**Market Data User API**

|  |  |
| --- | --- |
|  | **China Financial Futures Exchange** |

**September 1, 2013**

**Table of Contents**

[Chapter 1 Introduction 2](#_Toc8400769)

[1.1 About MduserAPI 2](#_Toc8400770)

[1.2 Platforms for MduserAPI 2](#_Toc8400771)

[1.3 Revision History 2](#_Toc8400772)

[1.3.1 Version 1.5 2](#_Toc8400773)

[Chapter 2 Architecture 3](#_Toc8400774)

[2.1 Communication Mode 3](#_Toc8400775)

[2.2 Data Flow 4](#_Toc8400776)

[Chapter 3 Interface Mode 6](#_Toc8400777)

[3.1 MduserAPI Interface 6](#_Toc8400778)

[3.1.1 Dialogue flow programming interface 6](#_Toc8400779)

[3.1.2 Market data flow programming interface 6](#_Toc8400780)

[Chapter 4 Operating Mode 7](#_Toc8400781)

[4.1 Work Process 7](#_Toc8400782)

[4.1.1 Initialization phase 7](#_Toc8400783)

[4.1.2 Function calling phase 7](#_Toc8400783)

[4.2 Worker Thread 7](#_Toc8400784)

[4.3 Connection to the Exchange's Front-end System 8](#_Toc8400785)

[4.4 Local Files 8](#_Toc8400786)

[4.5 List of Front-end Processors 8](#_Toc8400787)

[Chapter 5 Classification of MduserAPI Interfaces 11](#_Toc8400788)

[5.1 Management Interfaces 11](#_Toc8400789)

[5.2 Business Interfaces 11](#_Toc8400790)

[Chapter 6 MduserAPI Reference Manual 12](#_Toc8400791)

[6.1 CFfexFtdcMduserSpi Interface 12](#_Toc8400792)

[6.1.1 OnFrontConnected method 12](#_Toc8400793)

[6.1.2 OnFrontDisconnected method 12](#_Toc8400794)

[6.1.3 OnHeartBeatWarning method 12](#_Toc8400795)

[6.1.4 OnPackageStart method 12](#_Toc8400796)

[6.1.5 OnPackageEnd method 12](#_Toc8400797)

[6.1.6 OnRspUserLogin method 13](#_Toc8400798)

[6.1.7 OnRspUserLogout method 13](#_Toc8400799)

[6.1.8 OnRspSubscribeTopic method 14](#_Toc8400800)

[6.1.9 OnRspQryTopic method 14](#_Toc8400801)

[6.1.10 OnRtnDepthMarketData method 15](#_Toc8400802)

[6.1.11 OnRspError method 16](#_Toc8400803)

[6.2 CFfexFtdcMduserApi Interface 16](#_Toc8400804)

[6.2.1 CreateFtdcMduserApi method 16](#_Toc8400805)

[6.2.2 GetVersion method 17](#_Toc8400806)

[6.2.3 Release method 17](#_Toc8400807)

[6.2.4 Init method 17](#_Toc8400808)

[6.2.5 Join method 17](#_Toc8400809)

[6.2.6 GetTradingDay method 17](#_Toc8400810)

[6.2.7 RegisterSpi method 17](#_Toc8400811)

[6.2.8 RegisterFront method 17](#_Toc8400812)

[6.2.9 RegisterNameServer method 18](#_Toc8400813)

[6.2.10 SetHeartbeatTimeout method 18](#_Toc8400814)

[6.2.11 SubscribeMarketDataTopic method 18](#_Toc8400815)

[6.2.12 ReqUserLogin method 19](#_Toc8400816)

[6.2.13 ReqUserLogout method 19](#_Toc8400817)

[6.2.14 ReqSubscribeTopic method 20](#_Toc8400818)

[6.2.15 ReqQryTopic method 20](#_Toc8400819)

[Chapter 7 Development Example 22](#_Toc8400820)

Chapter 1 Introduction

1.1 About MduserAPI

The Market Data User API is a C++-based class library that implements the subscription and reception of market data by using and extending the interfaces provided by the class library. This class library contains the following five files:

|  |  |  |
| --- | --- | --- |
| Name | Version | Description |
| CFFEXFtdcMduserApi.h | **V1.59** | Market data interface header file |
| **CFFEXFtdcUserApiStruct.h** | **V1.59** | A header file that defines a set of data types required by the UserAPI |
| **CFFEXFtdcUserApiDataType.h** | **V1.59** | A header file that defines a set of business-related data structures |
| libCFFEXmduserapi.dll | **V1.59** | Dynamic link library binary file |
| libCFFEXmduserapi.lib | **V1.59** | Import library file |
| libCFFEXmduserapi.so.lnx64 | **Y1.59** | Linux dynamic library |

1.2 Windows supports MSVC6.0, MSVC.NET2003 compiler. Multi-threaded compilation option /MT is necessary. The Linux version of the api is based on the Redhat 6.3 version, and the gcc version is 4.4.6, depending on the openssl library. Platforms for MduserAPI

The versions for the following operating system platforms are currently launched:

• IntelX86/WindowsXP: including .h files, .dll files, and .lib files.

• LinuxRedHat6.3: including .h files and .so files.

1.3 Revision History

1.3.1 Version 1.5

This version is modified based on the *Market data User API* ***V****12*. The main changes made are as follows:

● This version provides disaster recovery features:

■ Add [4.10 Disaster Recovery Interface], anexplain the principle of disaster recovery.

■ Since the data center code is added to the login message, the **MdUserAPI** modifies the parameters of the **ReqUserLogin** and **OnRspUserLogin** methods.

● This version provides the function of querying data flow length:

■ When logging into the trading system, the current member private flow length and the trader private flow length will be returned in the response.

■ The **MdUserAPI** adds a description of the **ReqQryTopic** and **RspQryTpoic** methods for querying flow lengths.

● The problems found in previous versions are corrected:

■ A description of the **GetVersion** method is added, as the previous version provided the function without document description.

■ A description of the **RegisterNameServer** method is added, as the previous version provided the function without document description.

● Use limit

■ In order to improve the performance of the **API**, in the **WIN32** environment, the **MduserAPI** randomly uses the **Tcp** port of 39601-39650 during initialization. The **Linux** version does not have such limitation.

**1.3.2** Version **1.59**

This version is modified based on the *Market data User API* ***V****15*. There is no change in the interface. The main changes made are as follows:

● This version modifies the instructions on the version’s operation and compilation.

