APEX Trading System
Trading API Interface Specifications

Version 1.07

File Version	Modified Date	Notes
V1.0	12th Dec 2016	Version 1.0
V1.01	19th Apr 2017	Change: 1. Removed CombOffsetFlag from ReqOrderInsert Method since it will be ignored by Trading Engine. 2. Miscellaneous changes.
V1.02	30th June 2017	Change: 1. Added supported order conditions combination to order type mapping in Appendix 8.4.
V1.03	20th July 2017	Change: 1. Use "Business Unit" field in order and trade to store sub-account or trading account ID.
V1.04	26th Dec 2018	Change: 1. Updated document based on ApexTraderAPI Win32 v1.2.6 L3K a. Updated function prototype, struct, ErrorID and ErrorMsg. b. Removed methods that do not exist in the code c. Updated supported order types (removed "market to limit" order type). 2. Improved formatting. 3. Changed logo in header and added page number in footer.
V1.05	15th Jan 2019	Change: 1. Improved formatting. 2. Fixed typos.
V1.06	20th Jan 2020	Change: 1. Added APIs related to spread trading
V1.07	6th Jun 2021	Change: 1. Added OnRtnMarketData to update previous settlement price

Contents

1. Introduction

1.1. TraderAPI Overview

TraderAPI is a C++ based class library. Application program can use and extend the interfaces provided by the class library to implement all trading functions, including order input, order cancellation, order query, trade query, member's client query, member's position query, client's position query, contract query, contract trading status query, Exchange bulletin query, etc.

The class library contains the following 6 files:

File Name	File Description
APEXFtdcTraderApi.h	Header File for Trading Interface
APEXFtdcUserApiStruct.h	Defines a series of business-related data structure header files
APEXFtdcUserApiDataType.h	Defines a series of data type header files required by the API
APEXtraderapi.dll	Dynamic-link library (DLL) binary file
APEXtraderapi.lib	Import library (.Lib) file
APEXtraderapi.so	Dynamic library of Linux

Windows API version supports **MS VC 6.0** and **MS VC.NET 2003** compiler and requires multi-threading compilation option/MT. Liunx API version is based on Redhat 6.3, gcc 4.4.6 and depends on OpenSSL library.

Note: During the process of developing Member System, attention should be paid to the "**Businesses Unavailable in Current Version**" and specific description of each function.

1.2. Supported Platforms of TraderAPI

- Intel X86/WindowsXP: including **.h** files, **.dll** files and **.lib** files
- Linux RedHat6.3: including **.h** files and **.so** files

2. Architecture

TraderAPI communicates with trading gateway of APEX Trading System via FTD Protocol that is based on TCP Protocol. Trading gateway is designed to handle trading businesses from Member System rather than market data distribution that is handled by market data gateway.

2.1. Communication Mode

FTD involves the following three communication modes:

- Dialog Communication Mode
- Private Communication Mode
- Broadcast Communication Mode

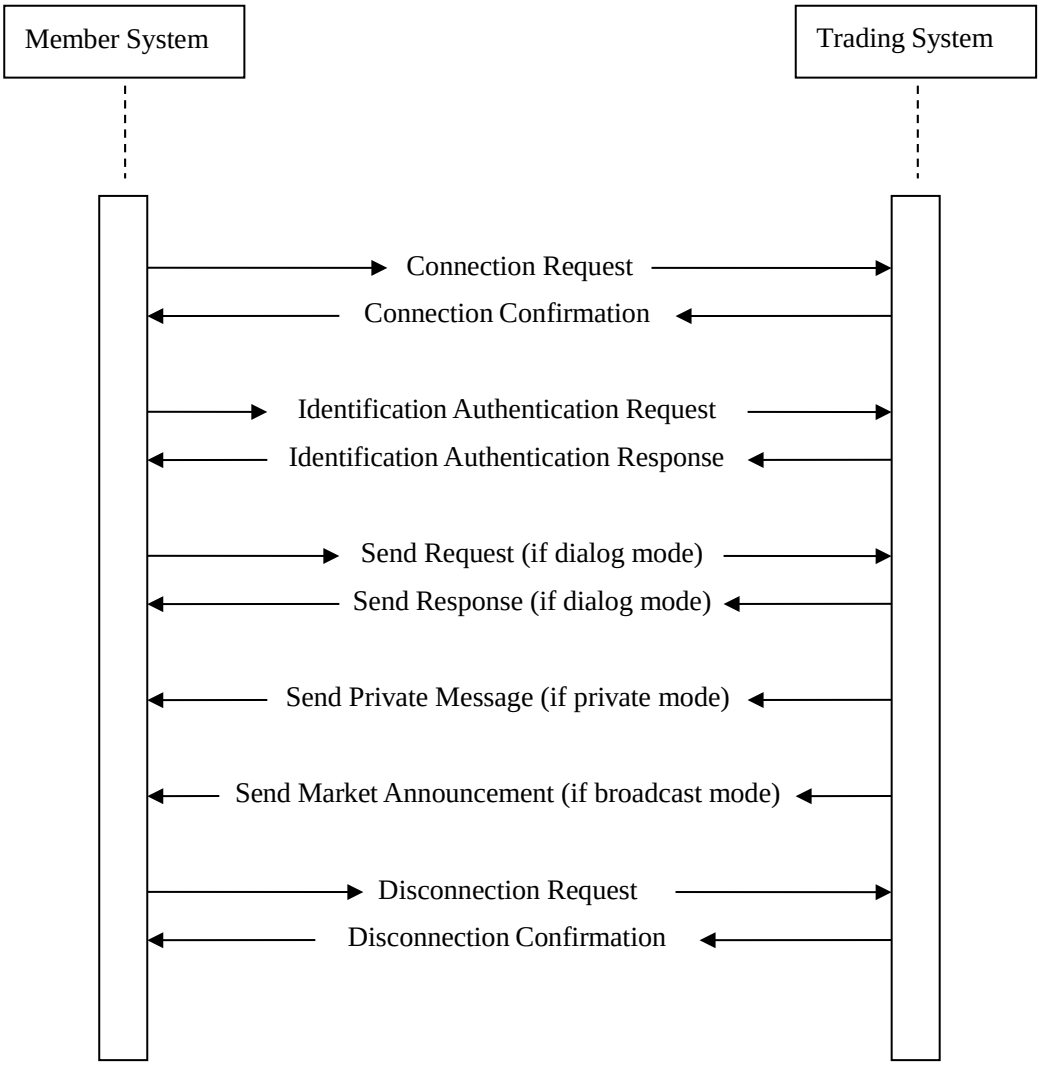
Dialog Communication Mode refers to communication whereby requests are initiated by Member System. Such requests (e.g. order, query, etc.) are received and processed by the Trading System and responses are sent back to the Member System. This is similar to the usual client/server mode.

Private Communication Mode means that trading system sends information (e.g. trade return) on its own initiative to a particular member or a particular trader of a particular member.

Broadcast Communication Mode means that trading system sends the same information (e.g. bulletin, public information in market etc.) to all traders.

Network connection and communication mode do not necessarily represent a simple one-to-one relationship. That is to say, messages of different communication modes can be sent within one network connection, while messages of one communication mode can also be delivered within different connections.

In any of the communication modes, the communication process is the same, as depicted below:



2.2. Data Stream

Trading gateway supports dialog communication mode, private communication mode, and broadcast communication mode. The market data distribution function of market data gateway supports dialog communication mode and broadcast communication mode.

1) Dialog Communication Mode

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

Dialog data stream is a bidirectional data flow through which Member System sends trading request and the Trading System feeds back response. Trading System does not maintain the status of the dialog stream. In the event of a system failure, dialog data stream is reset and the data in transit might be lost.

Query data stream is also a bidirectional data flow through which member system sends query request and the Trading System feeds back response. Trading System does not maintain the status of the query stream. In the event of a system failure, query data stream is reset and the data in transit might be lost.

2) Private Communication Mode

In Private Communication Mode, data stream is reliable. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within private data stream by specifiying a starting sequence number. Private data stream provides Member System with order status report, trade return message etc. Private data stream is classified into **member's private stream** and **trader's private stream**.

Trading system maintains a private data stream for each member. All return messages for a particular member such as order return and trade return, will be released through member's private stream. Only authorized traders can subscribe member's private stream.

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

3) Broadcast Communication Mode

Broadcast Communication Mode supports public data stream.

Public data stream is a uni-directional data stream that is sent from trading system or market data system to Member System for delivering public market information. Public data stream is a reliable data stream. Trading System maintains all public data streams within the system. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within public data stream by specifying a starting sequence number.

Take market data as an example. Market data stream is a public data stream that is sent from Trading System to Member System for delivering market data information. Market data stream is a reliable data stream. Trading System maintains all the market data streams. Within a trading day, when Member System resumes its connection after a disconnection, Member System can request the trading system to resend the data within public data stream by specifying a starting sequence number.

Market data provided by the Trading System is organized according to topics. Each topic covers market data for a particular group of contracts, as well as market data release contents and release methods, including market depth, sample frequency, delay time etc. The Exchange announces the specific contents of each topic of market data and the topic of market data that can be subscribed by each market data user. Each market data topic corresponds with one market data stream.

In order to get market data, Member System must subscribe to one or more market data release topics after connecting with the gateway.

3. Interfaces

TraderAPI provides two interfaces, namely **CApexFtdcTraderApi** and **CApexFtdcTraderSpi**. These two interfaces encapsulate FTD Protocol. Member system can send operating requests via **CApexFtdcTraderApi** and it can handle/process the response and reply from the APEX Trading System by inheriting **CApexFtdcTraderSpi** and overriding the callback functions.

3.1 Dialog Stream and Query Stream Programming Interface

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

```
   ////Request:
int CApexFtdcTraderApi::ReqXXX (CApexFtdcXXXField *pReqXXX,
                                int nRequestID)


   ////Response:
Void CApexFtdcTraderSpi::OnRspXXX(CApexFtdcXXXField *pRspXXX,
                                  CApexFtdcRspInfoField *pRspInfo,
                                  int nRequestID,
                                  bool bIsLast)
```

The 1st parameter for the request interface is the requested content, and it cannot be left empty. This parameter accepts a certain 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 the classes.

The 2nd parameter of the request interface is the request ID. The request ID is maintained by Member System and every request ID should be unique. The request ID filled in upon sending the request will be sent back to Member System together with the response from the APEX Trading System, and Member System can match a particular request with its corresponding response by using this number.

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

The callback function/method requires 4 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 response info, indicating whether 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 Private Stream Programming Interface

As described in section 2.2, private stream returns private information of a particular Exchange Member or a particular trader, including order return, trade return, etc.

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

```
void CApexFtdcTraderSpi::OnRtnXXX(CApexFtdcXXXField *pXXX);
////or
void CApexFtdcTraderSpi::OnErrRtnXXX(CApexFtdcXXXField *pXXX,
                                     CApexFtdcRspInfoField *pRspInfo);
```

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

3.3 Public Stream Programming Interface

Public stream returns public data from the Exchange, including convention, declaration etc.

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

```
   void CApexFtdcTraderSpi::OnRtnXXX(CApexFtdcXXXField *pXXX);
```

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

4. Operating Mode

4.1. Workflow

The interaction between Member System and the APEX 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 System has to complete the steps below (for more details, please refer to the codes in the **Development Example** section).

Steps	**Member System**
1	Generate an instance of **CApexFtdcTraderApi**
2	Generate an event handler instance
3	Register an event handler instance
4	Subscribe to the private stream; Subscribe to the public stream;
5	Set the network address for the trading gateway and/or **NameServer** [1]
6	Initialization

[1]In order to be compatible with the previous version, this API still provides interfaces for the registration of the trading gateway (and market data gateway). However, APEX recommends not using these interfaces directly, which will be removed in the next version. Please refer to Section 4.9 **Gateway List** for more details of the **NameServer**.

4.1.2. Function Calling Phase

In the function calling phase, Member System can call any of request methods from the **CApexFtdcTraderApi** interface, e.g. ReqUserLogin, ReqOrderInsert etc. and also provide callback functions to respond to 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 codes, please refer to the **Appendix** for **Error Code List**.

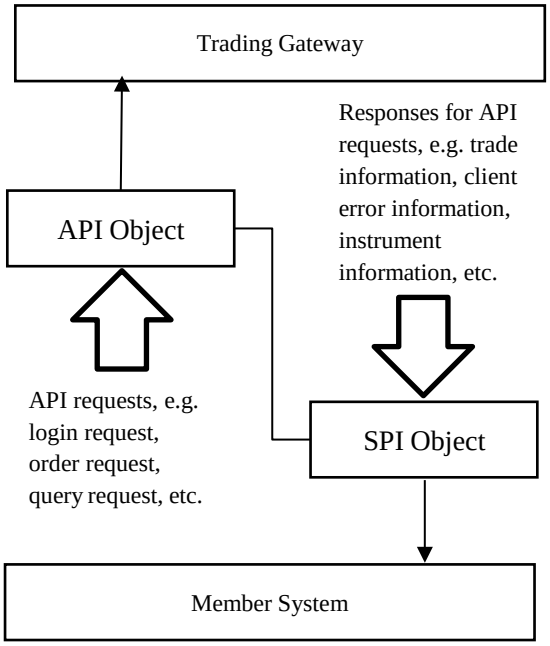
4.2. Working Thread

The application program of Member 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). The communication between the application program and the trading gateway or market data gateway is driven by the API working thread.

The interface provided by **CApexFtdcTraderApi** is thread-safe. Multiple application programs are allowed to send requests simultaneously.

The interface callback provided by **CApexFtdcTraderSpi** is driven by the TraderAPI working thread. It receives the required data from the gateways of the Trading System by implementing the interface method of SPI.

If the callback function of the Member System application program blocks, TraderAPI working thread will also be blocked. In this case, the communication between API and trading gateway will stop, therefore quick return is required for callback functions. In the callback functions of derived classes of **CApexFtdcTraderSpi**, the quick return can be achieved by storing the data into the buffer or via the Windows messaging mechanism.



4.3 Interaction between Member System and the Trading System via TraderAPI

Member System interacts with the Trading System through TraderAPI. Requests from Member System are sent to the Trading System through TraderAPI, reply and return messages from the Trading System are sent to Member System through TraderAPI as well.

Dialog stream interface, query stream interface and private stream interface of TraderAPI are interrelated. For instance, after user enters an order by **ReqOrderInsert**, order response **OnRspOrderInsert** is received immediately**,** which indicates that the Trading System has received the order. After the order enters the Trading System, if the order's trading status changes, an order return message **OnRtnOrder** will be received**.** If the order is matched (including partially matched and completely matched), trade return (or transaction return) message **OnRtnTrade** will be received. Meanwhile, the order and trade return (or transaction return) messages of one user will also be received by other authorized uesrs of the same member as this user.

Let's illustrate the concept with a day-to-day trading example. Assuming there are two Member Systems A and B, the following events occur:

1) **Trader A places an order, with details: PF1906, buy, 10 lots, USD 560, Local ID 1**

   - **CApexFtdcTraderApi::ReqOrderInsert:** Order entry request. This function is called by the main application thread of Member System, and the request is sent to the gateway of the Trading System through dialog stream.
   - **Trading System Order Processing:** The order's System ID is numbered 1. Since there is no counterparty in matching queue at the moment, the order status is "**Not Traded and Still Queuing**". The gateway of the Trading System sends 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 Trader A's member. Both the order response and the order return message are processed by calling the SPI object methods in the TraderAPI working thread.
   - **CApexFtdcTraderSpi::OnRspOrderInsert:** The gateway 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 gateway of the Trading System.
   - **CApexFtdcTraderSpi::OnRtnOrder:** The gateway of the Trading System immediately provides order return to private stream of Trader A and the private stream of Trader A's Member . Other authorized traders are able to obtain the order details in the order return, e.g. order status, etc. This function is called by

the TraderAPI working thread after receiving the order return from the gateway of the Trading System. If there are other traders of Member A who login into the Trading System and receive private stream of Member A, they will receive the same order return message (similarly in the below case).

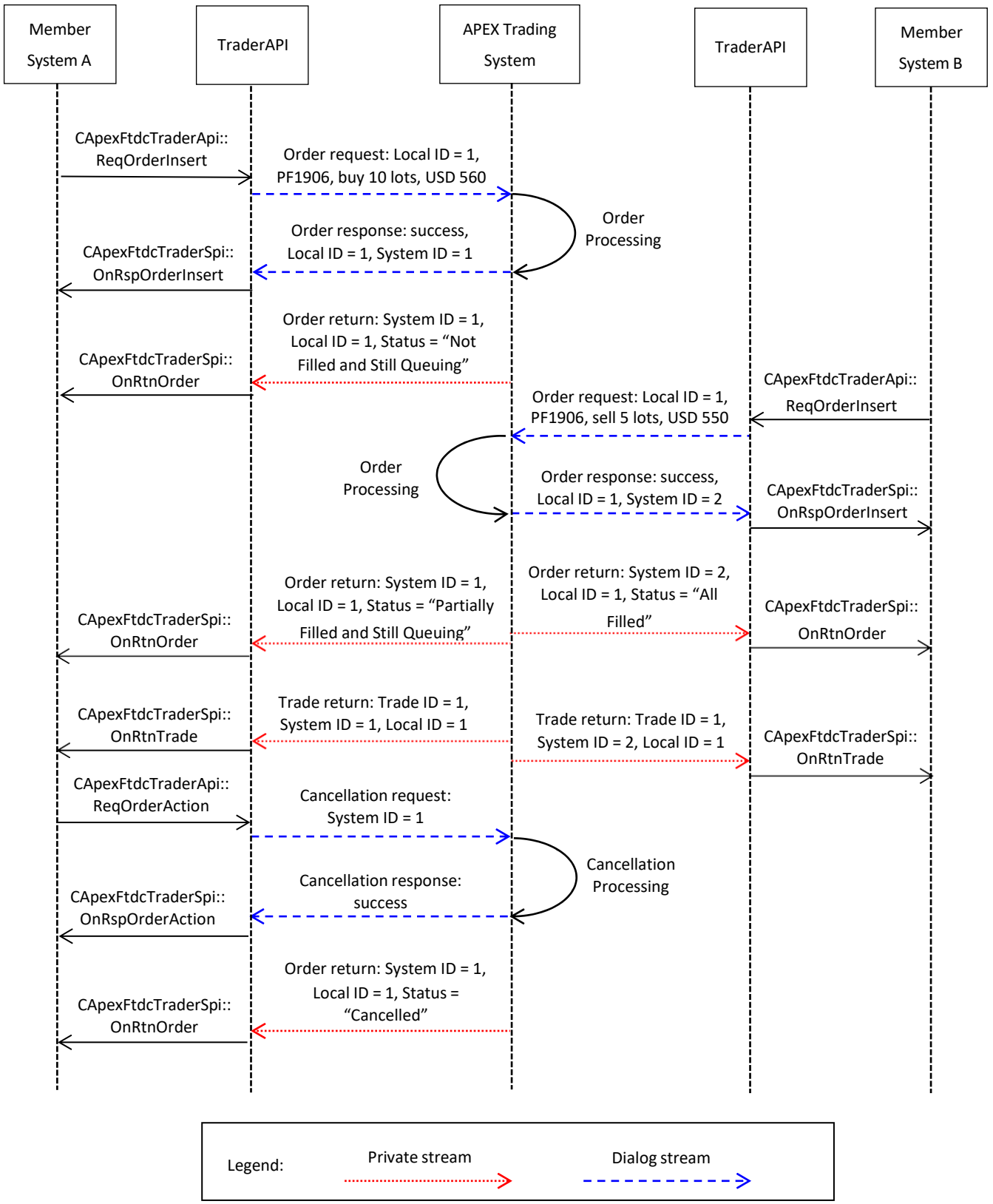
2) TraderB places an order, with details: PF1906, sell, 5 lots, USD 550, Local ID 1

- **CApexFtdcTraderApi::ReqOrderInsert:** Order entry request.
- **Order Processing of the Trading System:** The order's System ID is numbered 2. Since there is no counterparty in the queue waiting for matching, the order status is "Not Filled and Still Queuing".
- **Order Processing of the Trading System:** Matching is attempted and succeeds, thus the order status is "**All Filled**". The gateway 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 Trader B's member
  - ✓ order return to the private stream of Trader A and the private stream of Trader A's member, 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 "volume traded" is 5
  - ✓ trade return (or transaction return) to the private stream of Trader B and the private stream of Trader B's member
- **The Trading System ensures that:**
  - ✓ **order return is delivered to Member System ahead of the trade return (transaction return)**
  - ✓ "volume traded" field in order return has already reflected the updated amount in the order book of theTrading System, so there is no need to make adjustment again based on the volume in transaction return.
- **CApexFtdcTraderSpi::OnRspOrderInsert:** The trading gateway 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.
- **CApexFtdcTraderSpi::OnRtnOrder:** The trading gateway immediately provides order return to private stream of Trader B and the private stream of Trader B's member. The order status is "**All Filled**".
- **CApexFtdcTraderSpi::OnRtnOrder:** The trading gateway of APEX immediately provides order return to the private stream of Trader A and the private stream of Trader A's member. Order status is "**Partially Filled and Still Queuing**", and the "volume traded" is 5.

- **CApexFtdcTraderSpi::OnRtnTrade:** The trading gateway immediately provides trade return (or transaction return) to the private stream of Trader A and the private stream of Trader A's member.
- **CApexFtdcTraderSpi::OnRtnTrade:** The trading gateway of the Exchange immediately provides trade return (or transaction return) to the private stream of Trader B and the private stream of Trader B's member.

3) Trader A cancels the order

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


Member System A
TraderAPI
APEX Trading System
TraderAPI
Member System B
CApexFtdcTraderApi:: ReqOrderInsert
Order request: Local ID = 1, PF1906, buy 10 lots, USD 560
Order Processing
Order response: success, Local ID = 1, System ID = 1
CApexFtdcTraderSpi:: OnRspOrderInsert
Order return: System ID = 1, Local ID = 1, Status = "Not Filled and Still Queuing"
CApexFtdcTraderSpi:: OnRtnOrder
CApexFtdcTraderApi:: ReqOrderInsert
Order request: Local ID = 1, PF1906, sell 5 lots, USD 550
Order Processing
Order response: success, Local ID = 1, System ID = 2
CApexFtdcTraderSpi:: OnRspOrderInsert
Order return: System ID = 1, Local ID = 1, Status = "Partially Filled and Still Queuing"
CApexFtdcTraderSpi:: OnRtnOrder
Order return: System ID = 2, Local ID = 1, Status = "All Filled"
CApexFtdcTraderSpi:: OnRtnOrder
CApexFtdcTraderSpi:: OnRtnTrade
Trade return: Trade ID = 1, System ID = 1, Local ID = 1
Trade return: Trade ID = 1, System ID = 2, Local ID = 1
CApexFtdcTraderSpi:: OnRtnTrade
CApexFtdcTraderApi:: ReqOrderAction
Cancellation request: System ID = 1
Cancellation Processing
CApexFtdcTraderSpi:: OnRspOrderAction
Cancellation response: success
Order return: System ID = 1, Local ID = 1, Status = "Cancelled"
CApexFtdcTraderSpi:: OnRtnOrder
Legend:
Private stream
Dialog stream

4.4 Connection to the gateway of the Trading System

TraderAPI communicates with trading gateways of APEX via the FTD Protocol which is built upon the TCP. **TraderAPI** uses the **CApexFtdcTraderApi::RegisterFront** method to register the network address of the trading gateway.

APEX owns multiple trading and market data gateways, for both load balancing and backup purposes, to improve system performance and reliability. In order to guarantee the reliability for communications during trading hours, TraderAPI may register multiple gateways. After the API is initialized, it will randomly choose one gateway from the registered gateways and try to establish network connection with it. If the attempt fails, it will try other registered gateways one by one until the connection is successful. If there is network failure during trading process, the API will attempt to connect to the other gateways in a similar way.

APEX announces network addresses for at least 2 gateways (located in the Equinix SG3 or Singtel KC2). Hence, the Member System should register at least 2 gateway network addresses to prevent single point of failure resulting from the failure of the connected gateway.

APEX will use NameServer and will only publish NameServer addresses but not gateway addresses. TradeAPI uses the **CApexFtdcTraderApi::RegisterNameServer** method to register the network addresses of APEX NameServer. The method can be called multiple times to register multiple addresses.

4.5 Local Files

During runtime, **TraderAPI** writes 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 "**.con**". Different users should specify different local file path, otherwise they may not be able to receive some data from the Trading System.

4.6 Request-Reply Log Files

TraderAPI offers two logging interfaces for recording communication. **OpenRequestLog** is used to open the request log and **OpenResponseLog** is used to open the reply log. If the logs are opened, all service requests will be written into the request log, and all service reply and returns will be recorded into the reply log. Note that login request/reply and query request/reply are not logged to maintain confidentiality and save storage space.

Request Format

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

Reply (Message) Format
Date time, reply name, response ID, response content, [reply parameter name, reply parameter content]
Return (Message) Format
Date time, return name, [return parameter name, return parameter content]

4.7 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 the Trading System to the Member System in a reliable and orderly manner, are called reliable data streams. Reliable data streams ensure the correctness and completeness of the data in the Member System. For example, the Member System can obtain sufficient information through the return messages in the Member's private data stream to complete its business operation at the Member's end.

Reliable data stream relies on retransmission to ensure the reliability and order. The Member System is responsible for managing the Sequence ID of the data stream. In case of transmission interruption, the Member System resubscribe to the data stream from a specified Sequence ID. Data integrity can be ensured in this way.

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

The interface of the Trading System offers two methods for managing reliable data streams: retransmission Sequence ID managed by the API and retransmission Sequence ID managed by the Member System.

4.7.1 Retransmission Sequence ID Managed by API

Whenever API receives a message from reliable data stream, it (a) first calls the callback function in SPI to inform the Member System; (b) then records the message Sequence ID in the local file (with file extension ".con"). If the Member system resubscribes data stream after its logout, then the message sequence ID recorded in the local file can be used for subscription of the data stream.

SubscribePrivateTopic, SubscribePublicTopic, and **SubscribeUserTopic** from CApexFtdcTraderApi are used to subscribe to reliable data streams.

