	char[13]	Quoto number
TShfeFtdcTradeIDType	char[13]	Transaction number

Name of data type	Basic data type	Description of data type
TShfeFtdcOrderLocalIDType	char[13]	Local order number
TShfeFtdcComeFromType	char[21]	Source of message
TShfeFtdcAccountIDType	char[13]	Fund account
TShfeFtdcNewsTypeType	char[3]	Bulletin type
TShfeFtdcAdvanceMonthType	char[4]	Month in advance
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
TShfeFtdcProductClassType	char	Product type
TShfeFtdcOptionsTypeType	char	Option type
TShfeFtdcInstrumentStatusType	char	Trading status of contract
TShfeFtdcDirectionType	char	Buy-sell direction
TShfeFtdcPositionTypeType	char	Type of open interest
TShfeFtdcPosiDirectionType	char	Direction of long and short open interest
TShfeFtdcHedgeFlagType	char	Hedge flag
TShfeFtdcClientTypeType	char	Type of client
TShfeFtdcInstStatusEnterReasonType	char	Reasons for contract to enter the trading status
TShfeFtdcOrderPriceTypeType	char	Conditions of order price
TShfeFtdcOffsetFlagType	char	Offset flag
TShfeFtdcForceCloseReasonType	char	Reasons for forced closing-out
TShfeFtdcOrderStatusType	char	Status of order
TShfeFtdcOrderTypeType	char	Type of order
TShfeFtdcTimeConditionType	char	Type of valid period
TShfeFtdcVolumeConditionType	char	Volume type
TShfeFtdcContingentConditionType	char	Trigger conditions
TShfeFtdcActionFlagType	char	Operation flag
TShfeFtdcOrderSourceType	char	Source of order
TShfeFtdcTradeTypeType	char	Transaction type
TShfeFtdcPriceSourceType	char	Source of transaction price
TShfeFtdcExecResultType	char	Execution result
TShfeFtdcYearType	int	Year
TShfeFtdcMonthType	int	Month

Name of data type	Basic data type	Description of data type
TShfeFtdcBoolType	int	Bool type
TShfeFtdcPriceType	double	Price
TShfeFtdcUnderlyingMultipleType	double	Contract multiplier for basic commodity
TShfeFtdcCombOffsetFlagType	char[5]	Combination offset flag
TShfeFtdcCombHedgeFlagType	char[5]	Combination hedge flag
TShfeFtdcRatioType	double	Ratio
TShfeFtdcMoneyType	double	funds
TShfeFtdcLargeVolumeType	double	Large quantity
TShfeFtdcNewsUrgencyType	char	Urgency
TShfeFtdcSequenceSeriesType	short	Serial number in sequence
TShfeFtdcErrorMsgType	char[81]	Error message
TShfeFtdcAbstractType	char[81]	Message digest
TShfeFtdcContentType	char[501]	Message body
TShfeFtdcURLLinkType	char[201]	WEB address
TShfeFtdcIdentifiedCardNoType	char[51]	Certificate number
TShfeFtdcIdentifiedCardNoV1Type	char[21]	Original certificate number
TShfeFtdcPartyNameType	char[81]	Name of party involved
TShfeFtdcIdCardTypeType	char[16]	Type of certificate
TShfeFtdcDataCenterIDType	int	Datacenter ID
TShfeFtdcBusinessLocalIDType	int	Local business ID
TShfeFtdcCurrencyIDType	char[4]	Currency ID
TShfeFtdcRateUnitType	int	Exchange Rate Unit Type
TShfeFtdcExRatePriceType	double	Exchange Rate Price
TShfeFtdcExecOrderPositionFlagType	char	Flag indicating whether to retain the futures position after option exercise
TShfeFtdcExecOrderCloseFlagType	char	Whether the futures position generated from option exercise is self-hedge
TShfeFtdcMacAddressType	char[21]	MAC address information
TShfeFtdcOptionSelfCloseSysIDType	char[13]	Option self-hedge system ID
TShfeFtdcOptSelfCloseFlagType	char	Whether the position exercised by the option is self-hedge
TShfeFtdcAuthIDType	char[17]	Terminal authentication authorization ID type

4. API Return Value List

Return value	Return value meaning
0	Success

-1	Not logged in
-2	Exceeded in-transit flow control
-3	Exceeded request flow control
-4	File not found or file read/write failure
-5	Already logged in or duplicate call
-6	Mandatory field is empty
-7	Authentication enabled but authentication failed
-8	Exceeded maximum number of items
-9	Not initialized
-10	Already initialized
-11	Duplicate ID
-12	Not yet connected to the front-end
-13	Member ID mismatch
-14	User ID mismatch