Chapter 2 Architecture

The Market Data **API** uses the **FTD** protocol built on the **TCP** protocol to communicate with the Exchange's market data publishing server. The market data publishing server is responsible for the generation and release of market data, but does not participate in the trading process. Participating in the trading requires the use of the "Trader **API**".

2.1 Communication Mode

All communications in the **FTD** protocol are based on a certain communication mode. The communication mode is actually the way the communication parties work together.

There are two communication modes involved in the market data publishing:

● Dialogue communication mode

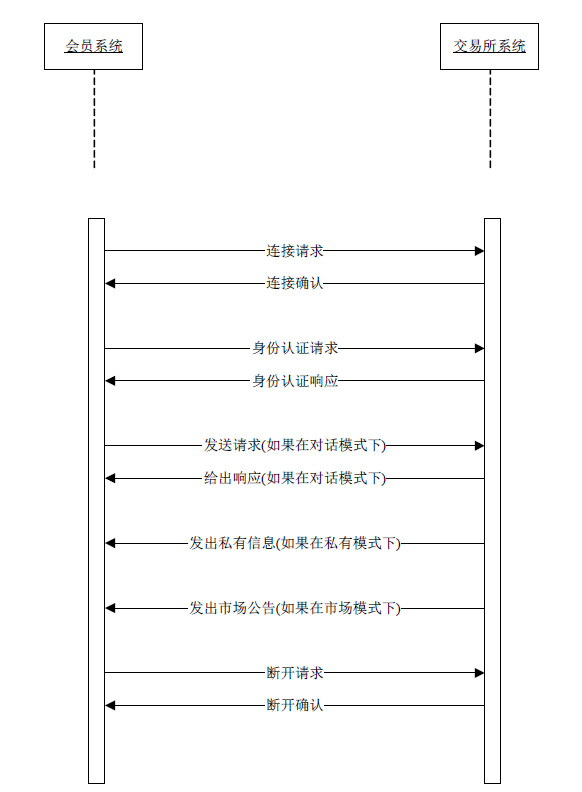
● Broadcast communication mode

The dialogue communication mode means that a communication request is initiated by the member end. The request is received and processed and a response is given by the Exchange. For example, login and logout. This communication mode is the same as the normal user/server mode.

The broadcast communication mode means that the Exchange takes the initiative to send the same information to relevant members in the market. For example, market data.

There is no simple one-to-one relationship in the communication mode and network connection. That is to say, messages of different communication modes may be sent in one network connection, and messages of one communication mode may also be sent in multiple different connections.

Regardless of the communication mode, the communication process is as shown in Figure 1:



Disconnection confirmation

Disconnection request

Send market bulletin (if in market mode)

Send private information (if in private mode)

Give a response (if in dialogue mode)

Send a request (if in dialogue mode)

Identity authentication response

Identity authentication request

Connection confirmation

Connection request

Exchange system

Member system

Figure **1**) Working process of each communication mode

2.2 Data Flow

The market data publishing supports both the dialogue communication mode and the broadcast communication mode:

**1、** **Dialogue communication mode**

Dialogue data flow is supported in the dialogue communication mode:

Dialogue data flow is a two-way mechanism where the member system sends a request, and the market data publishing system gives a response. The trading system does not maintain the status of the dialogue flow. In the event of a system failure, the dialogue data flow will be reset and the data in transit may be lost.

**2、** **Broadcast communication mode**

Market data flow is supported in broadcast communication mode:

Market data flow is a one-way mechanism where the market data is sent by the market data publishing system to the member system; market data flow is a reliable data flow, and the market system maintains the market flow of the entire system. When the member system is disconnected and reconnected within one trading day, the market data system can be requested to send the market data flow data after the specified sequence number.

The market data content provided by the market data service is organized by topics. Each topic includes a set of instrument market data, as well as market data publishing content and publishing methods, including market data depth, sampling frequency, delay time and so on. The Exchange publishes the specific content of each market data topic, and set the market data topic that each user can subscribe to. Each market data topic corresponds to a market data flow.

To get a market data notice, the user must subscribe to one or more market data publishing topics when connecting to the market data server.

Chapter 3 Interface Mode

3.1 MduserAPI Interface

Similar to **TraderAPI**, the **MduserAPI** provides two interfaces: **CFfexFtdcMduserApi** and **CFfexFtdcMduserSpi**. These two interfaces are the encapsulation of the **FTD** protocol.

The market data receiving system can send an action request through CFfexFtdcMduserApi, and handle the response of the trading system by inheriting CFfexFtdcMduserSpi and reloading the callback function

3.1.1 Dialogue flow programming interface

The programming interface for communication through the dialogue flow is usually as follows:

|  |
| --- |
| **////Request:**  int CFfexFtdcMduserApi::ReqXXX(  CFfexFtdcXXXField \*pReqXXX,  int nRequestID)  **intCFfexFtdcMduserApi::ReqXXX(**  **CFfexFtdcXXXField\*pReqXXX,**  **intnRequestlD)**  **////Response:**  void CFfexFtdcMduserSpi::OnRspXXX(  CFfexFtdcXXXField \*pRspXXX,  CFfexFtdcRspInfoField \*pRspInfo,  int nRequestID,  bool bIsLast) **CFfexFtdcXXXField\*pRspXXX,**  **CFfexFtdcRspInfoField\*pRspInfo,**  **intnRequestlD,**  **boolblsLast)** |

The first parameter of the request interface is the request content, which cannot be empty.

The second parameter of the request interface is the request ID. The request ID is maintained by the market data receiving system application, and the request ID of each request is not repeated under normal circumstances. When receiving the response of the trading system, the request ID filled in when the request is sent can be obtained, so that the response can be associated with the request.

The **CFfexFtdcMduserSpi** callback function is called when a trading system response is received. If there is more than one response datum, the callback function will be called multiple times.

The callback function contains a total of four parameters. Among them:

The first parameter is the specific data of the response. If there is an error or no result, it may be **NULL**.

The second parameter is the processing result, indicating whether the processing result of this request is a success or a failure. When multiple callbacks occur, the callbacks may all be NULL except the first one.

The third parameter is the request ID, which is the request ID originally filled in when the request is sent.

The fourth parameter is the response end flag, indicating whether it is the last callback of this response.

3.1.2 Market data flow programming interface

The data in the market data flow contains the market data sent from the trading system.