Retransmission mode can be designated via interface parameter, which is classified into three modes, namely, RESTART (retransmission), RESUME (resuming of a transmission) and QUICK (snapshot).

- **RESTART** mode starts the retransmission from the 1st message in the stream. The message Sequence ID recorded in the local file is ignored.
- **RESUME** mode starts the retransmission following the Sequence ID recorded in the local file. If it is a market data stream, the current market data snapshot of

each contract with the particular topic will be transmitted first, followed by market data transmission starting from the specified Sequence ID. In order to maintain the integrity of members' trading data, APEX recommends the "RESUME" mode for the private stream of the member or the trader.

- **QUICK** mode starts the retransmission at the maximum Sequence ID at the moment of subscribing the data stream. If it is a market data stream, the current market data snapshot of each contract/instrument with the particular topic will be transmitted first. The QUICK mode is mainly used for occasions in which there is no need to guarantee the data integrity. APEX does not recommend the use of QUICK method.

A certain degree of data inconsistency risk exists in the situation where the retransmission Sequence ID is maintained by the API. For example, if (a) is done but (b) is incomplete, a duplicate message will be received by the Member System, which will complicate the message processing in the Member System. Furthermore, if the local file which records the data stream Sequence ID is corrupted, all data streams have to be retransmitted, and this will probably affect the efficiency of the Member System.

If the API is utilized to maintain the Sequence ID of retransmission messages, it will record the 2 fields, **TradingDay** and **DataCenterID**, which are returned upon the previous login, into the file named **resume.con**. During login, the API will use the values in the file to overwrite these 2 fields filled by Member System.

4.7.2 Retransmission Sequence ID Managed by Member System

Whenever the API receives a message from the reliable data stream, it (a) first calls the **OnPackageStart** function of the SPI to inform the Member System that a message has been received, (b) then calls the callback function of the SPI to inform the Member System of the system business/service data, (c) finally calls the **OnPackageEnd** function of the SPI to inform the Member System that the callback of the message is completed. From the functions **OnPackageStart** and **OnPackageEnd**, the Member System can obtain the Sequence ID of the current callback message, and record the Sequence ID if necessary. When retransmitting the reliable data stream, the recorded Sequence ID can be provided to the **CApexFtdcTraderApi::ReqSubscribeTopic** function (similar to the RESUME mode).

Using the **CApexFtdcTraderApi::ReqSubscribeTopic** function, the Member System can specify the message Sequence ID for data stream retransmission. If the Sequence ID is 0, the entire data stream will be retransmitted (similar to RESTART mode); if the specified Sequence ID is -1, the message retransmission will start from the largest Sequence ID at the moment of subscription (similar to the QUICK mode).

If the subscribed stream is the market data stream, and if the specified retransmission Sequence ID is not 0, the market data snapshots for all the contracts prior to the specified Sequence ID will be transmitted. During the transmission of the market data snapshots, the **nSequenceNo** parameter value for the callback function **OnPackageStart** and **OnPackageEnd** is 0.

The retransmission Sequence ID maintained by the Member System is more consistent and reliable than that maintained by the API. This method should be used for the Member System which requires high level of transactional integrity.

Note: upon login, **TradingDay** and **DataCenterID** should be filled in using the return value from the previous login reply. If it is the first login or resuming transmission is not required, TradingDay can be set as an empty string, and DataCenterID can be filled in as 0 or the primary data center ID published by APEX.

4.8 Heartbeat Mechanism (Heartbeat)

The TCP virtual link is used for communication between the Member System and the gateways of the Trading System. If virtual link failure occurs and there is no data communication between Member System and the gateway during the dysfunction period, specifically, both sides do not call the functions **Socket recv()** and **Socket send()**, then both sides (Member System and the Trading System) will not be able to detect the working status at that moment, and need to wait for the **Socket** timeout. Generally, the timeout periods defined by operating systems are relatively long, which are not for real-time monitoring. Monitoring is crucial in accelerating the response speed and realizing the automatic recovery and processing.

One possible way to monitor the working status of two communicating sides is to add extra heartbeat information. The principle is quite simple and it will not incur additional cost for both sides. When there is business data transmission, both sides can detect the status of the virtual link and communication. When there is no business data transmission, the two sides need to send heartbeat messages to each other (in this case, no data is transferring along the virtual link, and hence the additional heartbeat messages will cause no pressure on bandwidth and cost as well). Although no additional communication cost is required for the server (e.g. for the gateway of the Trading System), the patrol cost (monitoring every second to find whether it is required to send heartbeat information and maintaining the connection table) increases linearly as the number of connections increases.

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, the TCP virtual link is considered invalid and it 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. Typically, the **timeout** is three times of the **interval**.

The API provides the **void SetHeartbeatTimeout(unsigned int timeout)** method for Member System to set the timeout period to monitor the validity of the TCP virtual link. During idle period, the Trading System sends heartbeat message to API every (**timeout-1**) / 3 seconds. If no message is received from the Trading System in more than **timeout**/2 seconds, the callback **CApexFtdcTraderApi::OnHeartBeatWarning()** will be triggered. If no message is received from the Trading System after **timeout** seconds, TCP connection will be interrupted and the callback function **CApexFtdcTraderApi::OnFrontDisconnected()** will be triggered.

For instance, assume that the Member System sets the heartbeat timeout period to be 16 seconds. The Trading System sends one heartbeat message to the API every 5 seconds during idle time. If API does not receive any message from the Trading System in 8 seconds, the callback function **CApexFtdcTraderApi::OnHeartBeatWarning()** will be triggered. If no message is received in 16 seconds, API will take the initiative to disconnect and trigger **CApexFtdcTraderApi::OnFrontDisconnected()**. In this case, the Member System can choose to reconnect with the gateway via alternative dedicated data link.

The gateway of APEX also monitors the TCP connection of the Member System via the heartbeat mechanism. If Member System does not call the **SetHeartbeatTimeout** method, the current timeout is fixed to 10 seconds. If the Member System calls the **SetHeartbeatTimeout** method, the same timeout setting will be synchronized to the gateway. After the link interruption, the gateway automatically disconnects with the member-side TCP link within acceptable time (about timeout + 5 seconds), so that the Member System can use alternative link (with a different IP address) to login, otherwise the gateway will hold that the original TCP connection is still valid and reject any login from the alternative address. This convenience is not available for the Member Systems using OFPv2 as they have to wait 60-90 seconds to log in from the alternative address.

Note:

If Member System never calls the **SetHeartbeatTimeout** method, after the API has initialized and established TCP connection to the gateway, it will automatically call the **SetHeartbeatTimeOut()** method and set the timeout to 10 seconds. The minimum value permissible for timeout parameter is 4 seconds. If the timeout parameter is set too high, in the situation of link disruption, Member system will have to take a much longer time to switch to the alternative link. If the timeout parameter is set too low, unexpected switching might occur. Therefore, the performance of the Member System and the network status should be taken into consideration when setting the timeout parameter.

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

4.9 Gateway List

For fault tolerance and load balancing, APEX deploys two groups of gateways at both the main data center and the backup data center. APEX publishes a list of the gateway network addresses. The Member System can randomly choose a gateway from the list to attempt to establish connection with it. The Member System can only connect to one gateway at a certain moment. If the connected gateway encounters a problem and results in connection failure or timeout, the Member System should try the other gateways in the list.

There are two ways for Member System to obtain the gateway list:

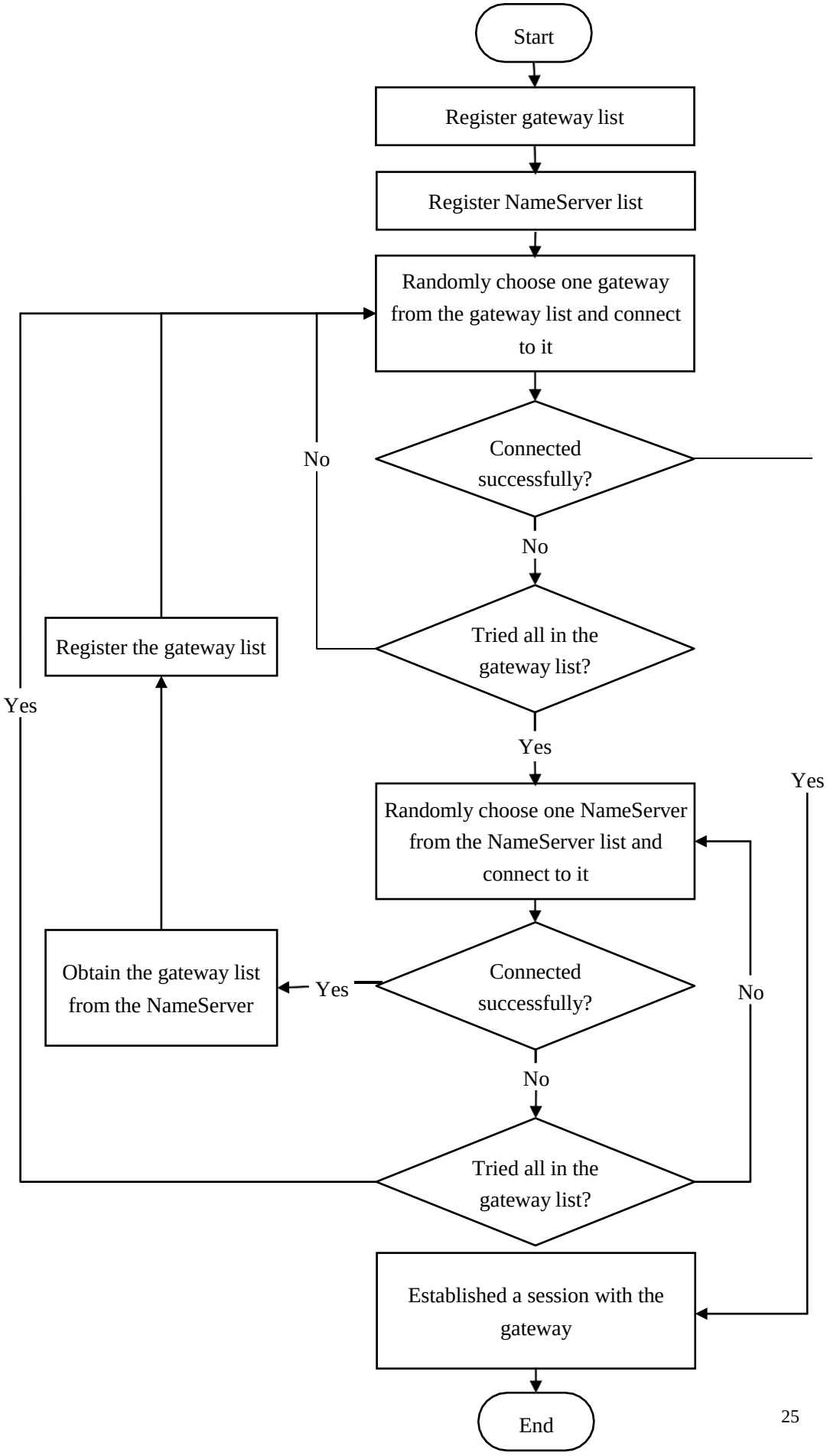
1) APEX announces the gateway list. The Member System registers the gateways of the list one by one into the API via the **RegisterFront** interface of API.
2) The Trading System provides **NameServer** to publish the gateway list for the API. APEX firstly announces the **NameServer** list, then the Member System registers the NameServer list into the API via the **RegisterNameServer** interface. The API first attempts to obtain the gateway list from the **NameServer**, then it will connect to one gateway based on the gateway list.

Advantages of employing NameServer include:

- APEX has more flexibility in gateway deployment. It can deploy additional gateways within a short time according to business requirement and load, without making any modification to the Member System.
- **NameServer** provides a better way for switching between the main system and disaster recovery system.
- **NameServer** is characterized by its unique function, simple structure and low load. There is no need to worry about load balancing. Hence, it can be deployed in a flexible way.

The Member System can simultaneously use the **RegisterFront()** method to register the gateway list, and use the **RegisterNameServer()** method to register the **NameServer** list. API will first attempt to connect to its existing registered gateway. If unsuccessful, it will try to connect the NameServer.

The flow chart for the API to connect to the gateway is as below:


Start
Register gateway list
Register NameServer list
Randomly choose one gateway from the gateway list and connect to it
Connected successfully?
No
No
Tried all in the gateway list?
Register the gateway list
Yes
Yes
Randomly choose one NameServer from the NameServer list and connect to it
Yes
Obtain the gateway list from the NameServer
Yes
Connected successfully?
No
No
Tried all in the gateway list?
Established a session with the gateway
End

4.10 Disaster Recovery Interface

APEX employs two data centers, namely Equinix SG3 data center and Singtel KC2 data center. The two data centers use high-speed optical fiber to connect each other. The APEX Trading System runs simultaneously at the two data centers. The main data center is responsible for business processing, and the backup center asynchronously receives the data from the main center and synchronizes the business with the main center. The backup data center is in the standby mode under normal circumstances.

If the main data center encounters a disastrous event, the business will be switched to the backup data center. The backup data center takes over the work of the main data center, and continues the business processing. During the data center switching, part of the business data might be lost. The Member System needs to know the orders to be cancelled via the API interfaces.

1) "Data Center ID" field is added to the API user login request interface to identify the data center ID of the previous login. "Data Center ID" field is also added to the user login response interface and the Trading System will send back the currently used Data Center ID. Member System should save the Data Center ID sent back from the Trading System, and fill it into the login request at the next login.
2) The "transaction cancellation" interface (**OnRtnFlowMessageCancel**) is added to the API. This interface is used to notify the to-be-cancelled messages from the subscribed topic after the member side sends out the subscription request. According to this interface, the Member System can get the Sequence ID of the message that is cancelled, and thus find the original message. The Sequence ID of the original message can be obtained through the **OnPackageStart** and **OnPackageEnd** interface.

5. Categories of TraderAPI Interfaces

5.1. Management Interfaces

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

Interface Type	Interface Name	Explanation
Lifecycle Management Interfaces	**CApexFtdcTraderApi ::CreateFtdcTraderApi**	Create a **TraderApi** instance
	CApexFtdcTraderApi ::GetVersion	Get API version
	CApexFtdcTraderApi ::Release	Delete the instance of interface
	CApexFtdcTraderApi ::Init	Initialization
	CApexFtdcTraderApi ::Join	Wait for Interface thread to terminate
Parameter Management Interfaces	**CApexFtdcTraderApi ::RegisterSpi**	Register callback interface
	CApexFtdcTraderApi ::RegisterFront	Register the network address of gateway
	CApexFtdcTraderApi ::RegisterNameServer	Register the network address of **NameServer**
	CApexFtdcTraderApi ::RegisterCertificateFile	Load certificate
	CApexFtdcTraderApi ::SetHeartbeatTimeout	Set the timeout for heartbeat
Subscription Interfaces	**CApexFtdcTraderApi ::SubscribePrivateTopic**	Subscribe to private stream
	CApexFtdcTraderApi ::SubscribePublicTopic	Subscribe to public stream
	CApexFtdcTraderApi ::SubscribeUserTopic	Subscribe to trader's stream
Audit Log Interfaces	**CApexFtdcTraderApi ::OpenRequestLog**	Open the request log file
	CApexFtdcTraderApi ::OpenResponseLog	Open the reply log file
Communication Status Interfaces	**CApexFtdcTraderSpi ::OnFrontConnected**	The method is called when communication connection with the Trading System (not logged in yet) is established.
	CApexFtdcTraderSpi ::OnFrontDisconnected	This method is called when communication with the Trading System is disconnected.
	CApexFtdcTraderSpi ::OnHeartBeatWarning	This method is called if no heartbeat message is received after a long time.
	CApexFtdcTraderSpi ::OnPackageStart	Notification for start of message callback
	CApexFtdcTraderSpi ::OnPackageEnd	Notification for end of the message callback
Disaster Recovery Interfaces	**CApexFtdcTraderSpi ::OnRtnFlowMessageCancel**	Notification for data stream cancellation

5.2. Service Interfaces

Service Type	Service	Request Interface / Response Interface	Data Stream
Login/logout	Login	**CApexFtdcTraderApi :: ReqUserLogin** **CApexFtdcTraderSpi :: OnRspUserLogin**	N / A
	Logout	**CApexFtdcTraderApi :: ReqUserLogout** **CApexFtdcTraderSpi :: OnRspUserLogout**	Dialog Stream
	User Password Update	**CApexFtdcTraderApi ::ReqUserPasswordUpdate** **CApexFtdcTraderSpi ::OnRspUserPasswordUpdate**	Dialog Stream
Subscription	Topic/Theme/Subject Subscription	**CApexFtdcTraderApi :: ReqSubscribeTopic** **CApexFtdcTraderSpi :: OnRspSubscribeTopic**	Dialog Stream
	Topic/Theme/Subject Query	**CApexFtdcMduserApi :: ReqQryTopic** **CApexFtdcMduserSpi :: OnRspQryTopic**	Query Stream
Trading	Order Entry	**CApexFtdcTraderApi :: ReqOrderInsert** **CApexFtdcTraderSpi :: OnRspOrderInsert**	Dialog Stream
	Order Action	**CApexFtdcTraderApi :: ReqOrderAction** **CApexFtdcTraderSpi :: OnRspOrderAction**	Dialog Stream
	Combination/Portfolio Order Entry	**CApexFtdcTraderApi :: ReqCombOrderInsert** **CApexFtdcTraderSpi :: OnRspCombOrderInsert**	Dialog Stream
	Price Quotation Entry	**CApexFtdcTraderApi :: ReqQuoteInsert** **CApexFtdcTraderSpi :: OnRspQuoteInsert**	Dialog Stream
	Price Quotation Action	**CApexFtdcTraderApi :: ReqQuoteAction** **CApexFtdcTraderSpi :: OnRspQuoteAction**	Dialog Stream
	Declaration entry	**CApexFtdcTraderApi :: ReqExecOrderInsert** **CApexFtdcTraderSpi :: OnRspExecOrderInsert**	Dialog Stream
	Declaration Action	**CApexFtdcTraderApi :: ReqExecOrderAction** **CApexFtdcTraderSpi :: OnRspExecOrderAction**	Dialog Stream
Private Return	Trade Return	**CApexFtdcTraderSpi :: OnRtnTrade**	Private Stream
	Order Return	**CApexFtdcTraderSpi :: OnRtnOrder**	Private Stream
	Combination/Portfolio Order Return	**CApexFtdcTraderSpi :: OnRtnCombOrder**	Private Stream
	Price Quotation Return	**CApexFtdcTraderSpi :: OnRtnQuote**	Private Stream
	Order Execution Return	**CApexFtdcTraderSpi :: OnRtnExecOrder**	Private Stream
	Order Entry Error Return	**CApexFtdcTraderSpi :: OnErrRtnOrderInsert**	Private Stream
	Order Action Error Return	**CApexFtdcTraderSpi :: OnErrRtnOrderAction**	Private Stream

Service Type	Service	Request Interface / Response Interface	Data Stream
	Combination/Portfolio Order Entry Error Return	**CApexFtdcTraderSpi ::OnErrRtnCombOrderInsert**	Private Stream
	Price Quotation Entry Error Return	**CApexFtdcTraderSpi :: OnErrRtnQuoteInsert**	Private Stream
	Price Quotation Action Error Return	**CApexFtdcTraderSpi :: OnErrRtnQuoteAction**	Private Stream
	Declaration Entry Error Return	**CApexFtdcTraderSpi :: OnErrRtnExecOrderInsert**	Private Stream
	Declaration Action Error Return	**CApexFtdcTraderSpi ::OnErrRtnExecOrderAction**	Private Stream
Public Notification	Contract/Instrument Trading Status Notification	**CApexFtdcTraderSpi :: OnRtnInstrumentStatus**	Public Stream
	Instrument Addition Notification	**CApexFtdcTraderSpi :: OnRtnInsInstrument**	Public Stream
	Instrument Deletion Notification	**CApexFtdcTraderSpi :: OnRtnDelInstrument**	Public Stream
	Combination Leg Entry Notification	**CApexFtdcTraderSpi :: OnRtnInsCombinationLeg**	Public Stream
	Combination Leg Deletion Notification	**CApexFtdcTraderSpi :: OnRtnDelCombinationLeg**	Public Stream
	Alias Definition Notification	**CApexFtdcTraderSpi :: OnRtnAliasDefine**	Public Stream
	Bulletin Notification	**CApexFtdcTraderSpi :: OnRtnBulletin**	Public Stream
Query	Member Cash Query	**CApexFtdcTraderApi :: ReqQryPartAccount** **CApexFtdcTraderSpi :: OnRspQryPartAccount**	Query Stream
	Order Query	**CApexFtdcTraderApi :: ReqQryOrder** **CApexFtdcTraderSpi :: OnRspQryOrder**	Query Stream
	Combination/Portfolio Order Query	**CApexFtdcTraderApi :: ReqQryCombOrder** **CApexFtdcTraderSpi :: OnRspQryCombOrder**	Query Stream
	Price Quotation Query	**CApexFtdcTraderApi :: ReqQryQuote** **CApexFtdcTraderSpi :: OnRspQryQuote**	Query Stream
	Trade Query (i.e.filled/matched order)	**CApexFtdcTraderApi :: ReqQryTrade** **CApexFtdcTraderSpi :: OnRspQryTrade**	Query Stream

Service Type	Service	Request Interface / Response Interface	Data Stream
	Client Query	**CApexFtdcTraderApi :: ReqQryClient** **CApexFtdcTraderSpi :: OnRspQryClient**	Query Stream
	Member Holding Position Query	**CApexFtdcTraderApi :: ReqQryPartPosition** **CApexFtdcTraderSpi :: OnRspQryPartPosition**	Query Stream
	Client Holding Position Query	**CApexFtdcTraderApi :: ReqQryClientPosition** **CApexFtdcTraderSpi :: OnRspQryClientPosition**	Query Stream
	Instrument/Contract Query	**CApexFtdcTraderApi :: ReqQryInstrument** **CApexFtdcTraderSpi :: OnRspQryInstrument**	Query Stream
	Instrument/Contract Trading Status Que	**CApexFtdcTraderApi :: ReqQryInstrumentStatus** **CApexFtdcTraderSpi ::OnRspQryInstrumentStatus**	Query Stream
	Hedge Volume Query	**CApexFtdcTraderApi :: ReqQryHedgeVolume** **CApexFtdcTraderSpi :: OnRspQryHedgeVolume**	Query Stream
	Market Data Query	**CApexFtdcTraderApi :: ReqQryMarketData** **CApexFtdcTraderSpi :: OnRspQryMarketData**	Query Stream
	Bulletin Query	**CApexFtdcTraderApi :: ReqQryBulletin** **CApexFtdcTraderSpi :: OnRspQryBulletin**	Query Stream
	Instrument Price Level Query	**CApexFtdcTraderApi :: ReqQryMBLMarketData** **CApexFtdcTraderSpi ::OnRspQryMBLMarketData**	Query Stream

5.3. Services Not Open To Public in Current Version

Service Type	Service	Request Interface / Response Interface	Opening Status
Trading	Order Entry	**CApexFtdcTraderApi ::ReqOrderInsert** **CApexFtdcTraderSpi ::OnRspOrderInsert**	Partially open
	Order Action	**CApexFtdcTraderApi ::ReqOrderAction** **CApexFtdcTraderSpi ::OnRspOrderAction**	Partially open
	Combination/Portfolio Order Entry	**CApexFtdcTraderApi ::ReqCombOrderInsert** **CApexFtdcTraderSpi ::OnRspCombOrderInsert**	Not open
	Price Quotation Entry	**CApexFtdcTraderApi ::ReqQuoteInsert** **CApexFtdcTraderSpi ::OnRspQuoteInsert**	Not open
	Price Quotation Action	**CApexFtdcTraderApi ::ReqQuoteAction** **CApexFtdcTraderSpi ::OnRspQuoteAction**	Not open
	Execution declaration entry	**CApexFtdcTraderApi ::ReqExecOrderInsert** **CApexFtdcTraderSpi ::OnRspExecOrderInsert**	Not open
	Execution declaration Action	**CApexFtdcTraderApi ::ReqExecOrderAction** **CApexFtdcTraderSpi ::OnRspExecOrderAction**	Not open
Return	Combination/Portfolio Order Return	**CApexFtdcTraderSpi ::OnRtnCombOrder**	Not open
	Price Quotation Return	**CApexFtdcTraderSpi ::OnRtnQuote**	Not open
	Order Execution Return	**CApexFtdcTraderSpi ::OnRtnExecOrder**	Not open

Service Type	Service	Request Interface / Response Interface	Opening Status
	Combination/Portfolio Order Entry Error Return	**CApexFtdcTraderSpi ::OnErrRtnCombOrderInsert**	Not open
	Price Quotation Entry Error Return	**CApexFtdcTraderSpi ::OnErrRtnQuoteInsert**	Not open
	Price Quotation Action Error Return	**CApexFtdcTraderSpi ::OnErrRtnQuoteAction**	Not open
	Execution declaration entry error return	**CApexFtdcTraderSpi ::OnErrRtnExecOrderInsert**	Not open
	Execution declaration action error return	**CApexFtdcTraderSpi ::OnErrRtnExecOrderAction**	Not open
Public Notification	Combination Leg Entry Notification	**CApexFtdcTraderSpi ::OnRtnInsCombinationLeg**	Not open
	Combination Leg Deletion Notification	**CApexFtdcTraderSpi ::OnRtnDelCombinationLeg**	Not open
Inquiry	Combination Order Query	**CApexFtdcTraderApi ::ReqQryCombOrder** **CApexFtdcTraderSpi ::OnRspQryCombOrder**	Not open

6. TraderAPI Reference Manual

6.1. CApexFtdcTraderSpi Interface

CApexFtdcTraderSpi implements event notification interface. Member System has to derive the **CApexFtdcTraderSpi** interface and provide event-handling methods to deal with the events of interest.

6.1.1 OnFrontConnected Method

After the TCP virtual link path connection between Member System and the gateway of the APEX Trading System is established, the method is called.

Function Prototype:

```
void OnFrontConnected();
```

Note: The fact that **OnFrontConnected** is called only indicates that TCP connection is successful. Member System must login to the Trading System to carry out any business operation afterwards. Login failure will not callback this method.