The programming interface for receiving returns through the market data flow is usually as follows:

**void CFfexFtdcMduserSpi::OnRtnXXX(CFfexFtdcXXXField \*pXXX)；**

|  |
| --- |
| **voidCFfexFtdcMduserSpi::OnRtnXXX(CFfexFtdcXXXF±eld\*pXXX)；** |

The CFfexFtdcMduserSpi callback function is called when the market data is published by the trading system through the market data flow. The parameters of the callback function are the specific content of the notification.

Chapter 4 Operating Mode

4.1 Work Process

The interaction process between the market data receiving system and the trading system is divided into two phases: the initialization phase and the function calling phase.

**4.1.1** **Initialization phase**

In the initialization phase, the program of the market data receiving system must complete the following steps (for specific code, please refer to the development instance):

|  |  |
| --- | --- |
| Sequence | Market data receiving system |
| 1 | Generate an instance of CFfexFtdcMduserApi |
| 2 | Generate an instance of event processing |
| 3 | Register an instance of event processing |
| 4 | Subscribe to the market data flow |
| 5 | Set the network address of the market data service’s NameServer. |
| 6 | Initialize |

1 In order to maintain compatibility with the previous version, this API still provides interfaces to register for market data, but the Exchange does not recommend using such interfaces, which will be removed from the next version. See the list of 0 front-end processors for a description of the NameServer.

4.1.2 Function calling phase

In the function calling phase, the member system can call any of the request methods in the market data receiving system, such as **ReqUserLogin** and **ReqOrderlnsert**, and provide a callback function in response to the return information. Notes:

1. The input parameters of the **API** request cannot be **NULL**.

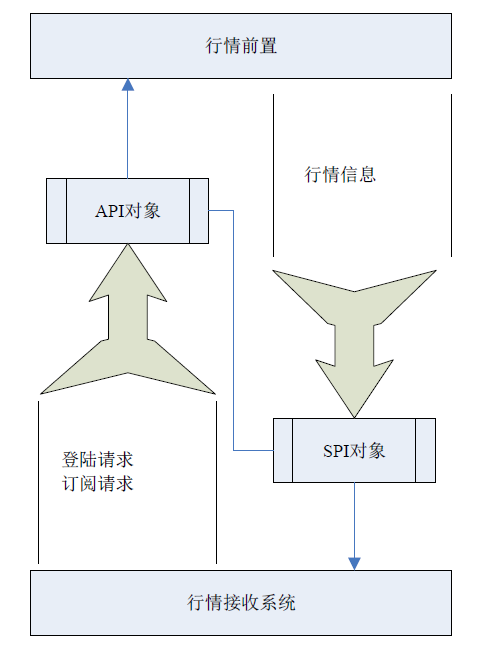
2. Among the returned parameters of the **API** request, 0 indicates correct, others indicate errors. For detailed error coding, please check the table.

4.2 Worker Thread

The market data receiving system application consists of at least two threads: one is the main thread of the application and the other is the worker thread of API. The communication between the application and the market data front-end processor is driven by the worker thread of API.

The interface provided by CFfexFtdcMduserApi is thread-safe. Multiple APP threads can make requests at the same time.

The interface callback provided by CFfexFtdcMduserSpi is driven by the worker thread of MduserAPI. If one of the overloaded callback functions is blocked, it is equal to blocking the worker thread of API, and communication between the API and the trading system will stop. Therefore, in the callback function of CFfexFtdcMduserSpi derived class. Quick return can be achieved by putting data into a buffer or by using the Windows’ messaging mechanism.



SPI object

Market data receiving system

Login request

Subscription request

Market data information

API object

Market data front-end processor

4.3 Connection to the Exchange's Front-end System

The MduserAPI uses the FTD protocol built on the TCP protocol to communicate with the Exchange's market data front-end system. The MduserAPI uses the CFfexFtdcMduserApi::RegisterFront method to register the network communication address of the Exchange's market data front-end system.

The Exchange has multiple market data front-end systems for load balancing and mutual backup, so as to improve the systems’ performance and reliability. To ensure the reliability of communication during trading, the MduserAPI can register multiple front-end processors.

After initialization, the **API** will randomly select a front-end processor from the registered front-end processors and try to establish a network connection. If that fails, it will try other front-end processors one by one until the connection is successfully established. If the network connection fails during trading, the **API** still uses the above process to try to connect to other front-end processors.

The China Financial Futures Exchange will release the network addresses of at least two front-end processors. Therefore, the member system should register at least the network addresses of the two front-end processors to prevent the connected trading front-end processors from failing and causing single point of failure. In view of the high bandwidth requirements of the FTD protocol, it is recommended that members use a DDN line of more than 128K or a 2MSDH digital line. The China Financial Futures Exchange and the Shanghai Futures Exchange will share member remote trading network access: the link directly connected to the China Financial Futures Exchange can be used as a backup link to access the Shanghai Futures Exchange; and vice versa.

The China Financial Futures Exchange will enable **NameServer**. In the future, only the address of **NameServer** will be announced rather than the addresses of front-end processors. The **MdUserAPI** uses the **CFfexFtdcMduserApi**::**RegisterNameServer** method to register the network address of the Exchange’s name server. This function can be called multiple times to register the network addresses of multiple name servers. The **MdUserAPI** will automatically connect to the **NameServer** to get the addresses of a set of front-end processors and then connect to them.

4.4 Local Files

The Trader API will write some data into the local files during the running process. Calling the CreateFtdcTraderApi function may transmit a parameter to indicate the path to the local files. This path must be created before the running. The extension of all the local files is ".con". The local files of the Trader **API** and the Market Data API must be saved in separate directories.

4.5 List of Front-end Processors

The member system can access the **V**1.5 trading system after connecting to the Exchange’s market data front-endprocessors. For fault tolerance and load balancing, the Exchange will deploy two sets of multiple front-end processors (each set comprises multiple units) in the primary and standby data centers. The Exchange will publish a list of the network addresses of the front-end processors. The member system will randomly select one from the list and try to establish a connection with it; the member system is connected to only one front-end processor at a certain time. If the connection is interrupted or times out due to a failure of the front-end processor, the member system needs to try to connect to other front-end processors in the list.

There are two ways in which a member system can get a list of front-end processors:

■ The Exchange publishes the list, and the member system registers the front-end processors into the **API** one by one through the RegisterFront interface of the API.

■ The trading system provides a NameServer with the function of publishing the list of the front-end processors to the **API**. The Exchange first publishes the NameServer list, and the member system registers the NameServer into the API through the **RegisterNameServer** interface of the API. The **API** first tries to get the list of the front-end processors from the NameServer and then tries to connect to a front-end processor based on the list.

The benefits of using NameServer are:

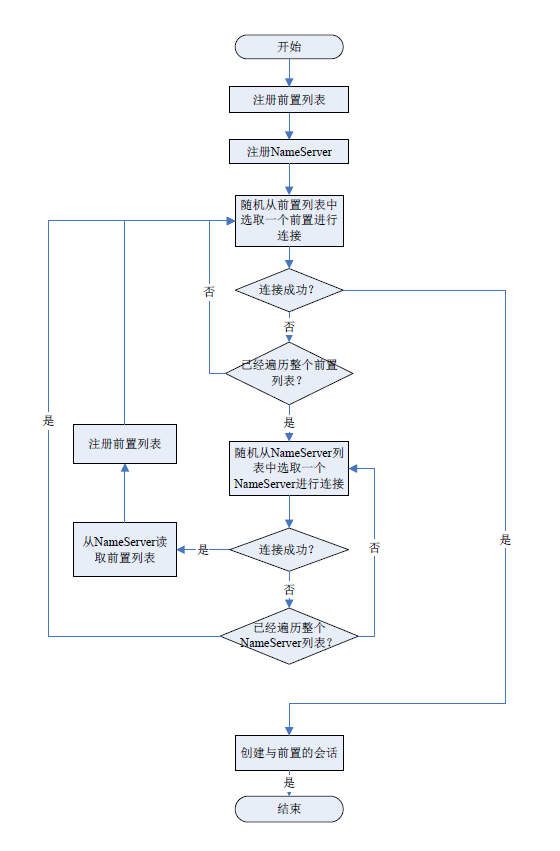
■ It increases the flexibility of the Exchange's front-end processor deployment. Front-end processors can be added in the short term according to business needs and load without any modification to the member system.

■ The NameServer can satisfactorily switch between the primary system and the disaster recovery system.

■ The NameServer features single function, simple structure, and low load. Without the need to consider load balancing, it can be deployed flexibly.

The member system can register the list of front-end processors with the RegisterFront() method and register the NameServer list with the RegisterNameServer() method. The API will first try to connect to the registered front-end processors, and try to connect to the NameServer later if the connection is unsuccessful.

Flowchart of API’s connection to front-end processors:



Start

Register the list of front-end processors

Randomly select a NameServer from the NameServer list for connection

Randomly select a front-end processor from the list of front-end processors for connection

Register the NameServer

Register the list of front-end processors

End

Has the entire NameServer list been traversed?

Connection successful

the entire list of front-end processors been traversed?

No

Connection succeededsuccessful?

Create a conversation with the front-end processor

Yes

Yes

Yes

No

Yes

Yes

No

No

Read the list of front-end processors from the NameServer

Chapter 5 Classification of MduserAPI Interfaces

5.1 Management Interfaces

The management interfaces of the MduserAPI are used to control the API's lifecycle and running parameters.

|  |  |  |
| --- | --- | --- |
| **Interface type** | **Interface name** | **Description** |
| Lifecycle management interfaces | CFfexFtdcMduserApi::CreateFtdcMduserApi | Create an instance of MduserApi |
| CFfexFtdcMduserApi::GetVersion | Get the API version |
| CFfexFtdcMduserApi::Release | Delete the instance of interface |
| CFfexFtdcMduserApi::Init | Initialize |
| CFfexFtdcMduserApi::Join | Wait for the interface thread to finish running |
| Parameter management interfaces | CFfexFtdcMduserApi::RegisterSpi | Register the callback interface |
| CFfexFtdcMduserApi::RegisterFront | Register the network address of the front-end processor |
| CFfexFtdcTraderApi::RegisterNameServer | Register the network address of the NameServer |
| CFfexFtdcMduserApi::SetHeartbeatTimeout | Set the heartbeat timeout |
| Subscription interface | CFfexFtdcMduserApi::SubscribeMarketDataTopic | Subscribe to the market data |
| Communication status interfaces | CFfexFtdcMduserSpi::OnFrontConnected | This method will be called when the communication connection is established with the trading system (before login). |
| CFfexFtdcMduserSpi::OnFrontDisconnected | This method will be called when the communication with the trading system is disconnected. |
| CFfexFtdcMduserSpi::OnHeartBeatWarning | This method will be called when the message has not been received for a long time. |

5.2 Business Interfaces

|  |  |  |  |
| --- | --- | --- | --- |
| **Business type** | **Business** | **Request interface/response interface** | **Data Flow** |
| Login | Login | CFfexFtdcMduserApi::ReqUserLogin  CFfexFtdcMduserSpi::OnRspUserLogin | **N/A** |
| Logout | CFfexFtdcMduserApi::ReqUserLogout  CFfexFtdcMduserSpi::OnRspUserLogout | Dialogue flow |
| Subscription | Subscribe to topics | CFfexFtdcMduserApi::ReqSubscribeTopic  CFfexFtdcMduserSpi::OnRspSubscribeTopic | Dialogue flow |
| Query topic | CFfexFtdcMduserApi::ReqQryTopic  CFfexFtdcMduserSpi::OnRspQryTopic | Query flow |
| Market data | Notice on market data | CFfexFtdcMduserSpi::OnRtnDepthMarketData | Market data flow |

Chapter 6 MduserAPI Reference Manual

The Market data User System API provides two interfaces: CFfexFtdcMduserApi and CFfexFtdcMduserSpi.

6.1 CFfexFtdcMduserSpi Interface

CffexFtdcMduserSpi implements the event notification interface. The user must derive the CffexFtdcMduserSpi interface and write event handling methods to handle events of interest.

6.1.1 OnFrontConnected method

This method will be called when the market data receiving system establishes a TCP virtual link with the market data publishing server.

**Function prototype:**

**void OnFrontConnected()；**

|  |
| --- |
| voidOnFrontConnected()； |

Note: Calling OnFrontConnected only indicates that the TCP connection has succeeded. Subsequent business actions can be performed only after logging in the market data receiving system. The method will not be called back in case of login failure.

6.1.2 OnFrontDisconnected method

This method will be called when the market data receiving system is disconnected from the trading system. When this happens, the API will automatically reconnect and the market data receiving system may take no action. The address for automatic reconnection may be the original registered address, or other available communication addresses supported by the system, which is automatically selected by the program.

**Function prototype:**

**void OnFrontDisconnected (int nReason)；**

|  |
| --- |
| voidOnFrontDisconnected(intnReason)； |

Parameter:

nReason: Reasons for disconnection

■ 0x1001 network reading failed

■ 0x1002 network writing failed

■ 0x2001 heartbeat receiving timeout

■ 0x2002 heartbeat sending failed

■ 0x2003 received error infoinfo

6.1.3 OnHeartBeatWarning method

Heartbeat timeout warning. This method will be called when a message has not been received for a long time.