6.1.2 OnFrontDisconnected Method

After the TCP virtual link path connection between Member System and the gateway of the APEX Trading System is broken, the method is called. In this case, API will automatically reconnect, and Member System does not need to deal with the reconnection. The automatically reconnected address may be the originally registered address or other available communication addresses that are supported by the system, which is chosen by the API.

Function Prototype:

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

Parameter: nReason: disconnection reason

- 0x1001 network reading failure
- 0x1002 network writing failure
- 0x2001 heartbeat receiving timeout
- 0x2002 heartbeat sending timeout
- 0x2003 error message received

6.1.3 OnHeartBeatWarning Method

The method is called if heartbeat message is not received after a long time. Default timeout warning period is 5 seconds. If the **SetHeartbeatTimeout(unsigned int timeout)** method is called, heartbeat timeout period can be reset, in which case, the warning time is set to be timeout/2.

Function Prototype:

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

Parameter:

nTimeLapse: time elapsed since the last time receiving the message (in seconds)

6.1.4 OnPackageStart Method

This method indicates the start of message/packets callback. After the API receives message/packet, it first calls this method, followed by the callback of the various data fields and then it calls OnPackageEnd to indicate the end of message callback.

Function Prototype:

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

Parameter:

nTopicID: Topic ID (e.g. private stream, public stream, market data stream etc.)
nSequenceNo: Message Sequence Number

6.1.5 OnPackageEnd Method

This method indicates the end of message/packets callback. After the API receives a message/packet, it first calls OnPackageStart to indicate the start of message/packet callback, followed by the callback of the various data fields and then it calls this method.

Function Prototype:

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

Parameters:

nTopicID: Topic ID(e.g. private stream, public stream, market data stream etc.)
nSequenceNo: Message Sequence Number

6.1.6 OnRspUserLogin Method

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

Function Prototype:

```
void OnRspUserLogin(
        CApexFtdcRspUserLoginField *pRspUserLogin,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pRspUserLogin: returns the address for user login information/message structure. The structure:

```
struct CApexFtdcRspUserLoginField {
    ///trading day
    TApexFtdcDateType TradingDay;
    ///successful login time
    TApexFtdcTimeType LoginTime;
    ///Maximum order local ID
    TApexFtdcOrderLocalIDType  MaxOrderLocalID;
    ///Trading User ID
    TApexFtdcUserIDType UserID;
    ///Exchange Member ID
    TApexFtdcParticipantIDType ParticipantID;
    ///Trading System Name
    TApexFtdcTradingSystemNameType TradingSystemName;
    ///Data Center ID
    TApexFtdcDataCenterIDType  DataCenterID;
    ///current length of the Member's private stream
    TApexFtdcSequenceNoType PrivateFlowSize;
    /// Trader-specific private stream current length
    TApexFtdcSequenceNoType UserFlowSize;
};
```

Note: if Member System maintains its own retransmission sequence number, it should save the returned TradingDay and DataCenterID, so that these can be filled in the login request upon next login.

pRspInfo: returns the address for user response information/message. **Special attention**: **When there are continuous successful response data, some returned value in between may be NULL, but the 1st returned value will never be NULL. This is the same below.** Error ID 0 means successful operation. This is the same below. Response information/message structure is:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
3	Participant not found	ParticipantID is wrong when logging in
45	Invalid data group datasync status in initialization	Trading System initialization is not completed, may try later in 30 seconds or 1 minute
106	Duplicated session	The trading user has logged in already
60	Invalid user or password	User ID or password is wrong
62	User not active	Trading System locked the trader's account
64	User does not belong to this participant	ParticipantID is wrong
65	Invalid login IP address	The computer used to login does not have the IP address allowed by APEX
100	Invalid user type	Non-trading user tries to log in to the Trading System

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

6.1.7 OnRspUserLogout Method

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

Function Prototype:

```
void OnRspUserLogout(
    CApexFtdcRspUserLogoutField *pRspUserLogout,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pRspUserLogout: returns the address for user logout message. User logout message structure:

```
struct CApexFtdcRspUserLogoutField {
    ///User ID
    TApexFtdcUserIDType UserID;
    ///Memebr ID
    TApexFtdcParticipantIDType ParticipantID;
};
```

pRspInfo: returns the address for user response information. Response information structure:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
1	Not login	User has not logged in yet
67	Not logged in by this user	User logging out is not the same as the one logged in
68	Not logged in by this participant	Participant logging out is not the same as the one logged in

nRequestID: returns 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

6.1.8 OnRspUserPasswordUpdate Method

After Member System sends out password update request, API calls this method to send back the response.

Function Prototype:

```
void OnRspUserPasswordUpdate(
    CApexFtdcUserPasswordUpdateField *pUserPasswordUpdate,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pUserPasswordUpdate: pointer to the address for user password update structure, including the input data for user password change request. The user password update structure is:

```
struct CApexFtdcUserPasswordUpdateField {
```

```
        ///Trading User ID
        TApexFtdcUserIDType UserID;
        ///Member ID
        TApexFtdcParticipantIDType ParticipantID;
        ///Old password
        TApexFtdcPasswordType   OldPassword;
        ///New password
        TApexFtdcPasswordType   NewPassword;
    };
```

pRspInfo: pointer to the address for response information structure. Response information structure:

```
    struct CApexFtdcRspInfoField {
        ///ErrorID
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error ID	Error message	Possible reason
58	User mismatch	User requesting for password update is not the same as the user logged in
60	Invalid user or password	Password is wrong
1	Not login	User not log in yet
68	Not logged in by this participant	Participant requesting password update not same as one logged in

nRequestID: returns user password update request ID; this ID is specified upon user password update.

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

6.1.9 OnRspSubscribeTopic Method

After Member System sends out topic subscription instruction, the API calls this method to send back the response.

Function Prototype:

```
    void OnRspSubscribeTopic (
        CApexFtdcDisseminationField *pDissemination,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pDissemination: pointer to the address for subscription topic structure, including topic subscribed and starting message sequence number. Subscription topic structure is:

```
struct CApexFtdcDisseminationField {
    ///sequence series
    TApexFtdcSequenceSeriesType SequenceSeries;
    ///sequence number
    TApexFtdcSequenceNoType SequenceNo;
};
```

pRspInfo: pointer to the address for response information/message structure. Response information structure:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
1	Not login	User not log in yet

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

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

6.1.10 OnRspQryTopic Method

After Member System sends out topic query instruction, the API calls this method to send back the response.

Function Prototype:

```
void  OnRspQryTopic  (
    CApexFtdcDisseminationField *pDissemination,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pDissemination: pointer to the address for topic query structure, including topic queried and number of messages in the topic. Topic query structure is:

```
struct CApexFtdcDisseminationField {
    ///sequence series
    TApexFtdcSequenceSeriesType SequenceSeries;
    ///sequence number
    TApexFtdcSequenceNoType SequenceNo;
```

```
};
```

pRspInfo: points to the address for response information/message structure. The response information/message structure is:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
1	Not login	User not log in yet

nRequestID: returns the topic query request ID; this ID is specified upon sending topic query request.

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

6.1.11 OnRspError Method

This method is called when a request returns an error.

Function Prototype:

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

Parameters:

pRspInfo: returns the address for response information structure. The response information structure is:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: returns the user operating request ID; this ID is specified at the time the request was sent.

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

6.1.12 OnRspOrderInsert Method

After Member System sends out order entry instruction, the API calls this method to send back the response.

Function Prototype:

```
void OnRspOrderInsert(
    CApexFtdcInputOrderField *pInputOrder,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameter:

pInputOrder: pointer to the address of order insert structure, including submitted input data as well as the order ID returned from the Trading System. Note: some fields in the structure are different from the order input request, the return value of Trading System is null.

Order Insert Structure:

```
struct CApexFtdcInputOrderField {
    ///Order System ID; this field is returned from the Trading System
    TApexFtdcOrderSysIDType OrderSysID;
    ///Exchange Member ID, not used[1]
    TApexFtdcParticipantIDType ParticipantID;
    ///Client ID, not used
    TApexFtdcClientIDType  ClientID;
    ///Trading User ID, not used
    TApexFtdcUserIDType UserID;
    ///Contract ID/Instrument ID, not used
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Order price type/condition, not used
    TApexFtdcOrderPriceTypeType OrderPriceType;
    ///buy/sell direction, not used
    TApexFtdcDirectionType Direction;
    ///combination offset flag, not used
    TApexFtdcCombOffsetFlagType CombOffsetFlag;
    ///combination speculation hedge flag, not used
    TApexFtdcCombHedgeFlagType CombHedgeFlag;
    ///Price, not used
    TApexFtdcPriceType LimitPrice;
```

[1] These data fields are kept for compatiblility with the future version of the Trading System, and their contents are meaningless in the current version. Member System should not assume any meaning for those fields. In the underlying communication implementation, TraderAPI uses compression algorithm to lower the communication bandwidth cost, while at the same time maintains the compatibility and extendablility of the protocol and TraderAPI. Similar in the following cases.

```
        ///quantity, not used
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///validity period type, not used
        TApexFtdcTimeConditionType TimeCondition;
        ///GTD date, not used
        TApexFtdcDateType  GTDDate;
        ///match volume type not used
        TApexFtdcVolumeConditionType   VolumeCondition;
        ///minimum volume not used
        TApexFtdcVolumeType MinVolume;
        ///trigger condition, not used
        TApexFtdcContingentConditionType   ContingentCondition;
        ///stop price, not used
        TApexFtdcPriceType  StopPrice;
        ///force close reasons, not used
        TApexFtdcForceCloseReasonType  ForceCloseReason;
        ///local order ID
        TApexFtdcOrderLocalIDType  OrderLocalID;
        ///automatic suspend flag, not used
        TApexFtdcBoolType  IsAutoSuspend;
        ///business unit, not used
        TApexFtdcBusinessUnitType  BusinessUnit;
    };
```

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

```
    struct CApexFtdcRspInfoField {
        ///ErrorID
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error ID	Error message	Possible reason
2	Instrument not found	Unable to find the instrument in the order
3	Participant not found	Unable to find the Participant in the order
4	Client not found	Unable to find the client in the order
6	Bad order field	Certain field in the order is illegal (e.g. enumeration value is out of bound) or non-forced close order with forced close reason
12	Duplicate order	The OrderLocalID sent is alphabetically less than the OrderLocalID of the last order placed or the ActionLocalID of the last order action

15	Client does not belong to participant	The client in the order has no account under the specified participant
16	IOC order can only apply to continuous trading	IOC (immediately-or-cancel) order is tried to be entered at non-continuous trading session
17	GFA order can only apply to auction trading	GFA order is tried to be entered at non-auction session
18	Market order cannot queue	The time condition of market order is not IOC
19	Volume constrain can only apply to IOC order	The order whose volume restriction is not arbitrary does not have the IOC time condition
20	GTD order expired	The GTD date in the GTD order is expired
21	Order volume smaller than minimum quantitiy	The order has minimum volume condition, but the order volume is less than this minimum volume
22	Exchange not in sync	The Trading System is not completely initialized, try later
23	Settlement group not in sync	Initialization of the Trading System is incomplete, try later
26	Invalid action in current status	The trading status of the instrument is not continuous-trading or auction or auction balance
31	Not enough client position to close	Client holding position is not enough while entering close order
32	Exceeds client position limit	When entering open position order, the client's speculation limit position is exceeded
34	Exceeds participant position limit	When entering open position order, the member's limit position is exceeded
35	Account not found	Unable to find the cash account used in the order
36	Insufficient credit	There is not enough cash in the cash account
37	Invalid volume	Order volume is not an integer multiple of the minimum volume, or exceeds the maximum order volume
48	Price must be integral multiple of tick	Order price is not an integer multiple of the minimum variable price unit
49	Price out of upper bound	Order price exceed the upper limit of the instrument
50	pPrice out of lower bound	Order price lower than the lower limit of the instrument
51	No trading right	Member, client or trader no rights to trade specified instrument/contract
52	Close only	Member, client or trader only have rights to close position
53	Invalid trading role	Member has no trading role with the client in the specified order
57	Cannot operate for other participant	Trader trying to operate for other participants that he is not working for
58	User mismatch	Trader in the order and trader upon login not match

1	Not login	User not logged in yet
78	GTD order date missing	GTD order does not specify the GTD date
79	Unsupported order type	APEX does not support this type of order
83	Stop order can only apply to continuous trading	Stop loss order is entered in non-continuous trading session
84	Stop order must be IOC or GFD	Time condition is neither IOC nor GFD at stop loss order
95	Stop order must have stop price	The stop loss order does not specify stop price
96	Not enough hedge volume	When entering hedging order, client hedge amount is not enough
103	Cannot close today's position for hedge	Hedging position should not use close-today-position order to close the position
114	Best price order cannot queue	Best price order time condition is not IOC

nRequestID: returns order insert operating request ID; this ID is specified by user upon Order Entry.

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

Note:

CApexFtdcRspInfoField.ErrorID is 0 implies that current order entry is successful. In **ApexFtdcInputOrderField *pInputOrder**, only OrderSysID (the system ID given by the Trading System) and OrderLocalID 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 **CApexFtdcInputOrderField.**

6.1.13 OnRspOrderAction Method

After the Member System sends an order operation (cancellation, suspension, activation and modification) request and the Trading System returns a response, this method is called.

Function prototype:

```
void OnRspOrderAction(
    CApexFtdcOrderActionField *pOrderAction,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pOrderAction: Address pointing to order operation structure, including the submitted input data. Note: some fields in the structure are different from the order operation request, the return value of Trading System is null. Order operation structure:

```
struct CApexFtdcOrderActionField {
    /// Order No.
    TApexFtdcOrderSysIDType OrderSysID;
    /// Local Order No.
    TApexFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of Order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member's code, not used
    TApexFtdcParticipantIDType ParticipantID;
    ///Client's code, not used
    TApexFtdcClientIDType  ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Price, not used
    TApexFtdcPriceType  LimitPrice;
    /// Change in quantity, not used
    TApexFtdcVolumeType VolumeChange;
    /// Operation of local No.
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit, not used
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
3	Participant not found	Participant cannot be found in the order operation
4	Client not found	Client cannot be found in the order operation
8	Bad order action field	Illegal field values in the order operation (out-of-range of the enumerated value).
15	Client does not belong to participant	Client didn't open an account at the designated participant
22.	Exchange not in sync	Initialization of trading system is not completed, please try later.

23	Settlement group not in sync	Initialization of trading system is not completed, please try later.
24.	Order not found	Order to be operated cannot be found
26.	Invalid action in current status	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	Order fully traded	Order has already been fulfilled
29	Order already cancelled	Order has already been cancelled
32	Exceeds client position limit	Exceeding the client's speculative position limit when modifying the order
34	Exceeds participant position limit	Exceeding the member's position limit when modifying the order
35.	Account not found	The capital account shall be used cannot be found
36	Insufficient balance	No sufficient funds in capital account
37.	Invalid volume	The number of order is not the positive integral multiple as required the Min. number of order or exceeds the Max. number of order
48	Price must be integral multiple of tick	Price of order after modification is not the integral multiple of the contract's tick size
49.	Price out of upper bound	Price of order after modification is higher than the contract's upward price limit
50	Price out of lower bound	Price of order after modification is lower than the contract's downward price limit
57	Cannot operate for other participant	Trader conducts operation on behalf of participant to whom he is not subordinate.
58	User mismatch	Trader in the order operation doesn't match with trader at the time of login
1	Not login	User hasn't logged in yet
76	Order suspended	Order has already been suspended when order is suspended.
77	Order activated	Order has already been activated when order is activated.
96	Not enough hedge volume	The client's hedge quota is insufficient when modifying the order
97	Duplicated action	The ActionLocalID sent is alphabetically less than the OrderLocalID of the last order placed or the ActionLocalID of the last order action.
99	Cannot action for other user	Unauthorized trader operates order submitted by other traders of the same member

nRequestID: ID for return to request for user's order operation. This ID will be designated at the time of order operation.

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

6.1.14 OnRspQuoteInsert Method

Not available in the current version.

This method is used to response to quote entry. When member system gave the instructions for entry of order and trading system returned a response, this method will be called.

Function prototype:

```
void OnRspQuoteInsert(
    CApexFtdcInputQuoteField *pInputQuote,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pInputQuote: Address pointing to the input quote sturcture, including the input data of quote entry operation and the quote No. returned from trading system. The input quote structure:

```
struct CApexFtdcInputQuoteField {
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    /// Transaction user's code
    TApexFtdcUserIDType UserID;
    /// Bid Volume
    TApexFtdcVolumeType BidVolume;
    /// Ask Volume
    TApexFtdcVolumeType AskVolume;
    ///Contact code
    TApexFtdcInstrumentIDType  InstrumentID;
    /// Local quote No.
    TApexFtdcQuoteLocalIDType  QuoteLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Flag of position opening and closing-out in buyer's portfolio
    TApexFtdcCombOffsetFlagType BidCombOffsetFlag;
    ///Flag of hedge in buyer's portfolio
    TApexFtdcCombHedgeFlagType BidCombHedgeFlag;
    ///Buyer's price
    TApexFtdcPriceType  BidPrice;
    ///Flag of position opening and closing-out in seller's portfolio
    TApexFtdcCombOffsetFlagType AskCombOffsetFlag;
    ///Flag of hedge in seller's portfolio
    TApexFtdcCombHedgeFlagType AskCombHedgeFlag;
    ///Seller's price
    TApexFtdcPriceType  AskPrice;
```

```
};
```

pRspInfo: Address pointing to the response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error ID	Error message	Possible reason
2	Instrument not found	Contract cannot be found in the quote.
3	Participant not found	Participant cannot be found in the quote
4	Client not found	Client cannot be found in the quote
7	Bad quote field	Illegal field values in the quote (out-of-range of the enumerated value).
13	Duplicate quote	duplicate local quote No. in the quote
15	Client does not belong to participant	Client in the quote didn't open an account at the designated member
22.	Exchange not in sync	Initialization of trading system is not completed, please try later.
23	Settlement group not in sync	Initialization of trading system is not completed, please try later.
26.	Invalid action in current status	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
31.	Not enough client position to close	The client's open interest is insufficient
32	Exceeds client position limit	This quote caused the client's speculative position exceeding position limit
34	Exceeds participant position limit	This quote caused the member's open interest exceeding position limit
35.	Account not found	The capital account used for quotation cannot be found
36	Insufficient balance	No sufficient funds in capital account
37.	Invalid volume	The number of order is not the positive integral multiple as required by the Min. number of order or exceeds the Max. number of order
48	Price must be integral multiple of tick	The quoted price is not the integral multiple of the contract's tick size
49.	Price out of upper bound	The quoted price is higher than the contract's upward price limit
50	Price out of lower bound	The quoted price is lower than the contract's downward price limit
51	No trading right	Not authorized to trade in the designated contract, or client or trader is not authorized to trade in the designated contract
52	Close only	As for the designated contract, member, client or trader is authorized to close out position only.

53.	Invalid trading role	On the designated contract, member doesn't has the trading role corresponding to such client
57	Cannot operate for other participant	Trader conducts operation on behalf of member to whom he is not subordinate.
58	User mismatch	Trader in the quote doesn't match with trader at the time of login
1	Not login	User hasn't logged in yet
79	Unsupported order type	The Exchange does not support this order type.
96	Not enough hedge volume	The client's hedge quota is insufficient when submitting the hedging quota
103.	Cannot close today's position for hedge	The hedge positions cannot be closed out using the quote for closing out position on that day

nRequestID: ID for return to user's request for quote entry operation. This ID will be designated at the time of quote entry.

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

6.1.15 OnRspQuoteAction Method

Not available in the current version.

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 gave the instuctions for quote operation and trading system returned a response, this method will be called.

Function prototype:

```
void OnRspQuoteAction(
    CApexFtdcQuoteActionField *pQuoteAction,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pQuoteAction: Address pointing to quote operation structure, including the input data of request for quote operation and quote No. returned from trading system. Quote operation structure:

```
struct CApexFtdcQuoteActionField {
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Local quote No.
    TApexFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
```

```
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Local No. of operation
        TApexFtdcOrderLocalIDType   ActionLocalID;
        ///Business unit
        TApexFtdcBusinessUnitType   BusinessUnit;

    };
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
    struct CApexFtdcRspInfoField {
        ///ErrorID
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error ID	Error message	Possible reason
3	Participant not found	Participant cannot be found in the quote operation
4	Client not found	Client cannot be found in the quote operation
9	Bad quote action field	Illegal field values in the quote operation (out-of-range of the enumerated value).
15	Client does not belong to participant	Client didn't open an account at the designated member
22.	Exchange not in sync	Initialization of trading system is not completed, please try later.
23	Settlement group not in sync	Initialization of trading system is not completed, please try later.
25.	Quote not found	Quote to be operated cannot be found
26.	Invalid action in current status	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 fully traded	Order derived from quote has already been fulfilled
35	Account not found	The capital account shall be used cannot be found
36	Insufficient balance	No sufficient funds in capital account
57	Cannot operate for other participant	Trader conducts operation on behalf of member to whom he is not subordinate.
58	User mismatch	Trader in the quote operation doesn't match with trader at the time of login
1	Not login	User hasn't logged in yet
70	Quote cancelled	Quote has already been cancelled
97	Duplicated action	Local operation No. in the quote operation is not unique.
99	Cannot action for other user	Unauthorized trader operates the quote submitted by other traders of the same member

nRequestID: ID for return to user's request for quote operation. This ID will be designated by user at the time of quote operation

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

6.1.16 OnRspExecOrderInsert Method

Not available in the current version.

This method is used to response to execution declaration entry. When member system executed the entry of declaration and trading system returned a response, this method will be called.

Function prototype:

```
void OnRspExecOrderInsert(
    CApexFtdcInputExecOrderField *pInputExecOrder,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID, bool bIsLast);
```

Parameters:

pInputExecOrder: Address pointing to the declaration entry structure. Structure of execution declaration entry:

```
struct CApexFtdcInputExecOrderField {
    /// Contract No.
    TApexFtdcInstrumentIDType  InstrumentID;
    /// Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    /// Local execution declaration No.
    TApexFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Quantity
    TApexFtdcVolumeType Volume;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    ///ErrorID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
```

```
    TApexFtdcErrorMsgType  ErrorMsg;
};
```

Error ID	Error message	Possible reason
2	Instrument not found	Contract cannot be found in the execution declaration.
3	Participant not found	Member cannot be found in the execution declaration
4	Client not found	Client cannot be found in the execution declaration
15	Client does not belong to participant	Client in the in the execution declaration didn't open an account at the designated member
22	Exchange not in sync	Initialization of trading system is not completed, please try later.
23	Settlement group not in sync	Initialization of trading system is not completed, please try later.
26	Invalid action in current status	Tthe contract's trading status is in the closing state
51	No trading right	Not authorized to trade in the designated contract, or client or trader is not authorized to trade in the designated contract
52	Close only	As for the designated contract, member, client or trader is authorized to close out position only.
53	Invalid trading role	On the designated contract, member doesn't has the trading role corresponding to such client
57	Cannot operate for other participant	Trader conducts operation on behalf of member to whom he is not subordinate.
58	User mismatch	Trader in the execution declaration doesn't match with trader at the time of login
66	Not login	User hasn't logged in yet
79	Unsupported order type	The Exchange does not support this order type.
89	Bad ExecOrder field	Illegal field values in the execution of declaration operation (out-of-range of the enumerated value).
91	Duplicated ExecOrder	The local announcment execution No. in execution declaration is not unique.
94	ExecOrder only for options	The contract in execution declaration is non-option contract

nRequestID: ID for return to request for execution declaration entry. This ID will be designated by user at the time of execution declaration entry.

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

6.1.17 OnRspExecOrderAction Method

Not available in the current version.

Response to execution of annoncement operation. When member system executed the declaration operation and trading system returned a response, this method will be called.

Function prototype:

```
void OnRspExecOrderAction(
```

```
        CApexFtdcExecOrderActionField *pExecOrderAction,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pInputExecAction: Address pointing to declaration operation structure. Declaration operation structure:

```
    struct CApexFtdcExecOrderActionField {
        ///Execution declaration No.
        TApexFtdcExecOrderSysIDType ExecOrderSysID;
        ///Local annoncement execution No.
        TApexFtdcOrderLocalIDType  ExecOrderLocalID;
        ///Flag of order operation
        TApexFtdcActionFlagType ActionFlag;
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
        TApexFtdcClientIDType ClientID;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Operation of local No.
        TApexFtdcOrderLocalIDType  ActionLocalID;
        ///Business unit
        TApexFtdcBusinessUnitType  BusinessUnit;

    };
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
    struct CApexFtdcRspInfoField {
        ///  ErrorID
        TApexFtdcErrorIDType    ErrorID;
        /// Error Message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error code	Error message	Possible reasons
2	Instrument not found	Contract cannot be found in the execution declaration.
3	Participant not found	Member cannot be found in the execution declaration
4	Client not found	Client cannot be found in the execution declaration
15	Client does not belong to participant	Client in the in the execution declaration didn't open an account at the designated member
22.	Exchange not in sync	Initialization of trading system is not completed, please try later.

23	Settlement group not in sync	Initialization of trading system is not completed, please try later.
26.	Invalid action in current status	Tthe contract's trading status is in the closing state
51	No trading right	Not authorized to trade in the designated contract, or client or trader is not authorized to trade in the designated contract
53.	Invalid trading role	On the designated contract, member doesn't has the trading role corresponding to such client
57	Cannot operate for other participant	Trader conducts operation on behalf of member to whom he is not subordinate.
58	User mismatch	Trader in the execution declaration doesn't match with trader at the time of login
66	Not login	User hasn't logged in yet
79	Unsupported order type	The Exchange does not support this order type.
90	Bad ExecOrder action field	Illegal field values in the execution of declaration operation (out-of-range of the enumerated value).
92	ExecOrder has cancelled	The declaration operation to be executed has been cancelled.
93	ExecOrder not found	The declaration operation to be executed cann not be found
97	Duplicated action	The local operation No. of the execution of declaration operation is not unique.

nRequestID: ID for return to request for exectution of declaration operation. This ID will be designated by user at the time of execution of declaration operation.

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

6.1.18 OnRspQryPartAccount Method

After the Member System requests to query for member's funds and trading system returned a response, this method is called.

Function prototype:

```
void OnRspQryPartAccount(
    CApexFtdcRspPartAccountField *pRspPartAccount,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pRspPartAccount: Address pointing to structure of response to member's funds. Structure of response to member's funds:

```
struct CApexFtdcRspPartAccountField {
    /// Business day
    TApexFtdcDateType TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Reserve funds for previous settlement
```

```
        TApexFtdcMoneyType  PreBalance;
        ///Total margin at present
        TApexFtdcMoneyType  CurrMargin;
        ///Profit & loss on closing-out of position
        TApexFtdcMoneyType  CloseProfit;
        ///Income and expense from option premium
        TApexFtdcMoneyType  Premium;
        ///Deposit amount
        TApexFtdcMoneyType  Deposit;
        ///Withdrawal amount
        TApexFtdcMoneyType  Withdraw;
        /// Reserve funds for futures settlement
        TApexFtdcMoneyType  Balance;
        ///Withdrawable funds
        TApexFtdcMoneyType  Available;
        /// Capital account
        TApexFtdcAccountIDType  AccountID;
        ///Frozen margin
        TApexFtdcMoneyType  FrozenMargin;
        ///Frozen premium
        TApexFtdcMoneyType  FrozenPremium;
        ///Basic reserve funds
        TApexFtdcMoneyType  BaseReserve;
    };
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
    struct CApexFtdcRspInfoField {
        ///  ErrorID
        TApexFtdcErrorIDType    ErrorID;
        /// Error Message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns 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.

6.1.19 OnRspQryOrder Method

After Member System sends out order query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryOrder(
    CApexFtdcOrderField *pOrder,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pOrder: points to the address for order information/message structure. The structure:

```
struct CApexFtdcOrderField {
    ///Trading Date
    TApexFtdcDateType  TradingDay;
    ///Settlement Group ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement ID
    TApexFtdcSettlementIDType  SettlementID;
    ///Order  ID
    TApexFtdcOrderSysIDType OrderSysID;
    ///Member ID
    TApexFtdcParticipantIDType ParticipantID;
    ///Client ID
    TApexFtdcClientIDType ClientID;
    ///Trading User ID
    TApexFtdcUserIDType UserID;
    ///Instrument/contract ID
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Order Price Type
    TApexFtdcOrderPriceTypeType OrderPriceType;
    ///buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Combo open-close position flag
    TApexFtdcCombOffsetFlagType CombOffsetFlag;
    ///Combo speculative hedge flag
    TApexFtdcCombHedgeFlagType CombHedgeFlag;
    ///Price
    TApexFtdcPriceType  LimitPrice;
    ///Volume
    TApexFtdcVolumeType VolumeTotalOriginal;
    ///Expiry Type
    TApexFtdcTimeConditionType TimeCondition;
    ///GTD Date, NOT USED
    TApexFtdcDateType  GTDDate;
```

```
///Match volume condition type
TApexFtdcVolumeConditionType    VolumeCondition;
///Minimum Volume
TApexFtdcVolumeType MinVolume;
///Trigger/Contingent Condition
TApexFtdcContingentConditionType    ContingentCondition;
///Stop loss Price, NOT USED
TApexFtdcPriceType  StopPrice;
///Forced close reasons
TApexFtdcForceCloseReasonType  ForceCloseReason;
///Local order ID
TApexFtdcOrderLocalIDType  OrderLocalID;
///Auto Suspend flag
TApexFtdcBoolType  IsAutoSuspend;
///Order  Source
TApexFtdcOrderSourceType    OrderSource;
///Order  Status
TApexFtdcOrderStatusType    OrderStatus;
///Order Type
TApexFtdcOrderTypeType OrderType;
///Today's trade volume
TApexFtdcVolumeType VolumeTraded;
///Remaining volume
TApexFtdcVolumeType VolumeTotal;
///order  date
TApexFtdcDateType  InsertDate;
///Entry  time
TApexFtdcTimeType  InsertTime;
///activation time, NOT USED
TApexFtdcTimeType  ActiveTime;
///Suspension time, NOT USED
TApexFtdcTimeType  SuspendTime;
///Last amendment time
TApexFtdcTimeType  UpdateTime;
///Cancellation time
TApexFtdcTimeType  CancelTime;
///Last modified trading user ID
TApexFtdcUserIDType ActiveUserID;
///Priority, NOT USED
TApexFtdcPriorityType  Priority;
///Sequence number by time order, NOT USED
TApexFtdcTimeSortIDType TimeSortID;
///Settlement member ID, NOT USED
TApexFtdcParticipantIDType ClearingPartID;
```

```
    ///Business unit, NOT USED
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
    ///Insert Milli second
    TApexFtdcMillisecType InsertMillisec;
    ///Update Milli second
    TApexFtdcMillisecType UpdateMillisec;
    ///Cancel Milli second
    TApexFtdcMillisecType CancelMillisec;

};
```

pRspInfo: points to the address for response information/message structure. The structure:

```
struct CApexFtdcRspInfoField {
    ///  ErrorID
    TApexFtdcErrorIDType    ErrorID;
    /// Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns user request ID for order query; 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.

6.1.20 OnRspQryQuote Method

Not available in the current version.

This function is the response to query for quote. When member system gave the instructions to query for quote and trading system returned a response, this method will be called.

Function prototype:

```
void OnRspQryQuote(
    CApexFtdcQuoteField *pQuote,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pQuote: Address pointing to quote message structure. Quote message structure:

```
struct CApexFtdcQuoteField {
    ///Business day
    TApexFtdcDateType TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Bid Volume
    TApexFtdcVolumeType BidVolume;
    ///Ask Volume
    TApexFtdcVolumeType AskVolume;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Local quote No.
    TApexFtdcQuoteLocalIDType  QuoteLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Flag of position opening and closing-out in buyer's portfolio
    TApexFtdcCombOffsetFlagType BidCombOffsetFlag;
    ///Flag of hedge in buyer's portfolio
    TApexFtdcCombHedgeFlagType BidCombHedgeFlag;
    ///Buyer's price
    TApexFtdcPriceType BidPrice;
    ///Flag of position opening and closing-out in seller's portfolio
    TApexFtdcCombOffsetFlagType AskCombOffsetFlag;
    ///Flag of hedge in seller's portfolio
    TApexFtdcCombHedgeFlagType AskCombHedgeFlag;
    ///Seller's price
    TApexFtdcPriceType AskPrice;
    ///Entry  Time
    TApexFtdcTimeType  InsertTime;
    ///Time of cancelation
    TApexFtdcTimeType  CancelTime;
    ///Transaction time
    TApexFtdcTimeType TradeTime;
    ///Buyer's order No.
```

```
    TApexFtdcOrderSysIDType BidOrderSysID;
    ///Seller's order No.
    TApexFtdcOrderSysIDType AskOrderSysID;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;

};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    /// Error ID
    TApexFtdcErrorIDType    ErrorID;
    /// Error Message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this member can be queried.
57	Cannot operate for other participant	The conditions under other members cannot be queried.

nRequestID: User's request ID for quote query. This ID will be designated by user at time of query for quote.

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

6.1.21 OnRspQryTrade Method

After Member System sends out matched order (i.e. trade) query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryTrade(
    CApexFtdcTradeField *pTrade,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

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

```
struct CApexFtdcTradeField {
    ///Trading Date
    TApexFtdcDateType TradingDay;
```

```
        ///Settlement Group ID
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Settlement ID
        TApexFtdcSettlementIDType  SettlementID;
        ///Matched order ID
        TApexFtdcTradeIDType    TradeID;
        ///Buy-Sell direction
        TApexFtdcDirectionType Direction;
        ///Order  ID
        TApexFtdcOrderSysIDType OrderSysID;
        ///Member ID
        TApexFtdcParticipantIDType ParticipantID;
        ///Client ID
        TApexFtdcClientIDType ClientID;
        ///Trading Role
        TApexFtdcTradingRoleType    TradingRole;
        ///Cash Account
        TApexFtdcAccountIDType AccountID;
        ///Instrument/Contract ID
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Open-Close position flag
        TApexFtdcOffsetFlagType OffsetFlag;
        ///Speculative hedge
        TApexFtdcHedgeFlagType HedgeFlag;
        ///Price
        TApexFtdcPriceType  Price;
        ///Volume
        TApexFtdcVolumeType Volume;
        ///Trade time / order matching time
        TApexFtdcTimeType  TradeTime;
        ///Trade Type / order matching type
        TApexFtdcTradeTypeType  TradeType;
        ///Trade Price Source / Order Matching Price Source
        TApexFtdcPriceSourceType    PriceSource;
        ///Trading User ID
        TApexFtdcUserIDType UserID;
        ///Local Order ID
        TApexFtdcOrderLocalIDType  OrderLocalID;
        ///Settlement Member ID
        TApexFtdcParticipantIDType ClearingPartID;
        ///Business Unit
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;
```

```
        ///Update milli second
        TApexFtdcMillisecType TradeMillisec;


    };
```

pRspInfo: points to the address for response information/message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        /// Error message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

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

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

6.1.22 OnRspForQuote Method

Not available in the current version.

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

Function Prototype:

```
    void  OnRspForQuote(
        CApexFtdcInputReqForQuoteField *pInputReqForQuote,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pInputReqForQuote: points to the address for quote information/message structure. The structure:

```
struct CApexFtdcInputReqForQuoteField {
      ///Quote ID
      TApexFtdcQuoteSysIDType ReqForQuoteID;
      ///Exchange Member ID
```

```
        TApexFtdcParticipantIDType ParticipantID;
        ///Client name
        TApexFtdcClientIDType ClientID;
        ///Instrument/Contract ID
        TApexFtdcInstrumentIDType  InstrumentID;
        ///TradingDay
        TApexFtdcTradingDayType TradingDay;
        ///Quote Time
        TApexFtdcTimeType  ReqForQuoteTime;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;
    };
```

pRspInfo: points to the address for response information/message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        /// Error message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error code	Error message	Possible reasons
2	Instrument not found	Quote contract does not exist.
26	Invalid action in current status	The trading status of the instrument is not continuous-trading.
57	Cannot operate for other participant	The conditions under other members cannot be quoted.
123	Req for quote client cannot be empty	Customer code should be fill in when sends out quote query instruction.

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

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

6.1.23 OnRspQryClient Method

After Member System sends out client query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
    void OnRspQryClient(
        CApexFtdcRspClientField*pClient,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameter:

pClient: points to the address for client information/message structure. The structure:

```
struct CApexFtdcRspClientField {
    ///Client ID
    TApexFtdcClientIDType ClientID;
    ///Client name
    TApexFtdcPartyNameType ClientName;
    ///ID Type
    TApexFtdcIdCardTypeType IdentifiedCardType;
    ///Original ID
    TApexFtdcIdentifiedCardNoV1TypeUseLess;
    ///Trading Role
    TApexFtdcTradingRoleType   TradingRole;
    ///Client type
    TApexFtdcClientTypeType ClientType;
    ///Active or not flag
    TApexFtdcBoolType IsActive;
    ///Member ID
    TApexFtdcParticipantIDType ParticipantID;
    ///ID Number
    TApexFtdcIdentifiedCardNoType  IdentifiedCardNo;
};
```

pRspInfo: points to the address for the response information/message structure. The structure:

```
struct CApexFtdcRspInfoField {
    /// Error code
    TApexFtdcErrorIDType   ErrorID;
    /// Error message
    TApexFtdcErrorMsgType ErrorMsg;
};
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns user request ID for client query; 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.

6.1.24 OnRspQryPartPosition Method

After Member System sends out member holding position query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryPartPosition(
    CApexFtdcRspPartPositionField *pRspPartPosition,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameter:

pRspPartPosition: points to the address for the member holding position response information/message structure. The structure:

```
struct CApexFtdcRspPartPositionField {
    ///Trading Date
    TApexFtdcDateType  TradingDay;
    ///Settlement Group ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement ID
    TApexFtdcSettlementIDType  SettlementID;
    ///Speculative hedge flag
    TApexFtdcHedgeFlagType HedgeFlag;
    ///Holding position over-under direction
    TApexFtdcPosiDirectionType PosiDirection;
    ///Previous day holding position
    TApexFtdcVolumeType YdPosition;
    ///Current day holding position
    TApexFtdcVolumeType Position;
    ///Long frozen
    TApexFtdcVolumeType LongFrozen;
    ///Short frozen
    TApexFtdcVolumeType ShortFrozen;
    ///Previous day long frozen
    TApexFtdcVolumeType YdLongFrozen;
    ///Previous day short frozen
    TApexFtdcVolumeType YdShortFrozen;
    ///Contract / instrument ID
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Member ID
    TApexFtdcParticipantIDType ParticipantID;
    ///Trading role
    TApexFtdcTradingRoleType   TradingRole;
};
```

pRspInfo: pointer to the address for response information strcture. The structure:

```
struct CApexFtdcRspInfoField {
    /// Error code
    TApexFtdcErrorIDType   ErrorID;
    /// Error message
```

```
        TApexFtdcErrorMsgType  ErrorMsg;
    };
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns user request ID for member holding position query; 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.

6.1.25 OnRspQryClientPosition Method

After Member System sends out client holding position query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryClientPosition(
    CApexFtdcRspClientPositionField *pRspClientPosition,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pRspClientPosition: points to the address for the member holding position response information/message structure. The structure:

```
struct CApexFtdcRspClientPositionField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Flag of speculation and hedge
    TApexFtdcHedgeFlagType  HedgeFlag;
    ///Direction of long and short open interest
    TApexFtdcPosiDirectionType  PosiDirection;
    ///Previous-day's open interest
    TApexFtdcVolumeType YdPosition;
    ///Open interest on that day
    TApexFtdcVolumeType Position;
    ///Long frozen
    TApexFtdcVolumeType LongFrozen;
    ///Short frozen
```

```
        TApexFtdcVolumeType ShortFrozen;
        ///Long frozen of yesterday
        TApexFtdcVolumeType YdLongFrozen;
        ///Short frozen of yesterday
        TApexFtdcVolumeType YdShortFrozen;
        ///Buying volume on that day
        TApexFtdcVolumeType BuyTradeVolume;
        ///Selling volume on that day
        TApexFtdcVolumeType SellTradeVolume;
        ///Cost of carry
        TApexFtdcMoneyType  PositionCost;
        ///Yesterday's cost of carry
        TApexFtdcMoneyType  YdPositionCost;
        ///Margin used
        TApexFtdcMoneyType  UseMargin;
        ///Frozen Margin
        TApexFtdcMoneyType  FrozenMargin;
        ///Margin frozen by the long
        TApexFtdcMoneyType LongFrozenMargin;
        ///Margin frozen by the short
        TApexFtdcMoneyType ShortFrozenMargin;
        ///Frozen premium
        TApexFtdcMoneyType  FrozenPremium;
        ///Contract code
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
        TApexFtdcClientIDType ClientID;
    };
```

pRspInfo: points to the address for the response information/message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        /// Error message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns user request ID for client holding position query; 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.

6.1.26 OnRspQryInstrument Method

After Member System sends out instrument/contract query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryInstrument(
    CApexFtdcRspInstrumentField *pRspInstrument,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pRspInstrument: points to the address for instrument/contract structure. The structure:

```
struct CApexFtdcRspInstrumentField {
///Settlement group's code
TApexFtdcSettlementGroupIDType SettlementGroupID;
///Product code
TApexFtdcProductIDType ProductID;
///Product suite's code
TApexFtdcProductGroupIDType ProductGroupID;
///Basic commodity code
TApexFtdcInstrumentIDType  UnderlyingInstrID;
///Product type
TApexFtdcProductClassType  ProductClass;
///Type of open interest
TApexFtdcPositionTypeType  PositionType;
///Strike price
TApexFtdcPriceType StrikePrice;
///Option type
TApexFtdcOptionsTypeType  OptionsType;
///Contract multiplier
TApexFtdcVolumeMultipleType VolumeMultiple;
///Contract multiplier for basic commodity
TApexFtdcUnderlyingMultipleTypeUnderlyingMultiple;
///Contract code
TApexFtdcInstrumentIDType  InstrumentID;
///Contract name
TApexFtdcInstrumentNameType InstrumentName;
///Delivery year
TApexFtdcYearType  DeliveryYear;
```

```
        ///Delivery month
        TApexFtdcMonthType  DeliveryMonth;
        ///Month in advance
        TApexFtdcAdvanceMonthType  AdvanceMonth;
        ///Is trading right now?
        TApexFtdcBoolType   IsTrading;
        ///Creation date
        TApexFtdcDateType   CreateDate;
        ///Listing day
        TApexFtdcDateType   OpenDate;
        ///Expiring date
        TApexFtdcDateType   ExpireDate;
        ///Date of starting delivery
        TApexFtdcDateType  StartDelivDate;
        ///The last delivery day
        TApexFtdcDateType  EndDelivDate;
        ///Benchmark price for listing
        TApexFtdcPriceType  BasisPrice;
        ///The Max. market order placement volume
        TApexFtdcVolumeType MaxMarketOrderVolume
        ///The Min. market order placement volume
        TApexFtdcVolumeType MinMarketOrderVolume
        ///The Max. limit order placemnt volume
        TApexFtdcVolumeType MaxLimitOrderVolume;
        ///The Min. limit order placement volume
        TApexFtdcVolumeType MinLimitOrderVolume;
        ///Tick size
        TApexFtdcPriceType  PriceTick;
        ///Position opened by natural person during delvery month
        TApexFtdcMonthCountType AllowDelivPersonOpen;
        ///Currency ID
        TFfexFtdcCurrencyIDType CurrencyID;
    };
```

pRspInfo: points to the address for response information/ message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        ///Error ID
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns user request ID for contract/instrument query; 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.

6.1.27 OnRspQryInstrumentStatus Method

After Member System sends out instrument/contract trading status query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryInstrumentStatus(
    CApexFtdcInstrumentStatusField *pInstrumentStatus,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pInstrumentStatus: pointer to the address for instrument/contract trading status structure. The structure:

```
struct CApexFtdcInstrumentStatusField {
    ///Settlement group ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Instrument/contract ID
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Contract/Instrument Trading Status
    TApexFtdcInstrumentStatusType  InstrumentStatus;
    ///Trading Phase/Stage/Segment ID
    TApexFtdcTradingSegmentSNType  TradingSegmentSN;
    ///Time of entering current status
    TApexFtdcTimeType  EnterTime;
    ///Reason for entering current status
    TApexFtdcInstStatusEnterReasonType EnterReason;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
};
```

pRspInfo: points to the address for response information structure. The structure:

```
struct CApexFtdcRspInfoField {
    ///Error ID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: returns user request ID for contract/instrument trading status query; 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.

6.1.28 OnRspQryCombinationLeg Method

After Member System sends out leg instruments query instruction and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryCombinationLeg(
    CApexFtdcRspCombinationLegField *pRspCombinationLeg,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pRspCombinationLeg: pointer to the address for leg instrument structure. The structure:

```
struct CApexFtdcRspCombinationLegField {
    ///Settlement Group ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Combination Instrument ID
    TApexFtdcInstrumentIDType CombInstrumentID;
    ///Leg ID
    TApexFtdcLegIDType LegID;
    ///Leg Instrument ID
    TApexFtdcInstrumentIDType LegInstrumentID;
    ///Direction
    TApexFtdcDirectionType Direction;
    ///Leg Multiple
    TApexFtdcLegMultipleType LegMultiple;
    ///Imply Level
    TApexFtdcImplyLevelType ImplyLevel;
};
```

pRspInfo: points to the address for response information structure. The structure:

```
struct CApexFtdcRspInfoField {
    ///Error ID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: returns user request ID for leg instruments query; 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.

6.1.29 OnRspQryBulletin Method

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

Function Prototype:

```
void OnRspQryBulletin(
    CApexFtdcBulletinField *pBulletin,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pBulletin: points to the address for the Exchange bulletin/public announcement structure. The structure:

```
struct CApexFtdcBulletinField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Bulletin No.
    TApexFtdcBulletinIDType BulletinID;
    ///Sequence No.
    TApexFtdcSequenceNoType SequenceNo;
    ///Bulletin type
    TApexFtdcNewsTypeType  NewsType;
    ///Urgency
    TApexFtdcNewsUrgencyType    NewsUrgency;
    ///Transmission time
    TApexFtdcTimeType  SendTime;
    ///Message digest
    TApexFtdcAbstractType   Abstract;
    ///Source of message
    TApexFtdcComeFromType   ComeFrom;
    ///Message body
    TApexFtdcContentType    Content;
    ///WEB address
    TApexFtdcURLLinkType    URLLink;
    ///Market code
    TApexFtdcMarketIDType   MarketID;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
};
```

pRspInfo: points to the address for response information/ message structure. The structure:

```
struct CApexFtdcRspInfoField {
    ///Error ID
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: returns user request ID for the Exchange bulletin query; 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.

6.1.30 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 the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryMarketData(
    CApexFtdcMarketDataField *pMarketData,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pMarketData: points to the address for market data structure. The structure:

```
struct CApexFtdcMarketDataField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///The latest price
    TApexFtdcPriceType LastPrice;
    ///Settlement of yesterday
    TApexFtdcPriceType PreSettlementPrice;
    ///Close of yesterday
    TApexFtdcPriceType PreClosePrice;
    ///Yesterday's open interest
    TApexFtdcLargeVolumeType   PreOpenInterest;
    ///Today's open price
    TApexFtdcPriceType OpenPrice;
```

```
        ///The highest price
        TApexFtdcPriceType HighestPrice;
        ///The lowest price
        TApexFtdcPriceType LowestPrice;
        ///Quantity
        TApexFtdcVolumeType Volume;
        ///Turnover
        TApexFtdcMoneyType Turnover;
        ///Open Interest
        TApexFtdcLargeVolumeType   OpenInterest;
        ///Today's closing
        TApexFtdcPriceType ClosePrice;
        ///Today's settlement
        TApexFtdcPriceType SettlementPrice;
        ///Upward limit price
        TApexFtdcPriceType UpperLimitPrice;
        ///Downward limit price
        TApexFtdcPriceType LowerLimitPrice;
        ///Yesterday's delta value
        TApexFtdcRatioType PreDelta;
        ///Today's delta value
        TApexFtdcRatioType CurrDelta;
        ///Last modification time
        TApexFtdcTimeType UpdateTime;
        ///The last modified millisecond
        TApexFtdcMillisecType UpdateMillisec;
        ///Contract code
        TApexFtdcInstrumentIDType  InstrumentID;

    };
```

pRspInfo: points to the address for response information/ message structure. The structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns user request ID; 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.

6.1.31 OnRspQryMBLMarketData Method

After Member System sends out the query instruction for instrument/contract price and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryMBLMarketData(
    CApexFtdcMBLMarketDataField *pMBLMarketData,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pMBLMarketData: points to the address for price list structure. The structure:

```
struct CApexFtdcMBLMarketDataField {
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Price
    TApexFtdcPriceType  Price;
    ///Quantity
    TApexFtdcVolumeType Volume;
};
```

pRspInfo: points to the address for response information/ message structure. The structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error Message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: returns user request ID. 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.

6.1.32 OnRspQryHedgeVolume Method

After Member System sends out the query instruction for hedge volume and the Trading System sends back the response, this method is called.

Function Prototype:

```
void OnRspQryHedgeVolume(
```

```
        CApexFtdcHedgeVolumeField *pHedgeVolume,
        CApexFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast);
```

Parameters:

pHedgeVolume: points to the address for hedge volume structure. The structure:

```
struct CApexFtdcHedgeVolumeField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Application for initial quantity of long hedge quota (unit: lot)
    TApexFtdcVolumeType LongVolumeOriginal;
    ///Application for initial quantity of short hedge quota (unit: lot)
    TApexFtdcVolumeType ShortVolumeOriginal;
    /// Long hedge quota (unit: lot).
    TApexFtdcVolumeType LongVolume;
    /// Short hedge quota (unit: lot)
    TApexFtdcVolumeType ShortVolume;
};
```

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

```
 struct CApexFtdcRspInfoField {
     /// Error code
     TApexFtdcErrorIDType    ErrorID;
     /// Error message
     TApexFtdcErrorMsgType ErrorMsg;
 };
```

Error code	Error message	Possible reasons
80	User has no permission	Only the conditions under this participant can be queried.
57	Cannot operate for other participant	The conditions under other participants cannot be queried.

nRequestID: returns user request ID; 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.

6.1.33 OnRtnTrade Method

Order match return / trade return: When an order is matched, i.e. when a trade is done, the Trading System will inform Member System, and this method will be called.

Function Prototype:

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

Parameter:

pTrade: pointer to the address for the match return structure. Note: some fields in match return are not used, the Trading System returns space/blank for those unused fields. The structure:

```
struct CApexFtdcTradeField {
    ///Business day
    TApexFtdcDateType TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Transaction No.
    TApexFtdcTradeIDType   TradeID;
    ///Buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Order No.
    TApexFtdcOrderSysIDType OrderSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Trading role, not used
    TApexFtdcTradingRoleType   TradingRole;
    ///Capital account, not used
    TApexFtdcAccountIDType AccountID;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Flag of position opening and closing-out
    TApexFtdcOffsetFlagType OffsetFlag;
    ///Flag of speculation and hedge
    TApexFtdcHedgeFlagType HedgeFlag;
    ///Price
    TApexFtdcPriceType Price;
    ///Quantity
```

```
        TApexFtdcVolumeType Volume;
        ///Transaction time
        TApexFtdcTimeType  TradeTime;
        ///Transaction type, not used
        TApexFtdcTradeTypeType  TradeType;
        ///Source of transaction price, not used
        TApexFtdcPriceSourceType    PriceSource;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Local order No.
        TApexFtdcOrderLocalIDType  OrderLocalID;
        ///Settlement member's No., not used
        TApexFtdcParticipantIDType ClearingPartID;
        ///Business unit, not used
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Calendar Date
        TApexDateType CalendarDate;
        ///Update milli second
        TApexFtdcMillisecType TradeMillisec;