**Function prototype:**

**void OnHeartBeatWarning(int nTimeLapse)；**

|  |
| --- |
| Void OnHeartBeatWarn±ng(int nTimeLapse)； |

**Parameter:**

nTimeLapse: the time since the last message was received

6.1.4 OnPackageStart method

Notice on message callback start. When the API receives a message, this method will be called first, then each data field is called back, and finally the notice on message callback end is given.

**Function prototype:**

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

|  |
| --- |
| Void OnPackageStart (int nTopicID, int nSequenceNo); |

**Parameter:**

nTopicID: the topic ID (such as private flow, public flow and market data flow).

nSequenceNo: the message sequence number.

6.1.5 OnPackageEnd method

Notice on the end of message callback. When the API receives a message, the notice on message callback start is called, then the each data field is called back, and this method is finally called.

Function prototype:

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

|  |
| --- |
| Void OnPackageEnd (int nTopicID, int nSequenceNo); |

Parameter:

nTopicID: the topic ID (such as private flow, public flow and market data flow).

nSequenceNo: the message sequence number.

6.1.6 OnRspUserLogin method

When the market data receiving system sends a login request and the trading system returns a response, the method is used to notify the market data receiving system whether the login is successful.

**Function prototype:**

**void OnRspUserLogin(**

**CFfexFtdcRspUserLoginField \*pRspUserLogin,**

**CFfexFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

|  |
| --- |
| **voidOnRspUserLogin(**  **CFfexFtdcRspUserLoginField \*pRspUserLogin, CFfexFtdcRspInfoField \*pRspInfo, intn RequestID, bool bIsLast)；** |

Parameter:

**pRspUserLogin**: Return the address of the user login information.

Structure of user login information:

**struct CFfexFtdcRspUserLoginField**

**{**

**/// Trading day**

**TFfexFtdcDateType TradingDay;**

**/// Login time**

**TFfexFtdcTimeType LoginTime;**

**///Maximum order local ID**

**TFfexFtdcOrderLocalIDType MaxOrderLocalID;**

**///User ID**

**TFfexFtdcUserIDType UserID;**

**/// Member ID name**

**TFfexFtdcParticipantIDType ParticipantID;**

**///Trading system name**

**TFfexFtdcTradingSystemNameType TradingSystemName;**

**/// Data center ID**

**TFfexFtdcDataCenterIDType DataCenterID;**

**};Struct CFfexFtdcRspUserLoginField{**

**pRspInfo**: Return the address of the user response information. It should be noted that when there is continuous successful response data, it is possible to return **NULL** in the process rather than at the first time. The same rule applies below. When the error ID is 0, it indicates that the action is successful. The same rule applies below.

Structure of response information:

**struct CFfexFtdcRspInfoField**

**{**

**/// Error ID**

**TFfexFtdcErrorIDType ErrorID;**

**/// Error info**

**TFfexFtdcErrorMsgType ErrorMsg;**

**};**

**nRequestID**: Return the **ID** of the user login request, which is specified by the user when logging in.

**blsLast**: Indicate whether this return is the last return for nRequestID.

6.1.7 OnRspUserLogout method

When the trading system returns a response after the market data receiving system sends a logout request, the method is used to notify the market data receiving system whether the logout is successful.

Function prototype:

|  |
| --- |
| **Void OnRspUserLogout(**  **CFfexFtdcRspUserLogoutField\*pRspUserLogout, CFfexFtdcRspInfoField\*pRspInfo,** |

**void OnRspUserLogout(**

**CFfexFtdcRspUserLogoutField \*pRspUserLogout,**

**CFfexFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

|  |
| --- |
| **intriRequestID,** |
| **boolblsLast);** |

Parameter:

**pRspUserLogout**: Return the address of the user logout information.

Structure of user logout information:

**struct CFfexFtdcRspUserLogoutField**

**{**

**/// User ID**

**TFfexFtdcUserIDType UserID;**

**///Member ID**

**TFfexFtdcParticipantIDType ParticipantID;**

**};**

**pRspInfo**: Return the address of the user response information. Structure of response information:

**struct CFfexFtdcRspInfoField**

**{**

**/// Error ID**

**TFfexFtdcErrorIDType ErrorID;**

**/// Error info**

**TFfexFtdcErrorMsgType ErrorMsg;**

**};**

**nRequestID**: Return the **id** of the user logout request, which is specified by the user when logging out.

**blsLast**: Indicate whether this return is the last return for **nRequestID**.

6.1.8 OnRspSubscribeTopic method

Response to topic subscription. This method will be used when the trading system returns a response after the memberystem sends a topic subscription order.

Function prototype:

**void OnRspSubscribeTopic(**

**CFfexFtdcDisseminationField \*pDissemination,**

**CFfexFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

|  |
| --- |
| **voidOnRspStibscribeTopic(**  **CFfeacFtxicDisseminationField\*pDissemination, CFfexFtdcRspInfoField\*pRspInfo, int nRequestID, bool blsLast**)； |

Parameter:

**pDissemination**: The address directing to the structure of topic subscription, including the topic to be subscribed to and the sequence number of the starting message. Structure of topic subscription:

**struct CFfexFtdcDisseminationField {**

**///**Sequence series

**TFfexFtdcSequenceSeriesType SequenceSeries;**

**///** Sequence number

**TFfexFtdcSequenceNoType SequenceNo;**

**};**

**pRspInfo**: The address directing to the structure of response information. Structure of response information:

**struct CFfexFtdcRspInfoField {**

**///** Error ID

**TFfexFtdcErrorIDType ErrorID;**

**///** Error info

**TFfexFtdcErrorMsgType ErrorMsg;**

**};**

**Possible error：**

**Error ID Error info Possible reason**

66 User has not logged in Not logged in yet

**nRequestID**: Return the **ID** of the topic subscription request, which is specified by the user when subscribing to the topic.

**blsLast**: Indicate whether this return is the last return for **nRequestID**.

6.1.9 OnRspQryTopic method

Response to topic query. The method will be used when the member system sends a topic query order and the trading system returns a response.

Function prototype:

**void OnRspQryTopic (**

**CFfexFtdcDisseminationField \*pDissemination,**

**CFfexFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)；**

|  |
| --- |
| **Void OnRspQryTopic(**  **CFfexFtdcDisseminationField\*pD±ssemination, CFfexFtdcRspInfoField\*pRspInfo, int nRequestID, boolblsLast)；** |

Parameter:

**pDissemination: The address directing to** the structure of the queried topic, including the topic to be queried and the number of the topic messages. Structure of topic query:

**struct CFfexFtdcDisseminationField {**

**/// Sequence series**

**TFfexFtdcSequenceSeriesType SequenceSeries;**

**///Sequence number**

**TFfexFtdcSequenceNoType SequenceNo;**

**};**

**pRspInfo**: The address directing to the structure of response information. Structure of response information:

**struct CFfexFtdcRspInfoField {**

**/// Error ID**

**TFfexFtdcErrorIDType ErrorID;**

**/// Error info**

**TFfexFtdcErrorMsgType ErrorMsg;**

**};**

**Possible error：**

**Error ID Error info Possible reason**

66 User has not logged in Not logged in yet

**nRequestID**: Return the **ID** of the topic query request, which is specified by the user when subscribing to the topic.

**blsLast**: Indicate whether this return is the last return for nRequestID.

6.1.10 OnRtnDepthMarketData method

Notice on market data. When the market data change, the trading system will notify the market data receiving system, and the method will be used.

Function prototype:

|  |
| --- |
| **voidOnRtnDepthMarketData(CFfexEtdcDepthMarketDataField\*pDepthMarketData);** |

Parameter:

**pDepthMarketData**: Return the address of the market data. Note: Some fields in the market data are unused.

Structure of depth market data:

**struct CFfexFtdcDepthMarketDataField**

**{**

**/// Trading day**

**TFfexFtdcDateType TradingDay;**

**/// Settlement group ID**

**TFfexFtdcSettlementGroupIDType SettlementGroupID;**

**/// Settlement ID**

**TFfexFtdcSettlementIDType SettlementID;**

**/// Last price**

**TFfexFtdcPriceType LastPrice;**

**/// Previous settlement price**

**TFfexFtdcPriceType PreSettlementPrice;**

**/// Previous close price**

**TFfexFtdcPriceType PreClosePrice;**

**/// Previous open interest**

**TFfexFtdcLargeVolumeType PreOpenInterest;**

**/// Open price**

**TFfexFtdcPriceType OpenPrice;**

**/// Highest price**

**TFfexFtdcPriceType HighestPrice;**

**/// Lowest price**

**TFfexFtdcPriceType LowestPrice;**

**/// Trading Volume;**

**TFfexFtdcVolumeType Volume;**

**/// Turnover**

**TFfexFtdcMoneyType Turnover;**

**/// Open interest**

**TFfexFtdcLargeVolumeType OpenInterest;**

**/// Close price**

**TFfexFtdcPriceType ClosePrice;**

**/// Settlement price**

**TFfexFtdcPriceType SettlementPrice;**

**/// Upper limit price**

**TFfexFtdcPriceType UpperLimitPrice;**

**///Lower limit price**

**TFfexFtdcPriceType LowerLimitPrice;**

**/// Previous delta**

**TFfexFtdcRatioType PreDelta;**

**/// Current delta**

**TFfexFtdcRatioType CurrDelta;**

**/// Update time**

**TFfexFtdcTimeType UpdateTime;**

**/// Update millisecond**

**TFfexFtdcMillisecType UpdateMillisec;**

**/// Instrument ID**

**TFfexFtdcInstrumentIDType InstrumentID;**

**/// Bid price 1**

**TFfexFtdcPriceType BidPrice1;**

**/// Bid volume 1**

**TFfexFtdcVolumeType BidVolume1;**

**/// Ask price 1**

**TFfexFtdcPriceType AskPrice1;**

**/// Ask volume 1**

**TFfexFtdcVolumeType AskVolume1;**

**/// Bid price 2**

**TFfexFtdcPriceType BidPrice2;**

**/// Bid volume 2**

**TFfexFtdcVolumeType BidVolume2;**

**/// Ask price 2**

**TFfexFtdcPriceType AskPrice2;**

**/// Ask volume 2**

**TFfexFtdcVolumeType AskVolume2;**

**/// Bid price 3**

**TFfexFtdcPriceType BidPrice3;**

**/// Bid volume 3**

**TFfexFtdcVolumeType BidVolume3;**

**/// Ask price 3**

**TFfexFtdcPriceType AskPrice3;**

**/// Ask volume 3**

**TFfexFtdcVolumeType AskVolume3;**

**/// Bid price 4**

**TFfexFtdcPriceType BidPrice4;**

**/// Bid volume 4**

**TFfexFtdcVolumeType BidVolume4;**

**/// Ask price 4**

**TFfexFtdcPriceType AskPrice4;**

**/// Ask volume 4**

**TFfexFtdcVolumeType AskVolume4;**

**/// Bid price 5**

**TFfexFtdcPriceType BidPrice5;**

**/// Bid volume 5**

**TFfexFtdcVolumeType BidVolume5;**

**/// Ask price 5**

**TFfexFtdcPriceType AskPrice5;**

**/// Ask volume 5**

**TFfexFtdcVolumeType AskVolume5;**

**};**

6.1.11 OnRspError method

Notice on error in user request.

Function prototype:

**void OnRspError(**

**CFfexFtdcRspInfoField \*pRspInfo,**

**int nRequestID,**

**bool bIsLast)**

|  |
| --- |
| **voidOnRspError(**  **CFfexFtdcRspInfoField\*pRspInfo, int nRequestID, bool blsLast)** |

Parameter:

**pRspInfo**: Return the address of the user response information. Structure of response information:

**struct CFfexFtdcRspInfoField**

**{**

**/// Error ID**

**TFfexFtdcErrorIDType ErrorID;**

**///** Error info

**TFfexFtdcErrorMsgType ErrorMsg;**

**};**

|  |  |
| --- | --- |
| **TFfexFtdcErrorMsgType** | **ErrorMsg;** |

**nRequestID**: Return the **ID** of the user logout request, which is specified by the user when logging out.

**blsLast**: Indicate whether this return is the last return for **nRequestID**.

6.2 CFfexFtdcMduserApi Interface

The functions provided by the **CFfexFtdcMduserApi** interface to users include: login/logout, market data query, market data subscription and other functions.

6.2.1 CreateFtdcMduserApi method

Generate an instance of **CFfexFtdcMduserApi** but not by **new**.

**Function prototype:**

|  |
| --- |
| **Static CFfexEtdcHduserApi\*CreateFtdcMduserApi(const char\*pszFlowPath="");** |

Parameter:

**pszFlowPath**: A constant character pointer that specifies a file directory to save the status of the market data service publishing message. The default value represents the current directory.