    };
```

6.1.34 OnRtnOrder Method

Order return: When an order is inserted, cancelled or partially match etc., causing the order status changes, the Trading System will automatically inform Member System, and this method will be called.

Function Prototype:

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

Parameter:

pOrder: points to the address for order return structure. Note: some fields in the order return is not used, the Trading System will return an empty/blank value for those unused fields. The structure:

```
    struct CApexFtdcOrderField {
        ///Business day,not used
        TApexFtdcDateType  TradingDay;
        ///Settlement group's code,not used
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Settlement No.,not used
        TApexFtdcSettlementIDType  SettlementID;
        ///Order No.
        TApexFtdcOrderSysIDType OrderSysID;
```

```
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
        TApexFtdcClientIDType ClientID;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Contract code
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Conditions of order price
        TApexFtdcOrderPriceTypeType OrderPriceType;
        ///Buy-sell direction
        TApexFtdcDirectionType Direction;
        ///Flag of position opening and closing-out in a portfolio
        TApexFtdcCombOffsetFlagType CombOffsetFlag;
        ///Flag of speculation and hedge in a portfolio
        TApexFtdcCombHedgeFlagType CombHedgeFlag;
        ///Price
        TApexFtdcPriceType LimitPrice;
        ///Quantity
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///Type of valid period
        TApexFtdcTimeConditionType TimeCondition;
        ///GTD DATE
        TApexFtdcDateType GTDDate;
        ///Volume type
        TApexFtdcVolumeConditionType   VolumeCondition;
        ///The Min.volume
        TApexFtdcVolumeType MinVolume;
        ///Trigger conditions
        TApexFtdcContingentConditionType   ContingentCondition;
        ///Stop-loss price
        TApexFtdcPriceType StopPrice;
        ///Reasons for forced closing-out
        TApexFtdcForceCloseReasonType  ForceCloseReason;
        ///Local order No.
        TApexFtdcOrderLocalIDType  OrderLocalID;
        ///Flag of auto-suspension
        TApexFtdcBoolType IsAutoSuspend;
        ///Source of order, not used
        TApexFtdcOrderSourceType   OrderSource;
        ///Status of order
        TApexFtdcOrderStatusType   OrderStatus;
        ///Type of order, not used
        TApexFtdcOrderTypeType OrderType;
```

```
        ///Volume on that day, not used
        TApexFtdcVolumeType VolumeTraded;
        ///Remaining quantity
        TApexFtdcVolumeType VolumeTotal;
        ///Date of order
        TApexFtdcDateType  InsertDate;
        ///Entry time, not used
        TApexFtdcTimeType  InsertTime;
        ///Time of activation, not used
        TApexFtdcTimeType  ActiveTime;
        ///Time of suspension, not used
        TApexFtdcTimeType  SuspendTime;
        ///Last modification time
        TApexFtdcTimeType UpdateTime;
        ///Time of cancelation, not used
        TApexFtdcTimeType  CancelTime;
        ///Last modification to transaction user's code
        TApexFtdcUserIDType ActiveUserID;
        ///Priority, not used
        TApexFtdcPriorityType  Priority;
        ///Sequence No. of queue by time, not used
        TApexFtdcTimeSortIDType TimeSortID;
        ///Settlement member's No., not used
        TApexFtdcParticipantIDType ClearingPartID;
        ///Business unit, not used
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;
        ///Insert Milli second
        TApexFtdcMillisecType InsertMillisec;
        ///Update Milli second
        TApexFtdcMillisecType UpdateMillisec;
        ///Cancel Milli second
        TApexFtdcMillisecType CancelMillisec;

    };
```

6.1.35 OnRtnQuote Method

Not available in the current version.

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

Function Prototype:

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

Parameter:

pQuote: points to the address for price quote return structure. The structure:

```
struct CApexFtdcQuoteField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Bid Volume
    TApexFtdcVolumeType BidVolume;
    ///Ask Volume
    TApexFtdcVolumeType AskVolume;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Local quote No.
    TApexFtdcOrderLocalIDType  QuoteLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Flag of position opening and closing-out in buyer's portfolio
    TApexFtdcCombOffsetFlagType BidCombOffsetFlag;
    ///Flag of hedge in buyer's portfolio
    TApexFtdcCombHedgeFlagType BidCombHedgeFlag;
    ///Buyer's price
    TApexFtdcPriceType  BidPrice;
    ///Flag of position opening and closing-out in seller's portfolio
    TApexFtdcCombOffsetFlagType AskCombOffsetFlag;
    ///Flag of hedge in seller's portfolio
    TApexFtdcCombHedgeFlagType AskCombHedgeFlag;
    ///Seller's price
    TApexFtdcPriceType  AskPrice;
    ///Entry Time
```

```
        TApexFtdcTimeType  InsertTime;
        ///Time of cancelation
        TApexFtdcTimeType  CancelTime;
        ///Transaction time
        TApexFtdcTimeType  TradeTime;
        ///Buyer's order No.
        TApexFtdcOrderSysIDType BidOrderSysID;
        ///Seller's order No.
        TApexFtdcOrderSysIDType AskOrderSysID;
        ///Settlement member's No.
        TApexFtdcParticipantIDType ClearingPartID;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;

    };
```

6.1.36 OnRtnForQuote Method

Not available in the current version.

Quote return: when Member System sends out quote query instruction, the Trading System will automatically inform Member System, and this method will be called.

The Trading System will only inform the Member System which calls the **SubscribeForQuote** Method to subscribe quote stream.

Function Prototype:

```
void OnRtnForQuote(CApexFtdcInputReqForQuoteField *pReqForQuote);
```

Parameter:

pReqForQuote: points to the address for quote structure. The structure:

```
    struct CApexFtdcInputReqForQuoteField {
        ///Quote ID
        TApexFtdcQuoteSysIDType ReqForQuoteID;
        ///Participant ID
        TApexFtdcParticipantIDType ParticipantID;
        ///Client name
        TApexFtdcClientIDType ClientID;
        ///Instrument/Contract ID
        TApexFtdcInstrumentIDType  InstrumentID;
        ///TradingDay
        TApexFtdcTradingDayType TradingDay;
        ///Quote Time
        TApexFtdcTimeType  ReqForQuoteTime;
        ///Calendar Date
```

```
        TApexFtdcDateType CalendarDate;
    };
```

6.1.37 OnRtnExecOrder Method

Not available in the current version.

Order execution return: The Trading System automatically informs Member System, and this method is called.

Function Prototype:

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

Parameter:

pExecOrder: points to the address for order execution return structure. The structure:

```
    struct CApexFtdcExecOrderField {
        ///Business day
        TApexFtdcDateType TradingDay;
        ///Settlement group's code
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Settlement No.
        TApexFtdcSettlementIDType  SettlementID;
        ///Contract No.
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
        TApexFtdcClientIDType ClientID;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Local No. of execution declaration
        TApexFtdcOrderLocalIDType  ExecOrderLocalID;
        ///Quantity
        TApexFtdcVolumeType Volume;

        ///Business unit
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Execution declaration No.
        TApexFtdcExecOrderSysIDType ExecOrderSysID;
        ///Date of order
        TApexFtdcDateType  InsertDate;
        ///Entry Time
        TApexFtdcTimeType  InsertTime;
        ///Time of cancelation
```

```
    TApexFtdcTimeType  CancelTime;
    ///Execution result
    TApexFtdcExecResultType ExecResult;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;

};
```

6.1.38 OnRtnInstrumentStatus Method

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

Function Prototype:

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

Parameter:

pInstrumentStatus: points to the address for contract/instrument status structure. The structure:

```
struct CApexFtdcInstrumentStatusField {
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Trading status of contract
    TApexFtdcInstrumentStatusType  InstrumentStatus;
    ///No.of trading sessions
    TApexFtdcTradingSegmentSNType  TradingSegmentSN;
    ///Time of entering this status
    TApexFtdcTimeType  EnterTime;
    ///Reasons for entering this status
    TApexFtdcInstStatusEnterReasonType EnterReason;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
};
```

6.1.39 OnRtnInsInstrument Method

Notification for instrument/contract added: After the Member System logs in successfully, the Trading System will send the added contract in the system to the Member System via the public stream.

Function Prototype:

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

Parameter:

pInstrument: points to the address for contract/instrument structure. The structure:

```
struct CApexFtdcInstrumentField {
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Product code
    TApexFtdcProductIDType ProductID;
    ///Product suite's code
    TApexFtdcProductGroupIDType ProductGroupID;
    ///Basic commodity code
    TApexFtdcInstrumentIDType  UnderlyingInstrID;
    ///Product type
    TApexFtdcProductClassType  ProductClass;
    ///Type of open interest
    TApexFtdcPositionTypeType  PositionType;
    ///Strike price
    TApexFtdcPriceType StrikePrice;
    ///Option type
    TApexFtdcOptionsTypeType   OptionsType;
    ///Contract multiplier
    TApexFtdcVolumeMultipleType VolumeMultiple;
    ///Contract multiplier for basic commodity
    TApexFtdcUnderlyingMultipleTypeUnderlyingMultiple;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Contract name
    TApexFtdcInstrumentNameType InstrumentName;
    ///Delivery year
    TApexFtdcYearType DeliveryYear;
    ///Delivery month
    TApexFtdcMonthType DeliveryMonth;
    ///Month in advance
    TApexFtdcAdvanceMonthType  AdvanceMonth;
    ///Is trading right now?
    TApexFtdcBoolType IsTrading;
    ///Currency ID
    TApexFtdcCurrencyIDType CurrencyID;
};
```

6.1.40 OnRtnDelInstrument Method

Not available in the current version.

Notification for instrument/contract deletion: After the Member System logs in successfully, the Trading System will send the deleted contract in the system to the Member System via the public stream.

Function Prototype:

```
void OnRtnDelInstrument(CApexFtdcInstrumentField *pInstrument);
```

Parameter:

pInstrument: points to the address for contract/instrument structure. The structure:

```
struct CApexFtdcInstrumentField {
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Product code
    TApexFtdcProductIDType ProductID;
    ///Product suite's code
    TApexFtdcProductGroupIDType ProductGroupID;
    ///Basic commodity code
    TApexFtdcInstrumentIDType  UnderlyingInstrID;
    ///Product type
    TApexFtdcProductClassType  ProductClass;
    ///Type of open interest
    TApexFtdcPositionTypeType  PositionType;
    ///Strike price
    TApexFtdcPriceType StrikePrice;
    ///Option type
    TApexFtdcOptionsTypeType   OptionsType;
    ///Contract multiplier
    TApexFtdcVolumeMultipleType VolumeMultiple;
    ///Contract multiplier for basic commodity
    TApexFtdcUnderlyingMultipleTypeUnderlyingMultiple;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Contract name
    TApexFtdcInstrumentNameType InstrumentName;
    ///Delivery year
    TApexFtdcYearType DeliveryYear;
    ///Delivery month
    TApexFtdcMonthType DeliveryMonth;
    ///Month in advance
    TApexFtdcAdvanceMonthType  AdvanceMonth;
    ///Is trading right now?
    TApexFtdcBoolType IsTrading;
    ///Currency ID
    TApexFtdcCurrencyIDType CurrencyID;
```

```
};
```

6.1.41 OnRtnInsCombinationLeg Method

Not available in the current version.

This function is used for notification on addition of single leg of contract. When one successfully logged into member system, trading system will notify member system about the addition of sinle leg of portfolio contract in system via public stream.

Function prototype:

```
void OnRtnInsCombinationLeg(
    CApexFtdcCombinationLegField *pCombinationLeg);
```

Parameter:

pCombinationLeg: Address pointing to structure of single leg of portfolio trading contract. The structure of single leg of portfolio trading contract:

```
struct CApexFtdcCombinationLegField {
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Portfolio contract code
    TApexFtdcInstrumentIDType  CombInstrumentID;
    ///Single leg No.
    TApexFtdcLegIDType LegID;
    ///Single leg contract code
    TApexFtdcInstrumentIDType  LegInstrumentID;
    ///Buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Single leg multiplier
    TApexFtdcLegMultipleType   LegMultiple;
    ///Deduction of layers
    TApexFtdcImplyLevelType ImplyLevel;
};
```

6.1.42 OnRtnDelCombinationLeg Method

Not available in the current version.

This method is used for notification on deletion of single leg of contract. When Member System successfuly logged into the Trading System, the Trading System will notify Member System about the deletion of single leg of portfolio contact in system via public stream.

Function prototype:

```
void OnRtnDelCombinationLeg(
    CApexFtdcCombinationLegField *pCombinationLeg);
```

Parameter:

pCombinationLeg: Address pointing to structue of single leg of trading contract. Structure of single leg of portfolio trading contract:

```
struct CApexFtdcCombinationLegField {
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Portfolio contract code
    TApexFtdcInstrumentIDType  CombInstrumentID;
    ///Single leg No.
    TApexFtdcLegIDType  LegID;
    ///Single leg contract code
    TApexFtdcInstrumentIDType  LegInstrumentID;
    ///Buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Single leg multiplier
    TApexFtdcLegMultipleType   LegMultiple;
    ///Deduction of layers
    TApexFtdcImplyLevelType ImplyLevel;
};
```

6.1.43 OnRtnBulletin Method

Notificaton for bulletin: When the Exchange sends announcement through the Trading System, the Trading System will automatically inform Member System, and this method is called.

Function Prototype:

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

Parameter:

pBulletin: points to the address for bulletin/annoucement structure. The structure:

```
struct CApexFtdcBulletinField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Bulletin No.
    TApexFtdcBulletinIDType BulletinID;
    ///Sequence No.
    TApexFtdcSequenceNoType SequenceNo;
    ///Bulletin type
    TApexFtdcNewsTypeType  NewsType;
    ///Urgency
    TApexFtdcNewsUrgencyType   NewsUrgency;
    ///Transmission time
    TApexFtdcTimeType  SendTime;
    ///Message digest
    TApexFtdcAbstractType  Abstract;
```

```
    ///Source of message
    TApexFtdcComeFromType   ComeFrom;
    ///Message body
    TApexFtdcContentType    Content;
    ///WEB address
    TApexFtdcURLLinkType    URLLink;
    ///Market code
    TApexFtdcMarketIDType   MarketID;
    ///Calendar Date
    TApexFtdcDateType Calendar Date;
};
```

6.1.44 OnRtnAliasDefine Method

Not available in the current version.

Notification for alias definition: The Trading System automatically informs Member System, and this method is called.

Function Prototype:

```
void OnRtnAliasDefine(CApexFtdcAliasDefineField *pAliasDefine);
```

Parameter:

pAliasDefine: points to the address for alias definition structure. The structure:

```
struct CApexFtdcAliasDefineField {
    ///Starting position
    TApexFtdcStartPosType StartPos;
    ///Alias
    TApexFtdcAliasType Alias;
    ///Original text
    TApexFtdcOriginalTextType  OriginalText;
};
```

6.1.45 OnRtnFlowMessageCancel Method

Notification for data stream cancellation: after the Trading System switches to the disaster recevery site, when user relogin the Trading System and subscribe to a data stream (private stream or public stream), the Trading System will automatically inform the Member System that some messages in that data stream is cancelled, and this method is called.

Function Prototype:

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

Parameter:

pFlowMessageCancel: points to the address for data stream cancellation structure. The structure:

```
    struct CApexFtdcFlowMessageCancelField {
        /// Serial No. in sequence
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Business day
        TApexFtdcDateType  TradingDay;
        ///Datacenter code
        TApexFtdcDataCenterIDType  DataCenterID;
        /// Starting sequence No. of rollback
        TApexFtdcSequenceNoType StartSequenceNo;
        ///Ending sequence No. of rollback
        TApexFtdcSequenceNoType EndSequenceNo;
    };
    SequenceSeries: Data stream code of rollback occured (Private stream or
public stream)
    Message range of rollback: (StartSequenceNo, EndSequenceNo]
```

6.1.46 OnErrRtnOrderInsert Method

Order entry error return: sent automatically by the Trading System to Member System, this method is called.

Function Prototype:

```
    void OnErrRtnOrderInsert(
        CApexFtdcInputOrderField *pInputOrder,
        CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pInputOrder: points to the address for order insert structure, including the input data while submitting the order entry and the order ID returned from the Trading System. The structure:

```
    struct CApexFtdcInputOrderField {
        ///Order No., this feild will be returned by trading system.
        TApexFtdcOrderSysIDType OrderSysID;
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
        TApexFtdcClientIDType ClientID;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Contract code
        TApexFtdcInstrumentIDType  InstrumentID;
        ///Conditions of order price
        TApexFtdcOrderPriceTypeType OrderPriceType;
```

```
        ///Buy-sell direction
        TApexFtdcDirectionType Direction;
        ///Flag of position opening and closing-out in a portfolio
        TApexFtdcCombOffsetFlagType CombOffsetFlag;
        ///Flag of speculation and hedge in a portfolio
        TApexFtdcCombHedgeFlagType CombHedgeFlag;
        ///Price
        TApexFtdcPriceType LimitPrice;
        ///Quantity
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///Type of valid period
        TApexFtdcTimeConditionType TimeCondition;
        ///GTD DATE
        TApexFtdcDateType GTDDate;
        ///Volume type
        TApexFtdcVolumeConditionType  VolumeCondition;
        ///The Min.volume
        TApexFtdcVolumeType MinVolume;
        ///Trigger conditions
        TApexFtdcContingentConditionType  ContingentCondition;
        ///Stop-loss price
        TApexFtdcPriceType StopPrice;
        ///Reasons for forced closing-out
        TApexFtdcForceCloseReasonType ForceCloseReason;
        ///Local order No.
        TApexFtdcOrderLocalIDType OrderLocalID;
        ///Flag of auto-suspension
        TApexFtdcBoolType IsAutoSuspend;
        ///Business unit
        TApexFtdcBusinessUnitType BusinessUnit;
    };
```

pRspInfo: points to the address for the information/message strucuture. The structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType   ErrorID;
        ///Error Message
        TApexFtdcErrorMsgType  ErrorMsg;
    };
```

6.1.47 OnErrRtnOrderAction Method

Order action/operation error return: sent automatically by the Trading System to Member System, this method is called.

Function Prototype:

```
void OnErrRtnOrderAction (
    CApexFtdcOrderActionField *pOrderAction,
    CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pOrderAction: point to the address for order action/operation structure, including the input data while submitting the order action/operaction and the order ID returned from the Trading System. The structure:

```
struct CApexFtdcOrderActionField {
    ///Order No., this field will be returned by trading system.
    TApexFtdcOrderSysIDType OrderSysID;
    ///Local order No.
    TApexFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Price
    TApexFtdcPriceType LimitPrice;
    ///Change in quantity
    TApexFtdcVolumeType VolumeChange;
    ///Local No. of operation
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

pRspInfo: points to the address for the information/message strucuture. The structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

6.1.48 OnErrRtnQuoteInsert Method

Not available in the current version.

Return on erroneous quote entry. When the Member System was notified by the Trading System of such message, this method will be called.

Function prototype:

```
void OnErrRtnQuoteInsert(
    CApexFtdcInputQuoteField *pInputQuote,
    CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pInputQuote: Address pointing to the input quote structue, including the input data for quote entry operation and the quote No. returned from trading system. The input quote structure:

```
struct CApexFtdcInputQuoteField {
    ///Quote No.,this field will be returned by trading system.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Bid Volume
    TApexFtdcVolumeType BidVolume;
    ///Ask Volume
    TApexFtdcVolumeType AskVolume;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Local quote No.
    TApexFtdcQuoteLocalIDType  QuoteLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Flag of position opening and closing-out in buyer's portfolio
    TApexFtdcCombOffsetFlagType BidCombOffsetFlag;
    ///Flag of hedge in buyer's portfolio
    TApexFtdcCombHedgeFlagType BidCombHedgeFlag;
    ///Buyer's price
    TApexFtdcPriceType  BidPrice;
    ///Flag of position opening and closing-out in seller's portfolio
    TApexFtdcCombOffsetFlagType AskCombOffsetFlag;
    ///Flag of hedge in seller's portfolio
    TApexFtdcCombHedgeFlagType AskCombHedgeFlag;
    ///Seller's price
    TApexFtdcPriceType  AskPrice;
```

```
};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

6.1.49 OnErrRtnQuoteAction Method

Not available in the current version.

Return on erroneous quote operation. When Member System was notified by the Trading System of such message, this message will be called.

Function prototype:

```
void OnErrRtnQuoteAction(
    CApexFtdcQuoteActionField *pQuoteAction,
    CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pQuoteAction: Address pointing to quote operation structure, including the input data for quote operation request and the quote No. returned from trading system. Quote operation structure. Quote operation structure:

```
struct CApexFtdcQuoteActionField {
    ///Quote No.,this field will be returned by trading system.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Local quote No.
    TApexFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Local No. of operation
    TApexFtdcOrderLocalIDType   ActionLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType   BusinessUnit;

};
```

pRspInfo: Address pointing to response message structure. The response message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

6.1.50 OnErrRtnExecOrderInsert Method

Not available in the current version.

Return on erroneous entry of execution declaration. When Member System was notified by the Trading System of such message, this method will be called.

Function prototype:

```
void OnErrRtnExecOrderInsert(
    CApexFtdcInputExecOrderField *pInputExecOrder,
    CApexFtdcRspInfoField *pRspInfo);
```

Function:

pInputExecOrder: Address ponting to execution declaration entry structure. The execution declaration entry structure:

```
struct CApexFtdcInputExecOrderField {
    ///Contract No.
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Local execution declaration No.
    TApexFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Quantity
    TApexFtdcVolumeType Volume;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

pRspInfo: Address pointing to response message structure. The response message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
```

```
        ///Error message
        TApexFtdcErrorMsgType ErrorMsg;
    };
```

6.1.51 OnErrRtnExecOrderAction Method

Not available in the current version.

Return on erroneous operation of execution declaration. When member system was notified by trading system of such message, this method will be called.

Function prototype:

```
    void OnErrRtnExecOrderAction (
        CApexFtdcExecOrderActionField *pExecOrderAction,
        CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pInputExecAction: Address pointing to declaration operation structure. Declaration operation structure:

```
struct CApexFtdcExecOrderActionField {
    ///Execution declaration No.
    TApexFtdcExecOrderSysIDType ExecOrderSysID;
    ///Local execution declaration No.
    TApexFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Local No. of operation
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;


};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
```

```
};
```

6.1.52 OnRspQryCombOrder Method

Not available in the current version.

Response to query for uncommon portfolio order. When Member System was notified by the Trading System of such message, this method will be called.

Function prototype:

```
void OnRspCombOrderInsert(
    CApexFtdcCombOrderField *pCombOrder,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameter:

pCombOrder: Address pointing to structure of uncommon portfolio order. Structure of uncommon portfolio order:

```
struct CApexFtdcCombOrderField {
    ///Business day
    TApexFtdcDateType  TradingDay;
    ///Settlement group's code
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Portfolio order No.
    TApexFtdcOrderSysIDType CombOrderSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Price
    TApexFtdcPriceType  LimitPrice;
    ///Quantity
    TApexFtdcVolumeType VolumeTotalOriginal;
    ///Local order No.
    TApexFtdcOrderLocalIDType  CombOrderLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Contract code 1
    TApexFtdcInstrumentIDType  InstrumentID1;
    ///Buy-sell direction 1
    TApexFtdcDirectionType  Direction1;
```

```
        ///Separate leg multiplier 1
        TApexFtdcLegMultipleType    LegMultiple1;
        ///Flag of position opening and closing-out 1
        TApexFtdcOffsetFlagType OffsetFlag1;
        ///Flag of speculation and hedge 1
        TApexFtdcHedgeFlagType  HedgeFlag1;
        ///Contract code 2
        TApexFtdcInstrumentIDType  InstrumentID2;
        ///Buy-sell direction 2
        TApexFtdcDirectionType Direction2;
        ///Separate leg multiplier 2
        TApexFtdcLegMultipleType    LegMultiple2;
        ///Flag of position opening and closing-out 2
        TApexFtdcOffsetFlagType OffsetFlag2;
        ///Flag of speculation and hedge 2
        TApexFtdcHedgeFlagType  HedgeFlag2;
        ///Contract code 3
        TApexFtdcInstrumentIDType  InstrumentID3;
        ///Buy-sell direction 3
        TApexFtdcDirectionType Direction3;
        ///Separate leg multiplier 3
        TApexFtdcLegMultipleType    LegMultiple3;
        ///Flag of position opening and closing-out 3
        TApexFtdcOffsetFlagType OffsetFlag3;
        ///Flag of speculation and hedge 3
        TApexFtdcHedgeFlagType  HedgeFlag3;
        ///Contract code 4
        TApexFtdcInstrumentIDType  InstrumentID4;
        ///Buy-sell direction 4
        TApexFtdcDirectionType Direction4;
        ///Separate leg multiplier 4
        TApexFtdcLegMultipleType    LegMultiple4;
        ///Flag of position opening and closing-out 4
        TApexFtdcOffsetFlagType OffsetFlag4;
        ///Flag of speculation and hedge 4
        TApexFtdcHedgeFlagType  HedgeFlag4;
        ///Source of order
        TApexFtdcOrderSourceType    OrderSource;
        ///Volume on that day
        TApexFtdcVolumeType VolumeTraded;
        ///Remaining quantity
        TApexFtdcVolumeType VolumeTotal;
        ///Date of order
        TApexFtdcDateType  InsertDate;
```

```
        ///Time of entry
        TApexFtdcTimeType  InsertTime;
        ///Settlement member's No.
        TApexFtdcParticipantIDType ClearingPartID;
        ///Calendar Date
        TApexFtdcDateType CalendarDate;
    };
```

pRspInfo: Address pointing to response message structure. The response message structure:

```
    struct CApexFtdcRspInfoField {
        ///Error code
        TApexFtdcErrorIDType    ErrorID;
        ///Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: ID for request for uncommon portfolio order query. This ID will be designated and managed by user.

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

6.1.53 OnRtnCombOrder Method

Not available in the current version.

Return on uncommon portfolio order. When Member System was notified by the Trading System of such message, this method willbe called.

Function prototype:

```
    void OnRtnCombOrder (CApexFtdcCombOrderField *pCombOrder);
```

Parameter:

pCombOrder: Address pointing to structure of uncommon porfolio order. Structure of uncommon porfolio order:

```
    struct CApexFtdcCombOrderField {
        ///Business day
        TApexFtdcDateType  TradingDay;
        ///Settlement group's code
        TApexFtdcSettlementGroupIDType SettlementGroupID;
        ///Settlement No.
        TApexFtdcSettlementIDType  SettlementID;
        ///Portfolio order No.
        TApexFtdcOrderSysIDType CombOrderSysID;
        ///Member code
        TApexFtdcParticipantIDType ParticipantID;
        ///Client code
```

```
        TApexFtdcClientIDType  ClientID;
        ///Transaction user's code
        TApexFtdcUserIDType UserID;
        ///Price
        TApexFtdcPriceType  LimitPrice;
        ///Quantity
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///Local order No.
        TApexFtdcOrderLocalIDType  CombOrderLocalID;
        ///Business unit
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Contract code 1
        TApexFtdcInstrumentIDType  InstrumentID1;
        ///Buy-sell direction 1
        TApexFtdcDirectionType Direction1;
        ///Separate leg multiplier 1
        TApexFtdcLegMultipleType   LegMultiple1;
        ///Flag of position opening and closing-out 1
        TApexFtdcOffsetFlagType OffsetFlag1;
        ///Flag of speculation and hedge 1
        TApexFtdcHedgeFlagType HedgeFlag1;
        ///Contract code 2
        TApexFtdcInstrumentIDType  InstrumentID2;
        ///Buy-sell direction 2
        TApexFtdcDirectionType Direction2;
        ///Separate leg multiplier 2
        TApexFtdcLegMultipleType   LegMultiple2;
        ///Flag of position opening and closing-out 2
        TApexFtdcOffsetFlagType OffsetFlag2;
        ///Flag of speculation and hedge 2
        TApexFtdcHedgeFlagType HedgeFlag2;
        ///Contract code 3
        TApexFtdcInstrumentIDType  InstrumentID3;
        ///Buy-sell direction 3
        TApexFtdcDirectionType Direction3;
        ///Separate leg multiplier 3
        TApexFtdcLegMultipleType   LegMultiple3;
        ///Flag of position opening and closing-out 3
        TApexFtdcOffsetFlagType OffsetFlag3;
        ///Flag of speculation and hedge 3
        TApexFtdcHedgeFlagType HedgeFlag3;
        ///Contract code 4
        TApexFtdcInstrumentIDType  InstrumentID4;
        ///Buy-sell direction 4
```

```
    TApexFtdcDirectionType Direction4;
    ///Separate leg multiplier 4
    TApexFtdcLegMultipleType   LegMultiple4;
    ///Flag of position opening and closing-out 4
    TApexFtdcOffsetFlagType OffsetFlag4;
    ///Flag of speculation and hedge 4
    TApexFtdcHedgeFlagType HedgeFlag4;
    ///Source of order
    TApexFtdcOrderSourceType   OrderSource;
    ///Volume on that day
    TApexFtdcVolumeType VolumeTraded;
    ///Remaining quantity
    TApexFtdcVolumeType VolumeTotal;
    ///Date of order
    TApexFtdcDateType   InsertDate;
    ///Time of entry
    TApexFtdcTimeType   InsertTime;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
};
```

6.1.54 OnErrRtnCombOrderInsert Method

Not available in the current version.

Return on erroneous entry of porfolio into an order. When Member System was notified by the Trading System of such message, this method will be called.

Function prototype:

```
void OnErrRtnCombOrderInsert (
    CApexFtdcInputCombOrderField *pInputCombOrder,
    CApexFtdcRspInfoField *pRspInfo);
```

Parameters:

pInputCombOrder: Address pointing to structure of entry of uncommon porfolio order. The structure of entry of uncommon porfolio order:

```
struct CApexFtdcInputCombOrderField {
    ///Portfolio order No.
    TApexFtdcOrderSysIDType CombOrderSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
```

```
        TApexFtdcUserIDType UserID;
        ///Price
        TApexFtdcPriceType  LimitPrice;
        ///Quantity
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///Local order No.
        TApexFtdcOrderLocalIDType  CombOrderLocalID;
        ///Business unit
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Contract code 1
        TApexFtdcInstrumentIDType  InstrumentID1;
        ///Buy-sell direction 1
        TApexFtdcDirectionType Direction1;
        ///Separate leg multiplier 1
        TApexFtdcLegMultipleType   LegMultiple1;
        ///Flag of position opening and closing-out 1
        TApexFtdcOffsetFlagType OffsetFlag1;
        ///Flag of speculation and hedge 1
        TApexFtdcHedgeFlagType HedgeFlag1;
        ///Contract code 2
        TApexFtdcInstrumentIDType  InstrumentID2;
        ///Buy-sell direction 2
        TApexFtdcDirectionType Direction2;
        ///Separate leg multiplier 2
        TApexFtdcLegMultipleType   LegMultiple2;
        ///Flag of position opening and closing-out 2
        TApexFtdcOffsetFlagType OffsetFlag2;
        ///Flag of speculation and hedge 2
        TApexFtdcHedgeFlagType HedgeFlag2;
        ///Contract code 3
        TApexFtdcInstrumentIDType  InstrumentID3;
        ///Buy-sell direction 3
        TApexFtdcDirectionType Direction3;
        ///Separate leg multiplier 3
        TApexFtdcLegMultipleType   LegMultiple3;
        ///Flag of position opening and closing-out 3
        TApexFtdcOffsetFlagType OffsetFlag3;
        ///Flag of speculation and hedge 3
        TApexFtdcHedgeFlagType HedgeFlag3;
        ///Contract code 4
        TApexFtdcInstrumentIDType  InstrumentID4;
        ///Buy-sell direction 4
        TApexFtdcDirectionType Direction4;
        ///Separate leg multiplier 4
```

```
    TApexFtdcLegMultipleType    LegMultiple4;
    ///Flag of position opening and closing-out 4
    TApexFtdcOffsetFlagType OffsetFlag4;
    ///Flag of speculation and hedge 4
    TApexFtdcHedgeFlagType HedgeFlag4;
};
```

pRspInfo: Address pointing to response message structure. The response message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

6.1.55 OnRspAdminOrderInsert Method

This method is for the reply on the administrator order entry. the Trading System will inform Member System, and this method will be called.

Function prototype:

```
void OnRspAdminOrderInsert(
    CApexFtdcInputAdminOrderField *pInputAdminOrder,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pInputAdminOrder: points to the address for administrator order structure. The structure:

```
struct CApexFtdcInputAdminOrderField {
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///administrator's command
    TApexFtdcAdminOrderCommandFlagType AdminOrderCommand;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;
    ///trading member's No
    TApexFtdcParticipantIDType ParticipantID;
    ///Amount
    TApexFtdcMoneyType  Amount;
```

```
    ///SettlementGroup ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
};
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
struct CApexFtdcRspInfoField {
    ///Error code
    TApexFtdcErrorIDType    ErrorID;
    ///Error message
    TApexFtdcErrorMsgType   ErrorMsg;
};
```

nRequestID: ID for request for administrator order query. This ID will be designated and managed by user.

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

6.1.56 OnRspQryCreditLimit Method

After Member System sends out the query instruction for hedge volume and the Trading System sends back the response, this method will be called.

Function prototype:

```
void OnRspQryCreditLimit(
    CApexFtdcCreditLimitField *pCreditLimit,
    CApexFtdcRspInfoField *pRspInfo,
    int nRequestID,
    bool bIsLast);
```

Parameters:

pQryCreaditLimit: points to the address for credit limit query structure. The structure:

```
struct CApexFtdcCreditLimitField {
    ///Business Day
    TApexFtdcDateType  TradingDay;
    ///SettlementGroup ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Settlement No.
    TApexFtdcSettlementIDType  SettlementID;
    ///Reserve funds for previous settlement
    TApexFtdcMoneyType  PreBalance;
    ///Total margin at present
    TApexFtdcMoneyType  CurrMargin;
```

```
        ///Profit & loss on closing-out of position
        TApexFtdcMoneyType  CloseProfit;
        ///Income and expense from option premium
        TApexFtdcMoneyType  Premium;
        ///Deposit amount
        TApexFtdcMoneyType  Deposit;
        ///Withdrawal amount
        TApexFtdcMoneyType  Withdraw;
        ///Reserve funds for futures settlement
        TApexFtdcMoneyType  Balance;
        ///Withdrawable funds
        TApexFtdcMoneyType  Available;
        ///trading member's No.
        TApexFtdcParticipantIDType ParticipantID;
        ///Settlement member's No.
        TApexFtdcParticipantIDType ClearingPartID;
        ///Frozen margin
        TApexFtdcMoneyType  FrozenMargin;
        ///Frozen premium
        TApexFtdcMoneyType  FrozenPremium;
    };
```

pRspInfo: Address pointing to response message structure. Response message structure:

```
    struct CApexFtdcRspInfoField {
        /// Error code
        TApexFtdcErrorIDType    ErrorID;
        /// Error message
        TApexFtdcErrorMsgType   ErrorMsg;
    };
```

nRequestID: returns user request ID of user's query for credit limit; 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.

6.1.57 OnRtnMarketData Method

Order match return / trade return: When previous settlement price is updated, the Trading System will inform Member System, and this method will be called.

Function Prototype:

```
    void OnRtnMarketDataCApexFtdcMarketDataField* pMarketData);
```

Parameter:

pMarketData: pointer to the address for the market data return structure. Note: some fields are not used, the Trading System returns space/blank for those unused fields. The structure:

```
    struct CApexFtdcMarketDataField {
      ///Trading Day
      TApexFtdcDateType   TradingDay;
      ///Settlement Group ID
      TApexFtdcSettlementGroupIDType    SettlementGroupID;
      ///Settlement ID
      TApexFtdcSettlementIDType  SettlementID;
      ///Last Price
      TApexFtdcPriceType  LastPrice;
      ///Previous Settlement Price
      TApexFtdcPriceType  PreSettlementPrice;
      ///Previous Close Price
      TApexFtdcPriceType  PreClosePrice;
      ///Previous Open Interest
      TApexFtdcLargeVolumeType   PreOpenInterest;
      ///Open Price
      TApexFtdcPriceType  OpenPrice;
      ///Highest Price
      TApexFtdcPriceType  HighestPrice;
      ///Lowest Price
      TApexFtdcPriceType  LowestPrice;
      ///Volume
      TApexFtdcVolumeType Volume;
      ///Turnover
      TApexFtdcMoneyType  Turnover;
      ///Open Interest
      TApexFtdcLargeVolumeType   OpenInterest;
      ///Close Price
      TApexFtdcPriceType  ClosePrice;
      ///Settlement Price
      TApexFtdcPriceType  SettlementPrice;
      ///Upper Limit Price
      TApexFtdcPriceType  UpperLimitPrice;
      ///Lower Limit Price
      TApexFtdcPriceType  LowerLimitPrice;
      ///Previoius Delta
      TApexFtdcRatioType  PreDelta;
      ///Current Delta
      TApexFtdcRatioType  CurrDelta;
      ///Update Time
      TApexFtdcTimeType   UpdateTime;
      ///Update Milli second
      TApexFtdcMillisecType      UpdateMillisec;
      ///Instrument ID
      TApexFtdcInstrumentIDType  InstrumentID;
};
```

6.2 CApexFtdcTraderApi Interfaces

Functions offered by the **CApexFtdcTraderApi** interfaces include order insert, order cancellation, order query, trade (or matched/filled order) query, member client query, member holding position query, client holding position query, contract/instrument query, contract/instrument trading status query, Exchange bulletin query, etc. The System has a frequency quota/limit (i.e. number of instructions sent every second) for sending instruction for each seat. Once the quota is exceeded, the instructions sent out will be blocked in the network. Please consult the relevant department of the Exchange for specific quota number.

6.2.1 CreateFtdcTraderApi Method

Creates an instance of the **CApexFtdcTraderApi.** This cannot be created with a "new".

Function Prototype:

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

Parameter:

pszFlowPath: constant character pointer, used to point to a directory that stores the local files generated by TraderAPI. The default value is the current directory.

Return Value:

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

6.2.2 GetVersion Method

Gets the API version.

Function Prototype:

```
    static const char *GetVersion();
```

Return Value:

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

6.2.3 Release Method

This is the proper method (instead of delete keyword) to release an instance of **CApexFtdcTraderApi.**

Function Prototype:

```
    void Release();
```

6.2.4 Init Method

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

Function Prototype:

```
    void Init();
```

6.2.5 Join Method

Member System waits for the end of an interface thread instance.

Function Prototype:

```
    void Join();
```

6.2.6 GetTradingDay Method

Gets the current trading day. The correct value can only be obtained after successful login to the Trading System.

Function Prototype:

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

Return Value:

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

6.2.7 RegisterSpi Method

Registers an instance derived from **CApexFtdcTraderSpi** class that performs events handling.

Function Prototype:

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

Parameter:

pSpi: the pointer for **ApexFtdcTraderSpi** interface instance.

6.2.8 RegisterFront Method

Registers network communication address of the Trading System gateway. The Trading System has multiple gateways and the Member System can register multiple network communication addresses of the gateways.

This method has to be called before the **Init Method** is called.

Function Prototype:

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

Parameter:

pszFrontAddress: a pointer that points to the gateway network communication address. The server address is in the format "**protocol://ipaddress:port**", e.g. "tcp://127.0.0.1:17001". "tcp" in the example is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represents the server port number.

6.2.9 RegisterNameServer Method

Registers the network communication address of the Trading System NameServer. The Trading System has multiple NameServer and the Member System can register multiple NameServer network communication addresses.

This method has to be called before the **Init Method** is called.

Function Prototype:

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

Parameter:

pszNsAddress: a pointer that points to the Exchange NameServer network communication address. The network address is in the format "**protocol://ipaddress:port**",e.g. "tcp://127.0.0.1:17001". in the example is the transmission protocol, "127.0.0.1" represents the server address, and "17001" represents the server port number.

6.2.10 SetHeartbeatTimeout Method

Sets the heartbeat timeout limit for network communication. After the connection between TraderAPI and the Trading System is established, it will send regular heartbeat to detect whether the connection is functioning well. **The Exchange suggests members to set the timeout to be between 10s and 30s.**

Function Prototype:

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

Parameter:

Timeout: heartbeat timeout time limit (in seconds). If no information/message is received from the Trading System after "timeout/2" seconds, **CApexFtdcTraderApi::OnHeartBeatWarning()** will be called/triggered. If no information/message is received from the Trading System after "timeout" seconds, the connection will be stopped, and **CApexFtdcTraderApi::OnFrontDisconnected()** will be called/triggered.

Please refer to Part I Section 4.8 for the heartbeat mechanism.

6.2.11 OpenRequestLog Method

Opens 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:

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

Parameter:

pszReqLogFileName: the request log file name.

6.2.12 OpenResponseLog Method

Opens 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:

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

Parameter:

pszRspLogFileName: reply log file name.

6.2.13 SubscribePrivateTopic Method

Subscribes to member-specific private stream. This method has to be called before the **Init** Method. If this method is not called, no private stream data will be received.

Function Prototype:

```
void SubscribePrivateTopic(APEX_TE_RESUME_TYPE nResumeType);
```

Parameter:

nResumeType: private stream retransmission method types:

- TERT_RESTART: to retransmit from current trading day
- TERT_RESUME: to retransmit by resuming and continuing from last transmission; **In order to ensure member trading data completeness/integrity, the Exchange recommend member to use this method of receiving private stream, and member should deal with other order operations after current day trading data is resumed/recovered.**
- TERT_QUICK: to only transmit those post-current-login member-specific private stream contents; **the Exchange does not recommend members to use this method of receiving private stream.**

6.2.14 SubscribePublicTopic Method

Subscribes to public stream. This method has to be called before the **Init** Method. If this method is not called, no public stream data will be received.

Function Prototype:

```
void SubscribePublicTopic(APEX_TE_RESUME_TYPE nResumeType);
```

Parameter:

nResumeType: public stream retransmission method types:

- TERT_RESTART: to retransmit from current trading day
- TERT_RESUME: to retransmit by resuming and continuing from last transmission
- TERT_QUICK: to only transmit those post-current-login member-specific private stream contents

6.2.15 SubscribeUserTopic Method

Subsribes to trader-specific private stream. This method has to be called before the **Init** Method. If this method is not called, no trader-specific private stream data will be received.

Function Prototype:

```
void SubscribeUserTopic(APEX_TE_RESUME_TYPE nResumeType);
```

Parameter:

nResumeType: private stream retransmission method types (similar to **Section 2.2.13** above):

- TERT_RESTART: to retransmit from current trading day
- TERT_RESUME: to retransmit by resuming and continuing from last transmission. **In order to ensure member trading data completeness/integrity, the Exchange recommend member to use this method of receiving private stream, and member should deal with other order operations after current day trading data is resumed/recovered.**
- TERT_QUICK: to only transmit those post-current-login member-specific private stream contents. **The Exchange does not recommend members to use this method of receiving private stream.**

6.2.16 SubscribeForQuote Method

Not available in the current version.

This is to subscribe to client quote stream. This method has to be called before the **Init** Method. If this method is not called, no trader-specific private stream data will be received.

Function Prototype:

```
void SubscribeForQuote(APEX_TE_RESUME_TYPE nResumeType);
```

Parameter:

nResumeType: private stream retransmission method types (similar to **Section 2.2.13** above):

- TERT_RESTART: to retransmit from current trading day
- TERT_RESUME: to retransmit by resuming and continuing from last transmission. **In order to ensure member trading data completeness/integrity, the Exchange recommend member to use this method of receiving private stream, and member should deal with other order operations after current day trading data is resumed/recovered.**

- TERT_QUICK: to only transmit those post-current-login member-specific private stream contents. **The Exchange does not recommend members to use this method of receiving private stream.**

6.2.17 ReqUserLogin Method

This is the user login request.

Function Prototype:

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

Parameter:

pReqUserLoginField: points to the address for login request structure. The structure:

```
struct CApexFtdcReqUserLoginField {
    ///Business day
    TApexFtdcDateType TradingDay;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Password
    TApexFtdcPasswordType Password;
    ///The user-end product information
    TApexFtdcProductInfoType   UserProductInfo;
    ///The interface-port product information, not used
    TApexFtdcProductInfoType   InterfaceProductInfo;
    ///Protocol information, not used
    TApexFtdcProtocolInfoType  ProtocolInfo;
    ///Datacenter code
    TApexFtdcDataCenterIDType  DataCenterID;
};
```

The Member System is required to fill its system name and version in "UserProductInfo" field. For example, ”ABC Trading System V100” represents the trading program and version No. developed by ABC firm.

If member system maintains the sequence No. of retransmission on its own, then the "TradingDay" and "DataCenterID" shall be the same as return value from previous login. If it is the first login or no resuming of transmission is required, then the "TradingDay" can be filled as empty string (“”) while DataCenterID can be filled as 0 or as the primary datacenter code published by the Exchange.

nRequestID: the request ID for login request; it is specified and managed by the user.

Return Value:

- 0, success
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.18 ReqUserLogout Method

This is the user logout request.

Function Prototype:

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

Parameter:

pReqUserLogout: points to the address for logout request structure. The structure:

```
struct CApexFtdcReqUserLogoutField {
    ///Trading User ID
    TApexFtdcUserIDType UserID;
    ///Member ID
    TApexFtdcParticipantIDType ParticipantID;
};
```

nRequestID: the request ID for logout request; it is specified and managed by the user.

Return Value:

- 0, success
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.19 ReqUserPasswordUpdate Method

This is the user password update request.

Function Prototype:

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

Parameters:

pUserPasswordUpdate: points to the address for user password update structure. The structure:

```
struct CApexFtdcUserPasswordUpdateField {
    ///Trading User ID
    TApexFtdcUserIDType UserID;
    ///Member ID
    TApexFtdcParticipantIDType ParticipantID;
    ///Old Password
    TApexFtdcPasswordType  OldPassword;
    ///New Password
    TApexFtdcPasswordType  NewPassword;
};
```

nRequestID: the request ID for user password update request; it is specified and managed by the user.

Return Value:

- 0, success
- -1,network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.20 ReqSubscribeTopic Method

This is the request to subscribe to topic/theme. This should be called after login.

Function Prototype:

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

Parameters:

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

```
struct CApexFtdcDisseminationField {
    ///Sequence series number
```

```
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence number
        TApexFtdcSequenceNoType SequenceNo;
    };
SequenceSeries: topics to be subscribed
SequenceNo: =-1 to retransmit using the "QUICK" method
= other value, to resume transmission from this sequence number onwards
```

nRequestID: the request ID; it is specified and managed by the user.

Return Value:

- 0, success
- -1,network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.21 ReqQryTopic Method

This is the request for querying topic/theme. This should be called after login.

Function Prototype:

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

Parameter:

pDissemination: points to the address for topic query structure, including topic to be queried. The structure:

```
    struct CApexFtdcDisseminationField {
        ///Sequence Series
        TApexFtdcSequenceSeriesType SequenceSeries;
        ///Sequence Number
        TApexFtdcSequenceNoType SequenceNo;
    };
SequenceSeries: topics to be queried
SequenceNo: no need to fill in
```

nRequestID: the request ID; it is specified and managed by the user.

Return Value:

- 0, success

- -1,network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.22 ReqOrderInsert Method

This is the request sent from Member System for order entry.

Function Prototype:

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

Parameters:

pInputOrder: points to the address for order entry structure. The structure:

```
struct CApexFtdcInputOrderField {
    ///Order No.; this field will be returned by trading system.
    TApexFtdcOrderSysIDType OrderSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Conditions of order price; only supports "price limit".
    TApexFtdcOrderPriceTypeType OrderPriceType;
    ///Buy-sell direction
    TApexFtdcDirectionType Direction;
    ///Combination Offset Flag
    TApexFtdcCombOffsetFlagType CombOffsetFlag;
    ///Flag of speculation and hedge in a portfolio; only the first sign
is effective.
    TApexFtdcCombHedgeFlagType CombHedgeFlag;
    ///Price
    TApexFtdcPriceType LimitPrice;
    ///Quantity
    TApexFtdcVolumeType VolumeTotalOriginal;
    ///Type of valid period,  supports "valid on that day" and "Immediate
or cancel"
    TApexFtdcTimeConditionType TimeCondition;
```

```
        ///GTD DATE, not used
        TApexFtdcDateType GTDDate;
        ///Volume type; supports "arbitrary quantity"; also, supports "entire
quality" when the TimeCondition is set to be TC_IOC
        TApexFtdcVolumeConditionType   VolumeCondition;
        ///The Min.volume, not used
        TApexFtdcVolumeType MinVolume;
        ///Trigger conditions; only supports "immediate".
        TApexFtdcContingentConditionType   ContingentCondition;
        ///Stop-loss price, not used
        TApexFtdcPriceType StopPrice;
        ///Reasons for forced closing-out; only supports "unforced closing-
out"
        TApexFtdcForceCloseReasonType  ForceCloseReason;
        ///Local order No.*
        TApexFtdcOrderLocalIDType  OrderLocalID;
        ///Flag of auto-suspension
        TApexFtdcBoolType IsAutoSuspend;
        ///Business unit,not used
        TApexFtdcBusinessUnitType  BusinessUnit;

    };
    * OrderLocalID: local order No. can only be monotonically increased. After
each successful login, the Max.OrderLocalID "MaxOrderLocalID" can be obtained
from the output parameter "CApexFtdcRspUserLoginField" of OnRspUserLogin.
Since trading system compares the size of OrderLocalID through character sting,
the entire space for "TApexFtdcOrderLocalIDType" shall be fully completed when
setting the "OrderLocalID".
```

nRequestID: the request ID for order entry request; it is specified and managed by the user. Within one conversation, this ID cannot be duplicate.

Price limit Order should fill in the following field:

1) ParticipantID, eg. "2008"
2) ClientID, eg. "10000029"
3) UserID, eg. "200801"
4) InstrumentID, eg. "PF1906"
5) OrderPriceType, can only fill in **APEX_FTDC_OPT_LimitPrice**
6) Direction, buy/sell direction, **APEX_FTDC_D_Buy** means buy, **APEX_FTDC_D_Sell** means sell
7) CombHedgeFlag, combine/hedge flag (not used), can only fill in "1", meaning Speculation
8) LimitPrice, price, eg. 3500.00
9) VolumeTotalOriginal, volume, eg. "5" means 5 lots

10) TimeCondition, fill in **APEX_FTDC_TC_IOC** ("immediately traded or cancel") or **APEX_FTDC_TC_GFD** ("valid in day")
11) VolumeCondition, can fill in **APEX_FTDC_VC_AV** (”any volume”)
12) ContingentCondition, can only fill in **APEX_FTDC_CC_Immediately** (immediately")
13) ForceCloseReason, (not used), can only fill in **APEX_FTDC_FCC_NotForceClose** ("not force close")
14) OrderLocalID, eg. "00000025"

Market price order should fill in the following field:

1) ParticipantID, eg. "2008"
2) ClientID, eg. "10000029"
3) UserID, eg. "200801"
4) InstrumentID, eg. "PF1906"
5) OrderPriceType, can only fill in **APEX_FTDC_OPT_AnyPrice**
6) Direction, buy/sell direction, **APEX_FTDC_D_Buy** means buy, **APEX_FTDC_D_Sell** means sell
7) CombHedgeFlag, combine/hedge flag (not used), can only fill in "1", meaning Speculation
8) LimitPrice, price, eg. 3500.00
9) VolumeTotalOriginal, volume, eg. "5" means 5 lots
10) TimeCondition, fill in **APEX_FTDC_TC_IOC** ("immediately traded or cancel")
11) VolumeCondition, can fill in either **APEX_FTDC_VC_AV** ("any volume")
12) ContingentCondition, can only fill in **APEX_FTDC_CC_Immediately** ("immediately")
13) ForceCloseReason, (not used), can only fill in **APEX_FTDC_FCC_NotForceClose** ("not force close")
14) OrderLocalID, eg. "00000025"

Return Value:

- 0, success
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.23 ReqOrderAction Method

This is the request sent by Member System for order action/operation, including order cancellation, order suspension, order activation, and order amendment.