Return value:

Return a pointer directing to an instance of CFfexFtdcMduserApi.

6.2.2 GetVersion method

Get the API version number.

Function prototype:

|  |
| --- |
| **constchar\*GetVersion(int &nMajorVersion, int &nMinorVersion)；** |

Parameter:

nMajorVersion: return the major version number

nMajorVersion: return the minor version number

Return value:

Return a constant pointer directing to the version identifier string.

6.2.3 Release method

Release a CFfexFtdcMduserApi instance. The “Delete” method cannot be used.

Function prototype:

**void Release();**

|  |
| --- |
| **Void Release()；** |

6.2.4 Init method

The market data receiving system establishes a connection with the trading system, and login can be made after the connection is successful.

Function prototype:

**void Init()；**

|  |
| --- |
| **void Init();** |

6.2.5 Join method

The market data receiving system waits for the end of an interface instance thread.

Function prototype:

**void Join()；**

|  |
| --- |
| **Void Join();** |

6.2.6 GetTradingDay method

Get the current trading day. The correct value can be taken only after the connection to the server is established.

Function prototype:

**const char \*GetTradingDay()；**

|  |
| --- |
| **Const char\*GetTradingDay()；** |

Return value:

Return a constant pointer directing to a date information string.

6.2.7 RegisterSpi method

Register an instance derived from the **CFfexFtdcMduserSpi** interface class, which will complete the event handling.

Function prototype:

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

|  |
| --- |
| **Void RegisterSpi(CFfexFtdcMduserSpi\*pSpi);** |

Parameter:

**pSpi**: Implement an instance pointer for the **CFfexFtdcMduserSpi** interface.

6.2.8 RegisterFront method

Set the network communication addresses of the Exchange’s market data front-end systems. The trading system has multiple market data front-end systems, and users can simultaneously register the network communication addresses of multiple market data front-end systems.

This method will be used before the **Init** method.

Function prototype:

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

|  |
| --- |
| **Void RegisterFront(char\*pszFrontAddress);** |

Parameter:

**pszFrontAddress**: The pointer directing to the backend server address. The format of the server address is: "protocol://ipaddress:port". For example, "**tcp**://127.0.0.1:17001". "tcp" represents the transport protocol and "127.0.0.1" represents the server address. "17001" represents the server port number.

6.2.9 RegisterNameServer method

Set the network communication address of the Exchange’s NameServer to obtain the market data service list. The trading system has multiple NameServers, and users can register the network communication addresses of multiple NameServers at the same time. This method will be used before the **Init** method.

Function prototype:

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

|  |
| --- |
| **Void RegisterNameServer(char\*pszNsAddress);** |

Parameter:

**pszNsAddress**: The pointer directing to the network communication address of the Exchange's NameServer. The format of the network address is: "protocol://ipaddress:port". For example, "tcp://127.0.0.1:17001". "tcp" represents the transport protocol and "127.0.0.1" represents the server address. "17001" represents the server port number.

6.2.10 SetHeartbeatTimeout method

Set the timeout period for network communication heartbeat. When the TCP connection between the MduserAPI and the trading system is established, the connection periodically sends a heartbeat to detect whether the connection is normal. This method is used to set the time to detect the heartbeat timeout. **The Exchange recommends that the member system set the timeout between 10 and 30 seconds.**

Function prototype:

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

|  |
| --- |
| **Virtual void SetHeartbeatTimeout (unsigned int timeout);** |

Parameter:

**timeout**: Heartbeat timeout (second). If no information is received from the trading system for more than **timeout**/2 seconds, **CFfexFtdcMduserApi**::**OnHeartBeatWaming**() will be triggered. If no information is received from the trading system for more than **timeout** seconds, the connection will be interrupted and **CFfexFtdcMduserApi**::**OnFrontDisconnected**() will be triggered.

6.2.11 SubscribeMarketDataTopic method

The market data receiving system subscribes to the market data needed. After subscription, the trading system will automatically send a market data notification to the market data receiving system.

Function prototype:

**void SubscribeMarketDataTopic (int nTopicID, TE\_RESUME\_TYPE nResumeType)；**

|  |
| --- |
| **Void SubscribeMarketDataTopic (int nTopicID, TE\_RESUME\_TYPE nResumeType)；** |

Parameter:

**nTopicID**: The topic representing the depth of market data, published by the Exchange.

**nResumeType**: market data retransmission mode

♦ TERT\_RESTART: Start retransmission from the current trading day

♦ TERT\_RESUME: Resume transmission from the last reception

♦ TERT\_QUICK: First transfer the current market data snapshot, and then transfer the contents of the market data after login. The Exchange recommends that members use this method to quickly recover the market data.

6.2.12 ReqUserLogin method

The user sends a login request.

Function prototype:

**int ReqUserLogin(**

**CFfexFtdcReqUserLoginField \*pReqUserLoginField,**

**int nRequestID)；**

|  |
| --- |
| **Int ReqUserLogin (**  **CFfexFtdcReqUserLoginField\*pReqUserLoginLField, intnRequestlD)；** |

Parameter:

**pReqUserLoginField**: The address directing to the structure of user login request.

Structure of user login request:

**struct CFfexFtdcReqUserLoginField**

**{**

**///** Trading day

**TFfexFtdcDateType TradingDay;**

**///** User ID

**TFfexFtdcUserIDType UserID;**

**///** Member ID

**TFfexFtdcParticipantIDType ParticipantID;**

**///** Password

**TFfexFtdcPasswordType Password;**

**///** User product information

**TFfexFtdcProductInfoType UserProductInfo;**

**///** Interface product information, not used

**TFfexFtdcProductInfoType InterfaceProductInfo;**

**///** Protocol information, not used

**TFfexFtdcProtocolInfoType ProtocolInfo;**

**///**Data center ID

**TFfexFtdcDataCenterIDType DataCenterID;**

**};**

**Users need to fill in the UserProductInfo field, which is the product information of the market data receiving system, such as software developer and version number. For example, "FfexMduserV100" represents the market data receiving program and version number developed by China Financial Futures Exchange.**

**For the first login on the current trading day, 0 or the main data center lD published by the Exchange can be filled in the DataCenterlD, and the DataCenterlD returned in the login response can be filled in when logging in again.**

**nRequestID**: The **ID** of the user login request, which is specified and managed by the user.

Return value:

■ 0 represents success.

■ -1 represents that the network connection failed;

■ -2 represents that the number of unprocessed requests exceeds the number permitted;

■ -3 represents that the number of requests sent per second exceeds the number permitted.