Function Prototype:

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

Parameters:

pOrderAction: points to the address for order action/operation structure. The structure:

```
struct CApexFtdcOrderActionField {
    ///Order No.*
    TApexFtdcOrderSysIDType OrderSysID;
    ///Local order No.*
    TApexFtdcOrderLocalIDType  OrderLocalID;
    ///Flag of order operation; only supports "deletion"
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Price, not used
    TApexFtdcPriceType LimitPrice;
    ///Local No. of operation
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Change in quantity, not used
    TApexFtdcVolumeType VolumeChange;
    ///Business unit, not used
    TApexFtdcBusinessUnitType  BusinessUnit;

};
* OrderSysID and OrderLocalID means that either of the target order to operated can be filled.
* ActionLocalID: local No. of operation can only be monotonically increased. After each successful login, the Max.OrderLocalID "MaxOrderLocalID" can be obtained from the output parameter "CApexFtdcRspUserLoginField" of OnRspUserLogin. Since trading system compares the size of OrderLocalID through character sting, the entire space for "TApexFtdcOrderLocalIDType" shall be fully completed when setting the "OrderLocalID".
```

nRequestID: the user request ID; it is specified and managed by the user.

Return Value:

- 0, successful
- -1,network connection failure

- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of request sent per second exceeds the allowable quantity.

6.2.24 ReqQuoteInsert Method

Not available in the current version.

This method is used by Member system to send the quote entry request.

Function prototype

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

Parameters

pInputQuote: Address pointing to the quote entry structure. The quote entry structure:

```
struct CApexFtdcInputQuoteField {
    ///Quote No.; this field will be returned from trading system.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Bid Volume
    TApexFtdcVolumeType BidVolume;
    ///Ask Volume
    TApexFtdcVolumeType AskVolume;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Local quote No.*
    TApexFtdcQuoteLocalIDType  QuoteLocalID;
    ///Business unit, not used
    TApexFtdcBusinessUnitType  BusinessUnit;
    ///Flag of position opening and closing-out in buyer's portfolio
    TApexFtdcCombOffsetFlagType BidCombOffsetFlag;
    ///Flag of hedge in buyer's portfolio
    TApexFtdcCombHedgeFlagType BidCombHedgeFlag;
    ///Buyer's price
    TApexFtdcPriceType BidPrice;
    ///Flag of position opening and closing-out in seller's portfolio
    TApexFtdcCombOffsetFlagType AskCombOffsetFlag;
```

```
        ///Flag of hedge in seller's portfolio
        TApexFtdcCombHedgeFlagType AskCombHedgeFlag;
        ///Seller's price
        TApexFtdcPriceType AskPrice;

    };
```

nRequestID: ID for user's quote request. This ID will be designated and managed by user.

Return value

- 0: success.
- -1: the network connection failure;
- -2: indicates that the unprocessed requests exceed the allowable quantity;
- -3: indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.25 ReqQuoteAction Method

Not available in the current version.

By using this method, Member System sends the request for quote operation, including cancellation of order, suspension of quote, activation of quote and modification to order.

Function prototype:

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

Parameters

pQuoteAction: Address pointing to the quote operation structure. The quote operation structure:

```
struct CApexFtdcQuoteActionField {
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Local quote No.
    TApexFtdcOrderLocalIDType  QuoteLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagType ActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
```

```
    TApexFtdcUserIDType UserID;
    /// Local No. of operation
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

nRequestID: ID for user's quote operation request. This ID will be designated and managed by user.

Return value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.26 ReqForQuote Method

Not available in the current version.

Member System sends the request for quote.

Function prototype:

```
int ReqForQuote(
    CApexFtdcInputReqForQuoteField *pReqForQuote,
    int nRequestID);
```

Parameters:

pQuoteAction: Address pointing to quote operation structure:

```
struct CApexFtdcInputReqForQuoteField {
    ///Quote ID
    TApexFtdcQuoteSysIDType ReqForQuoteID;
    ///Participant ID
    TApexFtdcParticipantIDType ParticipantID;
    ///Client name
    TApexFtdcClientIDType ClientID;
    ///Instrument/Contract ID
    TApexFtdcInstrumentIDType  InstrumentID;
    ///TradingDay
    TApexFtdcTradingDayType TradingDay;
```

```
    ///Quote Time
    TApexFtdcTimeType  ReqForQuoteTime;
    ///Calendar Date
    TApexFtdcDateType CalendarDate;
};
```

nRequestID: ID for user's quote operation. This ID will be designated and managed by user.

Return value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.27 ReqExecOrderInsert Method

Not available in the current version.

Request for execution of declaration entry.

Function prototype:

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

Parameters:

pInputExecOrder: Address pointing to execution declaration structure. Execution declaration structure:

```
struct CApexFtdcInputExecOrderField {
    ///Contract No.
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Local execution declaration No.
    TApexFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Quantity
    TApexFtdcVolumeType Volume;
    ///Business unit
```

```
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

nRequestID: ID for annoncement entry request. This ID will be designated and managed by user.

Return value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.28 ReqExecOrderAction Method

Not available in the current version.

Request for execution of declaration operation.

Function prototype:

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

Parameters:

pExecOrderAction: Address pointing to structure of execution declaration operation. The structure of execution declaration operation:

```
struct CApexFtdcExecOrderActionField {
    ///Execution declaration No.
    TApexFtdcExecOrderSysIDTypeExecOrderSysID;
    ///Local execution declaration No.
    TApexFtdcOrderLocalIDType  ExecOrderLocalID;
    ///Flag of order operation
    TApexFtdcActionFlagTypeActionFlag;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDTypeUserID;
    ///Local No. of operation
    TApexFtdcOrderLocalIDType  ActionLocalID;
    ///Business unit
```

```
    TApexFtdcBusinessUnitType  BusinessUnit;

};
```

nRequestID: ID for execution declaration operation request. This ID will be designated and managed by user.

Return value:

- 0, represents success
- -1, represents the network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity
- -3, indicates that the number of requests sent per second exceeds the allowable quantity

6.2.29 ReqQryPartAccount Method

This is the request for member fund/cash query. All those incomplete query requests after timeout will be removed (same for below other query methods).

Function Prototype:

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

Parameters:

pQryPartAccount: points to the address for member cash/fund query structure. The structure:

```
struct CApexFtdcQryPartAccountField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///Capital account, optional
    TApexFtdcAccountIDType AccountID;
};
```

nRequestID: User request ID; this ID is specified and managed by user.

Return Value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;

- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.30 ReqQryOrder Method

This is the order query request.

Function Prototype:

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

Parameters:

pQryOrder: points to the address for order query structure. The query conditions are related. If an optional query condition is empty, that query condition is ignored. The structure:

```
struct CApexFtdcQryOrderField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///Order No., optional
    TApexFtdcOrderSysIDType OrderSysID;
    ///Contract code, optional
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Client code, optional
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code, optional
    TApexFtdcUserIDType UserID;
    ///The starting time, optional
    TApexFtdcTimeType TimeStart;
    ///The finishing time,optional
    TApexFtdcTimeType TimeEnd;
};
```

nRequestID: user's order query request ID; this is specied and managed by user.

Return Value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.31 ReqQryQuote Method

Not available in the current version.

Quote query request.

Function prototype:

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

Parameters:

pQryQuote: Address pointing to quote query structue. Quote query structure:

```
struct CApexFtdcQryQuoteField {
    ///The starting member code
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code
    TApexFtdcParticipantIDType PartIDEnd;
    ///Quote No.
    TApexFtdcQuoteSysIDType QuoteSysID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
};
```

nRequestID: ID for user's quote query request. This ID will be designated and managed by user.

Return value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.32 ReqQryTrade Method

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

Function Prototype:

```
int ReqQryTrade(
```

```
        CApexFtdcQryTradeField *pQryTrade,
        int nRequestID);
```

Parameters:

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

```
struct CApexFtdcQryTradeField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///The starting contract code, optional
    TApexFtdcInstrumentIDType  InstIDStart;
    ///The ending contract code, optional
    TApexFtdcInstrumentIDType  InstIDEnd;
    ///Transaction No. ,optional
    TApexFtdcTradeIDType   TradeID;
    ///Client code,optional
    TApexFtdcClientIDType  ClientID;
    ///Transaction user's code,optional
    TApexFtdcUserIDType UserID;
    ///The starting time,optional
    TApexFtdcTimeType TimeStart;
    ///The finishing time,optional
    TApexFtdcTimeType  TimeEnd;
};
```

nRequestID: user trade query request ID; this is specified and managed by user.

Return Value:

- 0, represents successful.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.33 ReqQryClient Method

This is the member client query request.

Function Prototype:

```
int ReqQryClient(
    CApexFtdcQryClientField *pQryClient,
```

```
        int nRequestID);
```

Parameters:

pQryClient: points to the address for client query structure. The structure:

```
struct CApexFtdcQryClientField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///The starting client code, optional
    TApexFtdcClientIDType  ClientIDStart;
    ///The ending client code, optional
    TApexFtdcClientIDType ClientIDEnd;
};
```

nRequestID: user client query request ID; it is specified and managed by user.

Return Value:

- 0, represents successful.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.34 ReqQryPartPosition Method

This is the member position query request.

Function Prototype:

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

Parameters:

pQryPartPosition: points to the address for member position query structure. The structure:

```
struct CApexFtdcQryPartPositionField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///The starting contract code, optional
```

```
    TApexFtdcInstrumentIDType  InstIDStart;
    ///The ending contract code, optional
    TApexFtdcInstrumentIDType  InstIDEnd;
};
```

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

Return Value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.35 ReqQryClientPosition Method

This is the client position query request.

Function Prototype:

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

Parameters:

pQryClientPosition: points to the address for client position query structure. The structure:

```
struct CApexFtdcQryClientPositionField {
    ///The starting member code can only represent this member
    TApexFtdcParticipantIDType PartIDStart;
    ///The ending member code can only represent this member
    TApexFtdcParticipantIDType PartIDEnd;
    ///The starting client code, optional
    TApexFtdcClientIDType  ClientIDStart;
    ///The ending client code, optional
    TApexFtdcClientIDType ClientIDEnd;
    ///The starting contract code, optional
    TApexFtdcInstrumentIDType  InstIDStart;
    ///The ending contract code, optional
    TApexFtdcInstrumentIDType  InstIDEnd;
    ///Type of client,optional
    TApexFtdcClientTypeType ClientType;
};
```

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

Return Value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.36 ReqQryInstrument Method

The is the instrument/contract query request.**Function Prototype:**

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

Parameters:

pQryInstrument: pointer to the address for instrument/contract query structure. The structure:

```
struct CApexFtdcQryInstrumentField {
    ///Settlement group's code,optional
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///Product suite's code,optional
    TApexFtdcProductGroupIDType ProductGroupID;
    ///Product code,optional
    TApexFtdcProductIDType ProductID;
    ///Contract code ,optional
    TApexFtdcInstrumentIDType  InstrumentID;
};
```

nRequestID: instrument/contract query request ID; this is specified and managed by user.

Return Value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.37 ReqQryInstrumentStatus Method

This is the instrument/contract status query request.

Function Prototype:

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

Parameters:

pQryInstrumentStatus: points to the address for instrument/contract trading status query structure. The structure:

```
struct CApexFtdcQryInstrumentStatusField {
    ///The starting contract code, optional
    TApexFtdcInstrumentIDType  InstIDStart;
    ///The ending contract code, optional
    TApexFtdcInstrumentIDType  InstIDEnd;
};
```

nRequestID: instrument/contract trading status query request ID, specified and managed by user.

Return Value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.38 ReqQryCombinationLeg Method

This is the leg instrument query request.

Function Prototype:

```
int  ReqQryCombinationLeg(
    CApexFtdcQryCombinationLegField * pQryCombinationLeg,
    int nRequestID);
```

Parameters:

pQryCombinationLeg: points to the address for leg instrument query structure. The structure:

```
struct CApexFtdcQryCombinationLegField {
    ///Settlement Group ID, optional
```

```
    TApexFtdcSettlementGroupIDType SettlementGroupID;
    ///CombInstrument ID, optional
    TApexFtdcInstrumentIDType InstrumentID;
};
```

nRequestID: the request ID specified and managed by user.

Return Value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.39 ReqQryMarketData Method

Request sent by Member System for general market data query.

Function Prototype:

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

Parameters:

pQryMarketData: points to the address for market data query structure. The structure:

```
struct CApexFtdcQryMarketDataField {
        ///Product code,optional
        TApexFtdcProductIDType ProductID;
        ///Contract code ,optional
        TApexFtdcInstrumentIDType  InstrumentID;
};
```

nRequestID: user query request ID, specified and managed by user.

Return Value:

- 0, successful
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.40 ReqQryBulletin Method

This is the Exchange bulletin query request.

Function Prototype:

```
int ReqQryBulletin(
    CApexFtdcQryBulletinField *pQryBulletin,
    int nRequestID);
```

Parameters:

pQryBulletin: points to the address for Exchange bulletin query structure. The structure:

```
struct CApexFtdcQryBulletinField {
    ///Trading Day, Optional
    TApexFtdcDateType  TradingDay;
    ///market ID, optional
    TApexFtdcMarketIDType MarketID;
    ///bulletin ID, optional
    TApexFtdcBulletinIDType BulletinID;
    ///bulletin type, optional
    TApexFtdcNewsTypeType NewsType;
    ///urgency level, optional
    TApexFtdcNewsUrgencyType   NewsUrgency;
};
```

nRequestID: bulletin query request ID, specified and managed by user.

Return Value:

- 0, successful
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.41 ReqQryMBLMarketData Method

Instrument/Contract price/market data query request.

Function Prototype:

```
int ReqQryMBLMarketData(
    CApexFtdcQryMBLMarketDataField *pQryMBLMarketData,
    int nRequestID);
```

Parameters:

pQryMBLMarketData: points to the address for instrument/contract price/market data query structure. The structure:

```
struct CApexFtdcQryMBLMarketDataField {
    ///starting contract/instrument ID, optional
    TApexFtdcInstrumentIDType InstIDStart;
    /// ending contract/instrument ID, optional
    TApexFtdcInstrumentIDType InstIDEnd;
    ///buy-sell direction, optional
    TApexFtdcDirectionType Direction;
};
```

nRequestID: instrument/contract price/market data query request ID, specified and managed by user.

Return Value:

- 0, success
- -1,network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.42 ReqQryHedgeVolume Method

This is the hedge volume query request.

Function Prototype:

```
int ReqQryHedgeVolume(
    CApexFtdcQryHedgeVolumeField *pQryHedgeVolume,
    int nRequestID);
```

Parameters:

pQryHedgeVolume: points to the address for hedge volume query structure. The structure:

```
struct CApexFtdcQryHedgeVolumeField {
    ///starting member ID, can only be the specific member
    TApexFtdcParticipantIDType PartIDStart;
    ///ending member ID, can only be the specific member
    TApexFtdcParticipantIDType PartIDEnd;
    /// starting client ID, optional
    TApexFtdcClientIDType  ClientIDStart;
    /// ending client ID, optional
```

```
    TApexFtdcClientIDType  ClientIDEnd;
    /// starting contract/instrument ID, optional
    TApexFtdcInstrumentIDType InstIDStart;
    /// ending contract/instrument ID, optional
    TApexFtdcInstrumentIDType InstIDEnd;
};
```

nRequestID: Hedge volume query request ID, specified and managed by user.

Return Value:

- 0, successful
- -1, network connection failure
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.43 ReqCombOrderInsertMethod

Not available in the current version.

By using this method, Member System sends the request for entry of uncommon portfolio.

Function prototype:

```
int ReqCombOrderInsert (
    CApexFtdcInputCombOrderField *pInputCombOrder,
    int nRequestID);
```

Parameters:

pInputCombOrder: Address pointing to structure of entry of uncommon portfolio order. The structure of entry of uncommon portfolio order:

```
struct CApexFtdcInputCombOrderField {
    ///Portfolio order No.
    TApexFtdcOrderSysIDType CombOrderSysID;
    ///Member code
    TApexFtdcParticipantIDType ParticipantID;
    ///Client code
    TApexFtdcClientIDType ClientID;
    ///Transaction user's code
    TApexFtdcUserIDType UserID;
    ///Price
    TApexFtdcPriceType LimitPrice;
    ///Quantity
```

```
        TApexFtdcVolumeType VolumeTotalOriginal;
        ///Local order No.
        TApexFtdcOrderLocalIDType  CombOrderLocalID;
        ///Business unit
        TApexFtdcBusinessUnitType  BusinessUnit;
        ///Contract code 1
        TApexFtdcInstrumentIDType  InstrumentID1;
        ///Buy-sell direction 1
        TApexFtdcDirectionType Direction1;
        ///Separate leg multiplier 1
        TApexFtdcLegMultipleType   LegMultiple1;
        ///Flag of position opening and closing-out 1
        TApexFtdcOffsetFlagType OffsetFlag1;
        ///Flag of speculation and hedge 1
        TApexFtdcHedgeFlagType HedgeFlag1;
        ///Contract code 2
        TApexFtdcInstrumentIDType  InstrumentID2;
        ///Buy-sell direction 2
        TApexFtdcDirectionType Direction2;
        ///Separate leg multiplier 2
        TApexFtdcLegMultipleType   LegMultiple2;
        ///Flag of position opening and closing-out 2
        TApexFtdcOffsetFlagType OffsetFlag2;
        ///Flag of speculation and hedge 2
        TApexFtdcHedgeFlagType HedgeFlag2;
        ///Contract code 3
        TApexFtdcInstrumentIDType  InstrumentID3;
        ///Buy-sell direction 3
        TApexFtdcDirectionType Direction3;
        ///Separate leg multiplier 3
        TApexFtdcLegMultipleType   LegMultiple3;
        ///Flag of position opening and closing-out 3
        TApexFtdcOffsetFlagType OffsetFlag3;
        ///Flag of speculation and hedge 3
        TApexFtdcHedgeFlagType HedgeFlag3;
        ///Contract code 4
        TApexFtdcInstrumentIDType  InstrumentID4;
        ///Buy-sell direction 4
        TApexFtdcDirectionType Direction4;
        ///Separate leg multiplier 4
        TApexFtdcLegMultipleType   LegMultiple4;
        ///Flag of position opening and closing-out 4
        TApexFtdcOffsetFlagType OffsetFlag4;
        ///Flag of speculation and hedge 4
```

```
    TApexFtdcHedgeFlagType HedgeFlag4;

};
```

nRequestID: ID for request for entry of uncommon porfolio order. This ID will be designated and managed by user.

Return value:

- 0, represents success.
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.44 ReqQryCombOrder Method

Not available in the current version.

This method is used to perform the quote query request.

Function prototype:

```
int ReqQryCombOrder (
    CApexFtdcQryCombOrderField *pQryCombOrder,
    int nRequestID);
```

Parameters:

pQryCombOrder: pointer to CApexFtdcCombOrderField, whose structure is as below:

```
struct CApexFtdcQryCombOrderField {
    ///Participant ID to start with
    TApexFtdcParticipantIDType PartIDStart;
    ///Participant ID as an end
    TApexFtdcParticipantIDType PartIDEnd;
    ///Combined Order System ID
    TApexFtdcOrderSysIDType CombOrderSysID;
    ///Client ID
    TApexFtdcClientIDType ClientID;
    ///User ID
    TApexFtdcUserIDType UserID;
};
```

nRequestID: user's quote query request ID, which should be designated and managed by user.

Return value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.45 ReqAdminOrderInsert Method

This is the request to initialize, adjust or cancel credit. A client's credit cannot be adjusted before the credit is initialized. The amount of the credit adjustment can be positive or negative value. Positive value means add credit, negative value means reduce credit. Cancel credit is equivalent to clear credit.

Function prototype:

```
int ReqAdminOrderInsert(
    CApexFtdcInputAdminOrderField *pInputAdminOrder,
    int nRequuestID);
```

Parameters:

pInputAdminOrder: points to the address for administrator order structure. InstrumentID field of the structure is not required to fill in. The structure:

```
struct CApexFtdcInputAdminOrderField {
    ///Contract code
    TApexFtdcInstrumentIDType  InstrumentID;
    ///administrator's command
    TApexFtdcAdminOrderCommandFlagType AdminOrderCommand;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;
    ///trading member's No
    TApexFtdcParticipantIDType ParticipantID;
    ///Amount
    TApexFtdcMoneyType  Amount;
    ///SettlementGroup ID
    TApexFtdcSettlementGroupIDType SettlementGroupID;
};
```

nRequestID: administrator order request ID, which should be designated and managed by user.

Return value:

- 0, represents success;
- -1, represents the network connection failure;

- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

6.2.46 ReqQryCreditLimit Method

This is the request for credit limit query.

Function prototype:

```
int  ReqQryCreditLimit(
    CApexFtdcQryCreditLimitField *pQryCreaditLimit,
    int nRequestID);
```

Parameters:

pQryCreaditLimit: points to the address for credit limit query structure.

```
struct CApexFtdcQryCreditLimitField {
    ///trading member's No
    TApexFtdcParticipantIDType ParticipantID;
    ///Settlement member's No.
    TApexFtdcParticipantIDType ClearingPartID;
};
```

nRequestID: request ID of user's query for credit limit, which should be designated and managed by user.

Return value:

- 0, represents success;
- -1, represents the network connection failure;
- -2, indicates that the unprocessed requests exceed the allowable quantity;
- -3, indicates that the number of requests sent per second exceeds the allowable quantity.

7. TraderAPI—A Development Example

```
// A simple example that describes the use of CApexFtdcTraderApi and
CApexFtdcTraderSpi interfaces.
// This example shows the process of order entry operation.

#include <stdio.h>
#include <string>
```

```
#include "ApexFtdcTraderApi.h"

class CSimpleHandler : public CApexFtdcTraderSpi {
public:
  CSimpleHandler(CApexFtdcTraderApi *api) : m_pTraderApi(api) {}

  ~CSimpleHandler() {}

  virtual void OnFrontConnected() {

    CApexFtdcReqUserLoginField reqUserLogin{};

    // Get ParticipantID
    printf("participantid:");
    scanf("%s", &m_participantId);
    strcpy(reqUserLogin.ParticipantID, m_participantId);

    // Get UserID
    printf("userid:");
    scanf("%s", &m_userId);
    strcpy(reqUserLogin.UserID, m_userId);

    // Get password
    printf("password:");
    scanf("%s", &reqUserLogin.Password);

    // Send the login request
    m_pTraderApi->ReqUserLogin(&reqUserLogin, 0);
  }

  // This method is called when Member System disconnects. Since the API will
try to reconnect automatically, Member System is not required to do anything.
  virtual void OnFrontDisconnected(int nReason) {
    printf("OnFrontDisconnected.\n");
  }

  // After Member System sent the login request, this method is called to
notify Member System whether the login is successful or not.
  virtual void OnRspUserLogin(CApexFtdcRspUserLoginField *pRspUserLogin,
CApexFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {

    printf("OnRspUserLogin:\n");
    printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
```

```
    printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

    if (pRspInfo->ErrorID != 0) {
      // In case of login failure, Member System is required to perform
error-processing.
      printf("Failed to login, errorcode=%d errormsg=%s requestid=%d
chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
      exit(-1);
    }

    // In case of successful login, send an order entry request.
    CApexFtdcInputOrderField ord = CreateOrder();

    m_pTraderApi->ReqOrderInsert(&ord, 1);
  }

  // Response to order entry
  virtual void OnRspOrderInsert(CApexFtdcInputOrderField *pInputOrder,
CApexFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast) {
    printf("OnRspOrderInsert:\n");
    printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
  };

  // Return on order
  virtual void OnRtnOrder(CApexFtdcOrderField *pOrder) {
    printf("OnRtnOrder:\n");
    printf("OrderSysID=[%s]\n", pOrder->OrderSysID);
  }

  // Response to erroneous user request
  virtual void OnRspError(CApexFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {

    printf("OnRspError:\n");
    printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
    printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

    // Member System is required to perform error-processing
    exit(-1);
  }

  CApexFtdcInputOrderField CreateOrder() {
```

```
    CApexFtdcInputOrderField ord{};

    // Member code
    strcpy(ord.ParticipantID, m_participantId);
    // Client code
    strcpy(ord.ClientID, "003101");
    // Transaction user's code
    strcpy(ord.UserID, m_userId);
    // Contract code
    strcpy(ord.InstrumentID, "PF1906");
    // Conditions of order price
    ord.OrderPriceType = APEX_FTDC_OPT_LimitPrice;
    // Buy-sell direction
    ord.Direction = APEX_FTDC_D_Buy;
    // Flag of position opening and closing-out in a portfolio
    strcpy(ord.CombOffsetFlag, "0");
    // Flag of speculation and hedge in a portfolio
    strcpy(ord.CombHedgeFlag, "1");
    //  Price
    ord.LimitPrice = 540.0;
    // Quantity
    ord.VolumeTotalOriginal = 10;
    // Type of valid period
    ord.TimeCondition = APEX_FTDC_TC_GFD;
    // GTD DATE
    strcpy(ord.GTDDate, "");
    // Volume type
    ord.VolumeCondition = APEX_FTDC_VC_AV;
    // The Min.volume
    ord.MinVolume = 0;
    // Trigger conditions
    ord.ContingentCondition = APEX_FTDC_CC_Immediately;
    // Stop-loss price
    ord.StopPrice = 0;
    // Reasons for forced closing-out
    ord.ForceCloseReason = APEX_FTDC_FCC_NotForceClose;
    // Local order No.
    strcpy(ord.OrderLocalID, "0000000001");
    // Flag of auto-suspension
    ord.IsAutoSuspend = 0;

    return ord;
}
```

```
private:
  CApexFtdcTraderApi *m_pTraderApi;
  TApexFtdcParticipantIDType m_participantId;
  TApexFtdcUserIDType m_userId;
};

int main() {

  // Create a CApexFtdcTraderApi instance
  CApexFtdcTraderApi *pTraderApi =
CApexFtdcTraderApi::CreateFtdcTraderApi("./flow/");

  // Create an event-handling instance
  CSimpleHandler handler(pTraderApi);

  // Register the event-handling instance
  pTraderApi->RegisterSpi(&handler);

  // Subscription of topics
  //   TERT_RESTART: retransmit all messages of the current trading day
  //   TERT_RESUME: retransmit messages by resuming from last transmission
  // TERT_QUICK: only transmit messages after login
  pTraderApi->SubscribePublicTopic(APEX_TERT_RESUME);
  pTraderApi->SubscribeUserTopic(APEX_TERT_RESUME);

  // Set heartbeat timeout
  pTraderApi->SetHeartbeatTimeout(10);

  // Registers the NameServers of Trading System
  char *addresses[] = {
    "tcp://172.16.0.31:17001",
    "tcp://172.16.0.32:17001",
    "tcp://172.16.0.33:17001",
    "tcp://172.16.0.34:17001"
  };

  for (int i = 0; i < 4; i++) {
    pTraderApi->RegisterNameServer(addresses[i]);
  }

  // Member System starts to connect to Trading System
  pTraderApi->Init();
```

```
    // Release the CApexFtdcTraderApi instance
    pTraderApi->Release();

    return 0;
}
```

8. Appendix

8.1 Error Code List—To Translate Upon Request

Error No.	Error message	Reasons for error
1	Not login	Illegal dialog was found in each operation
2	Instrument not found	Contract cannot be found when inserting order, quote, OTC order or executing the declaration
3	Participant not found	Participant cannot be found in each operation
4	Client not found	Client cannot be found in each operation
6	Bad order field	Illegal field value was found on the order when inserting the order (out-of-range of the enumerated value)
7	Bad quote field	Illegal field value was found in the quote when inserting the quote (out-of-range of the enumerated value)
8	Bad order action field	Illegal field value was found in the order operation at the time of order operation (out-of-range of the enumerated value)
9	Bad quote action field	Illegal field value was found in the quote operation at the time of quote operation (out-of-range of the enumerated value)
12	Duplicate order	Local order No. was duplicate when inserting order or non-standard portfolio order.
13	Duplicate quote	Local quote No. was duplicate when inserting quote.
15	Client does not belong to participant	It was fount during each operation that the designated client didn't open an account at the designated participant.
16	IOC order can only apply to continuous trading	Attempt to insert IOC order during continuous trade session.
17	GFA order can only apply to auction trading	Attempt to insert GFA order during non-call-auction session.
18	Market order cannot queue	It was found in inserting market order that time conditions are not IOC
19	Volume constrain can only apply to IOC order	It was found in inserting the order with a quantity restriction of non-arbitrary quantity that time conditions are not IOC
20	GTD order expired	It was found in inserting the GTD order that GTD data had expired.
21	Order volume smaller than minimum quantity	It was found in inserting the order with a Min. number requirement that the Min. number exceeds the number of order.

22	Exchange not in sync	It was found during operation of each business that the Exchange's data is not in the synchronized state.
23	Settlement group not in sync	It was found during operation of each business that the settlement group's data is not in the synchronized state.
24	Order not found	It was found during order operation that order to be operated cannot be found.
25	Quote not found	It was found during quote operation that quote to be operated cannot be found.
26	Invalid action in current status	It was found in inserting the order that the contract's trading status is not the continuous trade, call auction order or call auction balancing At the time of order operation, it was found in activation operation that the contract's trading status is not the continuous trade, call auction order or call auction balancing; As to other operations: It was found in non-administrative user that the contract's trading status is not the continuous trade or call auction order; As for administrative user: It was found in order cancellation or order suspension that the contract's trading status is "closed"; It was found in other operations that the contract's trading status is not the continuous trade or call auction order. When inserting OTC order, it was found that the contract's trading status is not continuous trade.
27	Invalid instrument status shift	It was found in switching the contract’s trading status that this migration doesn't comply with regulations on contract state migration.
28	Order fully traded	It was found during order operation that order has been fulfilled.
29	Order already cancelled	It was found during order operation that order has been cancelled.
31	Not enough client position to close	It was found during each operation that may cause closing out that client's open interest is insufficient.
32	Exceeds client position limit	It was found during each operation that is likely to open a position that it has exceeded client's speculative position.
34	Exceeds participant position limit	It was found during each operation that is likely to open a position that it has exceeded participant’s position limit.
35	Account not found	It was found during each operation that the account shall be used for such operation cannot be found.
36	Insufficient balance	It was found during each operation that there is no sufficient capital in the account.
37	Invalid volume	During order entry, order operation, OTC order entry and order operation, the number of order is not the positive integral multiple as required by the Min. number of order or exceeds the Max. number of order

45	Invalid data group datasync status in initialization	It was found during user login that none of settlement group' data has achieved synchronization.
48	Price must be integral multiple of tick	It was found during each operation that price is not the integral mutiple of the contract's tick size.
49	Price out of upper bound	It was found during each operation that the price is higher than the contract's upward price limit.
50	Price out of lower bound	It was found during each operation that the price is lower than the contract's downward price limit.
51	No trading right	It was found during each operation that member is not authorized to trade in the designated contract, or client is not authorized to trade in the designated contract, or trader is not authorized to trade.
52	Close only	It was found during each operation that may result in an opening of position that member only has the right to close out the designated contract, or client only has the right close out the designated contract, or trader only has the right close out a position.
53	Invalid trading role	It was found in inserting the order, inserting the OTC order or inserting portfolio order that on the designated contract, this member doesn't have the trading role corresponding to such client.
57	Cannot operate for other participant	It was found during each operation that user conduct operation on behalf participant to whom he is not subordinate.
58	User mismatch	It was found during each operation that user for operation doesn't match with user for dialog.
59	Duplicated user login	It was found during user's login that this user has already logged into the system.
60	Invalid user or password	It was found during user's login or password modification that username cannot be found or password is incorrect.
62	User not active	It was found during user's login that this user is not active
64	User does not belong to this participant	It was found during user's login that user doesn't belong to the designated member.
65	Invalid login IP address	It was found during user's login that user's IP address is illegal.
67	Not logged in by this user	It was found during user's login that user didn't log in using this user.
66	User not login	It was found during each operation that user hasn’t logged in yet.
68	Not logged in by this participant	It was found during user logout, forced user logout or modification to password that user didn't log in using this participant.
70	Quote cancelled	It was found during quote operation that quote has been cancelled.
76	Order suspended	It was found during suspension of order that order has already been suspended.
77	Order activated	It was found during activation of order that order has already been activate.
78	GTD order date missing	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 has no permission	Use ordinary user to conduct each operation that only can be conducted by administrative user.
83	Stop order can only appliy to continuous trading	Attempt to insert stop-loss order during non-continuous trading session.
84	Stop order must be IOC or GFD	It was found in inserting stop-loss order that time condition is neither IOC nor GFD
89	Bad ExecOrder field	illegal field value was found in execution declaration when inserting declaration(out-of-range of the enumerated value)
90	Bad ExecOrder action field	illegal field value was found in execution declaration operation when operating declaration(out-of-range of the enumerated value)
91	Duplicated ExecOrder	At the time of inserting execution declaration, local execution declaration No. is duplicate.
92	ExecOrder had cancelled	It was found during execution declaration operation that declaration has already been cancelled.
93	ExecOrder not found	It was found during execution declaration operation that to-be-operated declaration cannot be found.
94	ExecOrder only for option	It was found in inserting the execution declaration that the contract is non-option contract.
95	Stop order must have stop price	It was found in inserting stop-loss order that stop-loss price is not specified.
96	Not enough hedge volume	It was found during each operation that is likely to open a position that client's hedge quota is insufficient.
97	Duplicated action	At the time of order operation, quote operation or execution declaration operation, the local operation No. is duplicate.
99	Force close only used by adminstrator	It was found during order operation that the unauthorized user attempt to operate the order inserted by other users of the same member.
100	Invalid user type	It was found during trader's login that the user type is market data user.
103	Cannot close today's position for hedge	Attempt to insert the order for closing out today's position into hedge position.
104	Unknown admin order	Upon the receipt of administration command, the command type cannot be recognized.
106	Duplicated session	When the user login in, and found to have a successful login session is established
107	Not authorized for this function	When the user login in , insert order or other operation, the trading system find the user has no corresponding permission
108	Only clearing member can do this	When initializing, adjust credit, the user is not clearing member
109	Clearing participant does not match	When initializing, adjust credit, the user cannot find clearing member or the corresponding clearing member is not the member.
110	Bad admin order field	When initializing, adjust credit , the command field error
111	Insufficient credit	When user insert an order find the users' credit limit is not enough.
113	Credit limit not initialized	When adjust credit, find the member has not been initialized

114	Best price order cannot queue	It was found in inserting the best price order that time condition is not IOC.
121	No quoting right	Upon the receipt of a request quote command, the user has no market maker quote right.
122	Bad req for quote field	Upon the receipt of a request quote command, the user has no market maker quote right.
123	Req for quote client cannot be empty	Upon the receipt of a request quote command, client field of quote command cannot be empty
124	Req for quote participant cannot be empty	Upon the receipt of a request quote command, member field of quote command cannot be empty
125	Price out of upper price band	It was found during order operation that the order price is higher than price banding .
126	Price out of lower price band	It was found during order operation that the order price is lower than price banding .
127	Market order can only apply to continuous trading	It was found during order operation that the instrument condition is not in “continuous trading phase” and the order price type is not “limit price”
128	Time condition of any price order not correct	It was found during order operation that the any price type order’s time condition is neither IOC nor GFD.
129	Time condition of best price order not correct	It was found during order operation that the best price order’s time condition is neither IOC nor GFD.
130	Time condition of five level price order not correct	It was found during order operation that the five-step price order’s time condition is neither IOC nor GFD.
131	Combined position is not enough.	It was found during force close order operation that when break the combined positions, the amounts of leg positions which waiting for closing is still not enough.
132	Leg position is not enough	It was found during order insert, order action, manual combine that the amounts of leg positions is less than the amount of close, unfrozen and combine.
133	The direction of combine is not support.	It was found during upon the receipt of a request for combine/uncombined command that combined action direction is neither combine nor uncombined .
134	No combine right	Upon the receipt of a request for combine action command, client’s margin type is not “manual strategy” or “manual strategy and big leg” .
135	Combination rule does not exist	Upon the receipt of a request for combine action command, the combination rule does not exist or no single leg exist.

8.2 Enumeration Value List—Translated

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
1	Trading role	ER	TradingRole	Broker	Broker	1
				Proprietary trading	Host	2
				Market maker	MarketMaker	3
2	Transaction user type	UT	UserType	Trader	Trader	1
				Trade manager	TradeManager	2
				Market data provider's user	MDUser	3
				Unauthorized trader	SingleTrader	4
3	Product type	PC	ProductClass	Futures	Futures	1
				Option	Options	2
				Portfolio	Combination	3
				Spot	Spot	4
				EFP	EFP	5
4	Option type	OT	OptionsType	Non-option	NotOptions	0
				Bullish (call)	CallOptions	1
				Bearish (put)	PutOptions	2
5	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
6	Buy-sell direction	D	Direction	Bid	Buy	0
				Ask	Sell	1
7	Type of open interest	PT	PositionType	Net position	Net	1
				Gross position	Gross	2
8	Direction of long and short open interest	PD	PosiDirection	Net	Net	1
				Long	Long	2
				Short	Short	3

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
9	Synchroni-zation state of the Exchange's data	EDS	ExchangeDataSyncStatus	Unsynchronized	Asynchronous	1
				During synchronization	Synchronizing	2
				Synchronized	Synchronized	3
10	Synchroni-zation state of settlemt group's data	SGDS	SGDataSyncStatus	Unsynchronized	Asynchronous	1
				During synchronization	Synchronizing	2
				Synchronized	Synchronized	3
11	Flag of speculation and hedge	HF	HedgeFlag	Speculation	Speculation	1
				Arbitrage	Arbitrage	2
				Hedge	Hedge	3
				Market maker	MarketMaker	4
12	Type of client	CT	ClientType	Natural person	Person	0
				Legal person	Company	1
				Investment fund	Fund	2
13	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
14	Conditions of order price	OPT	OrderPriceType	Arbitrary price	AnyPrice	1
				Price limit	LimitPrice	2
				Best price	BestPrice	3
15	Flag of position opening and closing-out	OF	OffsetFlag	Position opening	Open	0
				Closing-out of position	Close	1
				Forced closing-out	ForceClose	2
				Closing out today's position	CloseToday	3
				Closing out yesterday's position	CloseYesterday	4
16	Reasons for forced closing-out	FCC	ForceCloseReason	Non-forced closing out	NotForceClose	0
				Insufficient fund	LackDeposit	1
				Client exceeded the position limit	ClientOverPositionLimit	2

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
				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
17	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
18	Type of order	ORDT	OrderType	Normal	Normal	0
				Quote derivatives	DeriveFromQuote	1
				Portfolio derivatives	DeriveFromCombination	2
19	Status of OTC order	OOS	OTCOrderStatus	Input by one party	Inputed	0
				Already confirmed	Confirmed	1
				Already cancelled	Canceled	2
				Already rejected	Refused	3
20	Type of valid period	TC	TimeCondition	Immediate or cancel order	IOC	1

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
				Good for this session	GFS	2
				Good for the day	GFD	3
				Good till date	GTD	4
				Good till cancelled	GTC	5
				Good for call auction	GFA	6
21	Volume type	VC	VolumeCondition	Any quantity	AV	1
				The Min. quantity	MV	2
				Total number	CV	3
22	Trigger conditions	CC	ContingentCondition	Immediately	Immediately	1
				Stop-loss	Touch	2
23	Operation flag	AF	ActionFlag	Deletion	Delete	0
				Suspension	Suspend	1
				Activation	Active	2
				Modification	Modify	3
24	Source of order	OSRC	OrderSource	From participants	Participant	0
				From administrator	Administrator	1
25	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
26	Source of transaction price	PSRC	PriceSource	Previous transaction price	LastPrice	0
				Bid price	Buy	1
				Ask price	Sell	2
27	Status of acccount	ACCS	AccountStatus	Status of activation	Enable	0

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
				Stop status	Disable	1
28	Member type	MT	MemberType	Trading member	Trading	0
				Settlement member	Settlement	1
				Comprehensive member	Compositive	2
29	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
30	Administrative order command	AOC	AdminOrderCommandFlag	Position in contract month is not the integral multiple of the forced closing-out position	NotMultipleForceClose	1
				Initialization of trading meber's credit limit	InitCreditLimit	2

Seq. No.	Description of enumeration	Prefix of enumeration	Name of enumeration	Code description	Code Name	Numerical value of code
				Adjustment to trading member's credit limit	AlterCreditLimit	3
				Cancellation of trading member's creadit limit	CancelCreditLimit	4

8.3 Data Type List—Translated

Name of data type	Basic data type	Description of data type
TApexFtdcErrorIDType	int	Error code
TApexFtdcPriorityType	int	Priority
TApexFtdcSettlementIDType	int	Settlement No.
TApexFtdcMonthCountType	int	Number of month
TApexFtdcTradingSegmentSNType	int	No.of trading sessions
TApexFtdcVolumeType	int	Quantity
TApexFtdcTimeSortIDType	int	Sequence No.of queue by time
TApexFtdcFrontIDType	int	Gateway No.
TApexFtdcSessionIDType	int	Dialog No.
TApexFtdcSequenceNoType	int	Sequence No.
TApexFtdcBulletinIDType	int	Bulletin No.
TApexFtdcInformationIDType	int	Information Message
TApexFtdcMillisecType	int	Time (millisecond)
TApexFtdcVolumeMultipleType	int	Contract multiplier
TApexFtdcImplyLevelType	int	Layer of derivatives
TApexFtdcStartPosType	int	Starting position
TApexFtdcAliasType	char[3]	Alias
TApexFtdcOriginalTextType	char[3]	Original text
TApexFtdcParticipantIDType	char[11]	Member code
TApexFtdcParticipantNameType	char[51]	Member name
TApexFtdcParticipantAbbrType	char[9]	Abbreviation of member
TApexFtdcUserIDType	char[16]	Transaction user's code
TApexFtdcPasswordType	char[41]	Password
TApexFtdcClientIDType	char[11]	Client code
TApexFtdcInstrumentIDType	char[31]	Contract code

Name of data type	Basic data type	Description of data type
TApexFtdcProductIDType	char[9]	Product code
TApexFtdcProductNameType	char[21]	Product name
TApexFtdcExchangeIDType	char[9]	The Exchange's code
TApexFtdcDateType	char[9]	Date
TApexFtdcTimeType	char[9]	Time
TApexFtdcInstrumentNameType	char[21]	Contract name
TApexFtdcProductGroupIDType	char[9]	Product suite's code
TApexFtdcProductGroupNameType	char[21]	Name of product suite
TApexFtdcMarketIDType	char[9]	Market code
TApexFtdcSettlementGroupIDType	char[9]	Settlement group's code
TApexFtdcOrderSysIDType	char[13]	Order No.
TApexFtdcOTCOrderSysIDType	char[13]	OTC order No.
TApexFtdcExecOrderSysIDType	char[13]	System No. of execution declaration
TApexFtdcQuoteSysIDType	char[13]	Quote No.
TApexFtdcTradeIDType	char[13]	Transaction No.
TApexFtdcOrderLocalIDType	char[13]	Local order No.
TApexFtdcComeFromType	char[21]	Source of message
TApexFtdcAccountIDType	char[13]	Capital account
TApexFtdcNewsTypeType	char[3]	Bulletin type
TApexFtdcAdvanceMonthType	char[4]	Month in advance
TApexFtdcCommodityIDType	char[9]	Commodity code
TApexFtdcIPAddressType	char[16]	IP address
TApexFtdcProductInfoType	char[41]	Product information
TApexFtdcProtocolInfoType	char[41]	Protocol information
TApexFtdcBusinessUnitType	char[21]	Business unit
TApexFtdcTradingSystemNameType	char[61]	Name of trading system
TApexFtdcTradingRoleType	char	Trading role
TApexFtdcUserTypeType	char	Transaction user's type
TApexFtdcProductClassType	char	Product type
TApexFtdcOptionsTypeType	char	Option type
TApexFtdcInstrumentStatusType	char	Trading status of contract
TApexFtdcDirectionType	char	Buy-sell direction
TApexFtdcPositionTypeType	char	Type of open interest
TApexFtdcPosiDirectionType	char	Direction of long and short open interest
TApexFtdcExchangeDataSyncStatusType	char	Synchronization state of the Exchange's data
TApexFtdcSGDataSyncStatusType	char	Synchronization state of settlement group's data
TApexFtdcHedgeFlagType	char	Flag of speculation and hedge

Name of data type	Basic data type	Description of data type
TApexFtdcClientTypeType	char	Type of client
TApexFtdcInstStatusEnterReasonType	char	Reasons for contract to enter the trading status
TApexFtdcOrderPriceTypeType	char	Conditions of order price
TApexFtdcOffsetFlagType	char	Flag of position opening and closing-out
TApexFtdcForceCloseReasonType	char	Reasons for forced closing-out
TApexFtdcOrderStatusType	char	Status of order
TApexFtdcOrderTypeType	char	Type of order
TApexFtdcOTCOrderStatusType	char	Status of OTC order
TApexFtdcTimeConditionType	char	Type of valid period
TApexFtdcVolumeConditionType	char	Volume type
TApexFtdcContingentConditionType	char	Trigger conditions
TApexFtdcActionFlagType	char	Operation flag
TApexFtdcOrderSourceType	char	Source of order
TApexFtdcTradeTypeType	char	Transaction type
TApexFtdcPriceSourceType	char	Source of transaction price
TApexFtdcAccountStatusType	char	Account status
TApexFtdcMemberTypeType	char	Member type
TApexFtdcExecResultType	char	Execution result
TApexFtdcYearType	int	Year
TApexFtdcMonthType	int	Month
TApexFtdcLegMultipleType	int	Single leg multiplier
TApexFtdcLegIDType	int	Single leg No.
TApexFtdcBoolType	int	Bool type
TApexFtdcUserActiveType	int	Trader's status of activeness
TApexFtdcPriceType	double	Price
TApexFtdcUnderlyingMultipleType	double	Contract multiplier for basic commodity
TApexFtdcCombOffsetFlagType	char[5]	Flag of position opening and closing-out in a portfolio
TApexFtdcCombHedgeFlagType	char[5]	Flag of speculation and hedge in a portfolio
TApexFtdcRatioType	double	Ratio
TApexFtdcMoneyType	double	funds
TApexFtdcLargeVolumeType	double	Large quantity
TApexFtdcNewsUrgencyType	char	Urgency
TApexFtdcSequenceSeriesType	short	Serial No.in sequence
TApexFtdcCommPhaseNoType	short	Communication phase No.
TApexFtdcContentLengthType	int	Length of main text

Name of data type	Basic data type	Description of data type
TApexFtdcErrorMsgType	char[81]	Error message
TApexFtdcAbstractType	char[81]	Message digest
TApexFtdcContentType	char[501]	Message body
TApexFtdcURLLinkType	char[201]	WEB address
TApexFtdcIdentifiedCardNoType	char[51]	Certificate No.
TApexFtdcIdentifiedCardNoV1Type	char[21]	Original certificate No.
TApexFtdcPartyNameType	char[81]	Name of party involved
TApexFtdcIdCardTypeType	char[16]	Type of certificate
TApexFtdcAdminOrderCommandFlagType	char	Administrative order command
TApexFtdcTradingDayType	char[9]	Trading day type
TApexFtdcDataCenterIDType	int	Data center ID.

8.4 Supported Order Types

This table lists all supported combination of “PriceCondition”, “VolumeCondition” and “TimeCondition” flags and how they are handled in APEX Trading Engine:

PriceCondition	VolumeCondition	TimeCondition	Supported	Remarks
LimitPrice	AllVolume	GFD	No	
		IOC	Yes	Typical “FOK” order type
	AnyVolume	GFD	Yes	Typical “GFD” order type
		IOC	Yes	Typical “FAK” order type
	MinVolume	GFD	No	
		IOC	No	
AnyPrice	AllVolume	GFD	No	
		IOC	No	
	AnyVolume	GFD	No	
		IOC	Yes	Typical “Market” order type
	MinVolume	GFD	No	
		IOC	No	

Note:

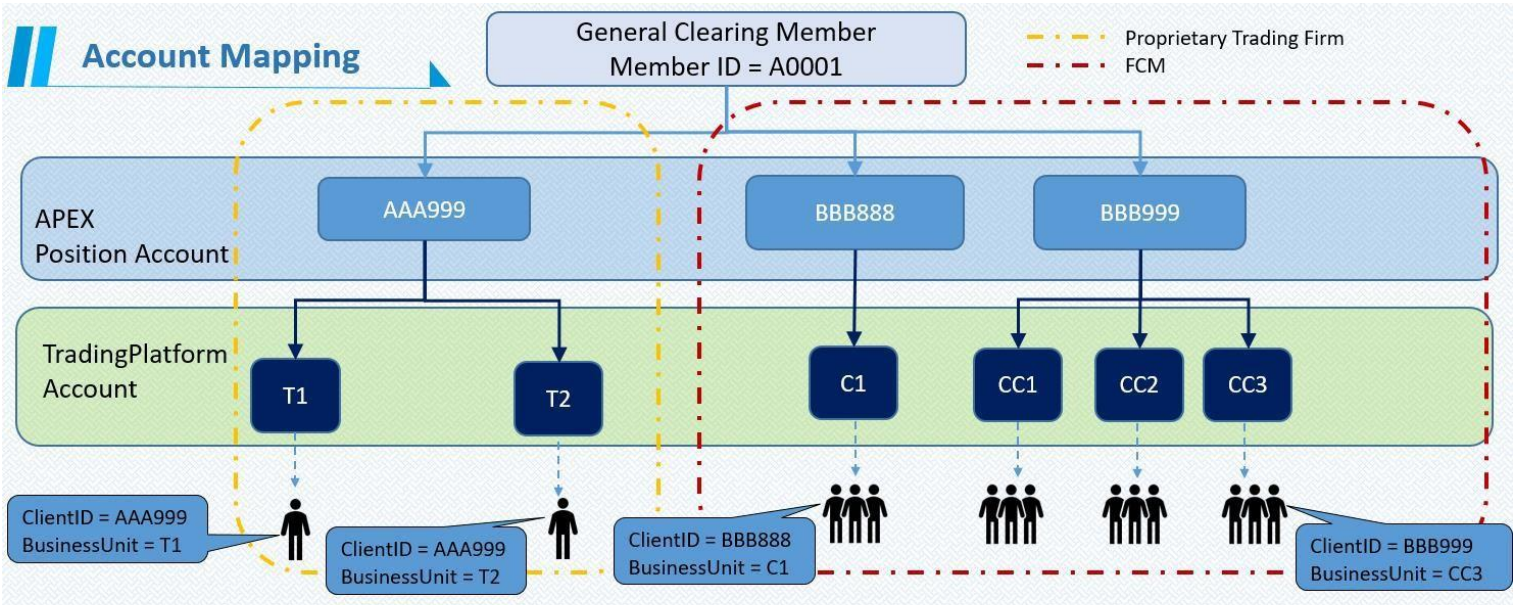
1) The abbreviated terms used in this table are explained as below:
   FOK: Fill or Kill
   FAK: Fill and Kill
   GFD: Good For Day
   IOC: Immediate or Cancel
   GFA: Good For Auction
   GTD: Good Till Date
   GFS: Good For Session
   GTC: Good Till Cancel

2) There are other options for TimeCondition defined in “APEXFtdcUserApiDataType.h” other than “IOC” and “GFD”, including “GFA”, “GTD”, “GFS” and “GTC”, all of which are currently not enabled in APEX Trading Engine.

8.5 Business Unit

“Business Unit” is a free text field attached to each new order. This field can be used to store information like the sub-account of an omnibus account or trader’s information. And the same information will be sent back in order acknledgement. And if the order is traded, the same information will be sent back in Trade Report message.

Below disgram ellustrate a suggested usage to store sub-account information in “Buisness Unit” field:



APEX Trade Report:

Position Account	Status	Report for Date	Trade Date	Trade Time	...	Business Unit
AAA999	Accepted	21/07/2017	21/07/2017	10:11:22	...	T1
AAA999	Accepted	21/07/2017	21/07/2017	14:22:33	...	T2
BBB888	Accepted	21/07/2017	21/07/2017	15:05:03	...	C1
BBB999	Accepted	21/07/2017	21/07/2017	21:03:22	...	CC1
BBB999	Accepted	21/07/2017	21/07/2017	22:05:22	...	CC2
BBB999	Accepted	21/07/2017	21/07/2017	23:01:33	...	CC3