6.2.13 ReqUserLogout method

The user sends a logout request.

Function prototype:

**int ReqUserLogout(**

**CFfexFtdcReqUserLogoutField \*pReqUserLogout,**

**int nRequestID)；**

|  |
| --- |
| **Int ReqUserLogout(**  **CFfexFtdcReqUserLogoutField\*pReqUserLogout, intnRequestlD)；** |

Parameter:

**pReqUserLogout**: The address directing to the user logout request structure.

User logout request structure:

**struct CFfexFtdcReqUserLogoutField**

**{**

**/// User ID**

**TFfexFtdcUserIDType UserID;**

**/// Member ID**

**TFfexFtdcParticipantIDType ParticipantID;**

**};**

**nRequestID**: The ID of the user logout request, which is specified by the user and managed.

Return value:

■ 0 represents success.

■ -1 represents that the network connection failed;

■ -2 represents that the number of unprocessed requests exceeds the number permitted;

■ -3 represents that the number of requests sent per second exceeds the number permitted.

6.2.14 ReqSubscribeTopic method

Topic subscription request. It should be used after the login is completed.

Function prototype:

**int ReqSubscribeTopic (**

**CFfexFtdcDisseminationField \* pDissemination,**

**int nRequestID)；**

Parameter:

**pDissemination**: The address directing to the structure of topic subscription, including the topic to be subscribed to and

the sequence number of the starting message. Structure of topic subscription:

**struct CFfexFtdcDisseminationField {**

**/// Sequence series**

**TFfexFtdcSequenceSeriesType SequenceSeries;**

**/// Sequence number**

**TFfexFtdcSequenceNoType SequenceNo;**

**};**

**SequenceSeries：The topic to be subscribed to**

**SequenceNo:=-1 represents using the QUICK method, while other values indicate that retransmission starts from the sequence number.**

**nRequestID**: The **ID** of the user action request, which is specified and managed by the user.

Return value:

■ 0 represents success.

■ -1 represents that the network connection failed;

■ -2 represents that the number of unprocessed requests exceeds the number permitted;

■ -3 indicates that the number of requests sent per second exceeds the number permitted.

6.2.15 ReqQryTopic method

Query topic request. It should be used after the login is completed.

Function prototype:

**int ReqQryTopic (**

**CFfexFtdcDisseminationField \*** **pDissemination,**

**int nRequestID)；**

|  |
| --- |
| **Int ReqQryTopic(**  **CFfexFtdcDisseminationField\*pDissemination, intnRequestID)；** |

Parameter:

**pDissemination**: The address directing to the structure of the queried topic, including the topic to be queried. Topic subscription structure:

**struct CFfexFtdcDisseminationField {**

**/// Sequence series**

**TFfexFtdcSequenceSeriesType SequenceSeries;**

**/// Sequence number**

**TFfexFtdcSequenceNoType SequenceNo;**

**};**

**SequenceSeries：The topic to be queried**

**SequenceNo：no need to fill in**

**nRequestID**: The ID of the user action request, which is specified and managed by the user.

Return value:

■ 0 represents success.

■ -1 represents that the network connection failed;

■ -2 represents that the number of unprocessed requests exceeds the number permitted;

■ -3 indicates that the number of requests sent per second exceeds the number permitted.

Chapter 7 Development Example

|  |
| --- |
| // mdusertest.cpp :  // An example is presented to explain the use of the CFfexFtdcMduserApi and CFfexFtdcMduserSpi interfaces.  #include "stdio.h"  #include "FtdcMduserApi.h"  class CSimpleHandler : public CFfexFtdcMduserSpi  {  public:  //Constructor, a valid pointer to the CFfexFtdcMduserApi instance is needed  CSimpleHandler(CFfexFtdcMduserApi \*pUserApi) : m\_pUserApi(pUserApi) {}  ~CSimpleHandler() {}    **//**When the user establishes a communication connection with the market data publishing server, the user needs to log in.  void OnFrontConnected() {  CFfexFtdcReqUserLoginField reqUserLogin;  strcpy(reqUserLogin.TradingDay, m\_pUserApi->GetTradingDay());  strcpy(reqUserLogin.ParticipantID, "P001");  strcpy(reqUserLogin.UserID, "U001");  strcpy(reqUserLogin.Password, "P001");  m\_pUserApi->ReqUserLogin(&reqUserLogin, 0);  }  // This method will be used when the communication connection between the user and the market data publishing server is disconnected  void OnFrontDisconnected() {  // When this happens, the API will automatically reconnect and the market data receiving system may take no action.  printf("OnFrontDisconnected.\n");  }  **//**When the user makes a login request, the method will be used to notify the user whether the login is successful.  void OnRspUserLogin(CFfexFtdcRspUserLoginField \*pRspUserLogin, CFfexFtdcRspInfoField \*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) {  //// If user login failed, the user needs to handle the error  printf("Failed to login, errorcode=%d errormsg=%s requestid=%d chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);  }  }  / / Notice on deep market data, the market data server will notify the user  void OnRtnDepthMarketData(CffexFtdcDepthMarketDataField \*pMarketData) {  / / The client processes the returned data on demand  }    / / Error notice in user request  void OnRspError(CFfexFtdcRspInfoField \*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);  // Client needs to handle the error  }    private:  // Pointer directing to the CFfexFtdcMduserApi instance  CFfexFtdcMduserApi \*m\_pUserApi;  };  int main()  {  // Generate a CFfexFtdcMduserApi instance  CFfexFtdcMduserApi \*pUserApi = CFfexFtdcMduserApi::CreateFtdcMduserApi();  // Generate an instance of event processing  CSimpleHandler sh(pUserApi);  // Register an instance of event processing  pUserApi->RegisterSpi(&sh);  // Registers the required depth market data topic  /// TERT\_RESTART: Retransmission from this trading day  /// TERT\_RESUME: Resume transmission from the last reception  /// TERT\_QUICK: First transfer the current market data snapshot, and then transfer the contents of the market data after login  pUserApi-> SubscribeMarketDataTopic (102, TERT\_RESUME);  // Set the heartbeat timeout  pUserApi->SetHeartbeatTimeout(19);  // Set the address of the trading system’s market data front-end NameServer  pUserApi->RegisterNameServer("tcp://192.168.1.1:17011");  // Set the address of the market data publishing server  // Make the client establish a connection to the market data publishing server  pUserApi->Init();  // Release the useapi instance  pUserApi->Release();  return 0;  } |