上海期货
信息技术有限公司
Shanghai Futures Information Technology Co.,Ltd.

CTP Client Development Guidance

Improving people's lives with high-technology

Update：2014-11-7
Version：2.0

Preface

This guide is to help new developers work with the CTP's API. It provides an overview of the API, explains it's underlying mechanisms, and outlines the general steps needed to develop a client program. It also answers some of the most-frequent questions developers ask.

This guide summarizes the contents of existing documents and complements new features in the latest version of API. To make the contents of this document rich, accurate, and easy-enough to help developers, we need as many developers as possible to report any errors or omissions, or volunteer anything they think might help improve of this document, by contacting Qiao Yu (if you e-mail, please let me know it's about advice/suggestions for this document in your e-mail subject).

Contact Information

Financial Business Department of SFIT

apiSupport@sfit.shfe.com.cn

Please subscribe the WeChat public account for more real-time information from SFIT:

Table of Contents

1. CTP

1.1. Introduction

Comprehensive Transaction Platform known as CTP is a future brokerage management system developed specially for futures companies. It is composed of trading system, risk management system, and settlement information management system.

The trading system is responsible for order processing, quotation forwarding, and bank-futures transfer. The settlement information management system takes care of transactions, accounts, brokers, funds, rates, daily clearing, information query, reports, etc. The risk management system provides high-speed and real-time calculation to reveal and control risks. CTP can connect four domestic futures exchanges simultaneously and support trading and settlement businesses of domestic commodity futures and stock index futures. And the system can generate and submit margin monitoring and anti-money laundering monitoring files automatically.

The CTP borrows the experience of SHFE's "New Generation Exchange System (NGES)", which represents the advanced level of futures international derivatives trading system. It adopts an innovative distributed architecture, which can precisely replay history and record data at every operation.

Based on memory entirely, the CTP supports 7*24 continuous trading. There is no need for operation staff to stop and start the system manually every day, because "One Button Operation and Maintenance" is realized. This feature makes no increment of operation and maintenance cost, when adding trading centers to expand business.

Currently, only CTP realizes the "One Button Switch" for multiple active trading centers supporting FENS mechanism. In the case of one trading center crashing, the system can switch to a standby trading center immediately, so as to achieve the goal of real continuous trading.

The CTP has exposed and released the trade API, through which users can receive quotations from exchanges and execute trading instructions. The interface uses the way of open interface (API), which has already been a standard in the futures industry.

CTP mini version (CTP mini) is a faster and lighter system. Comparing with the CTP, it pursues more compact configuration and more conservation of resources and equipment. Client programs developed via CTP API are fully compatible with CTP mini system.

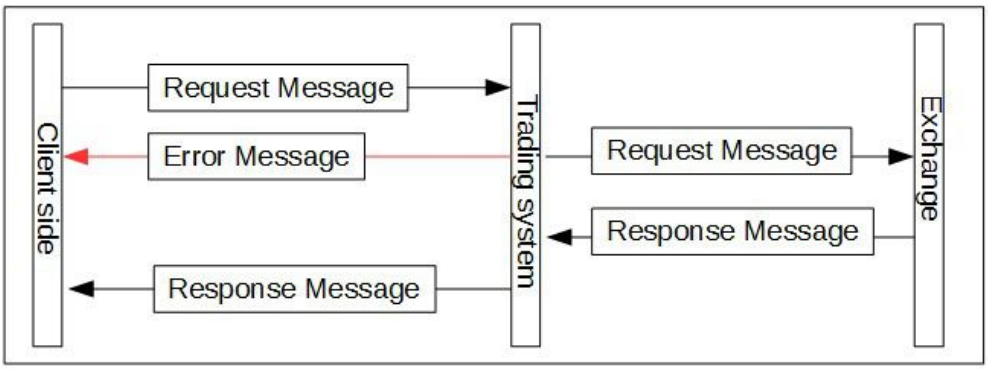
1.2. FTD Communication Protocol

Futures Trading Data Exchange Protocol (FTD) is used for data exchange and communication between futures trading system and its subordinate trading client. This chapter introduces communication modes and data flows in the FTD protocol.

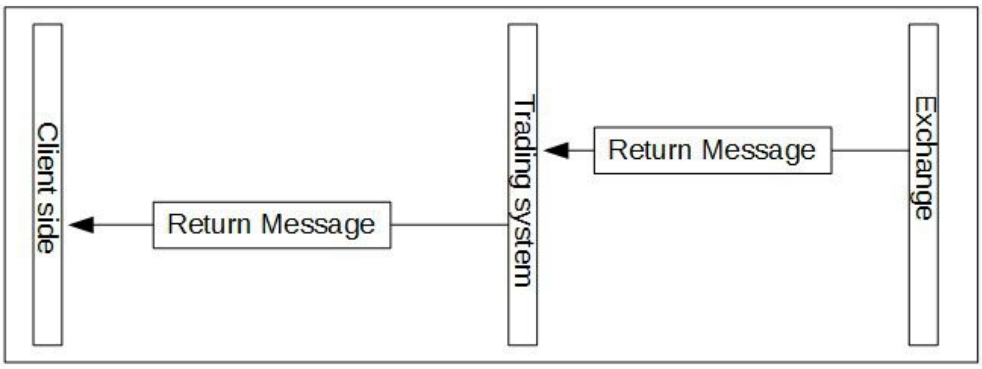
1.2.1. Communication Mode

All communications within the FTD Protocol are based on certain communication mode. Each communication mode is a way for both parties to cooperate with each other. FTD protocol involves the following three communication modes:

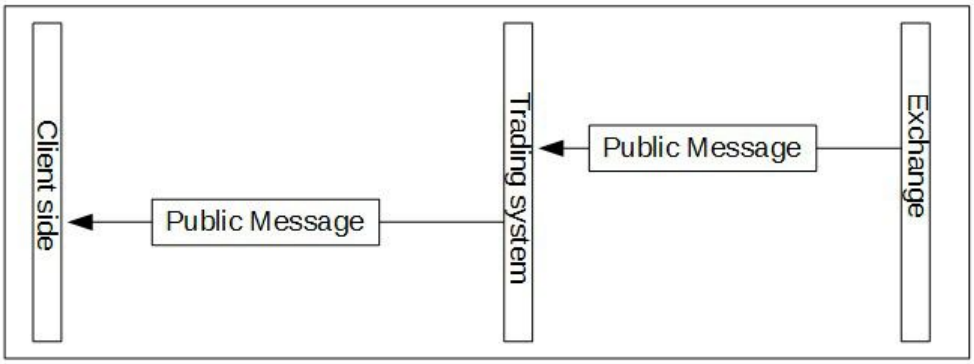
Dialog Communication Mode is that the communication requests are initiated by client programs. The requests (e.g. querying contracts) are received and processed by the trading system and responses are sent back to the client programs. This communication mode is similar to the usual client/server mode.



Private Communication Mode is that the trading system sends information (e.g. trade return) initiatively to a particular client.



Broadcast Communication Mode is that the trading system sends the same information (e.g. quotation data) to all connected clients.



Generally, messages returned in dialog communication mode are called **Response,** while messages returned in private and broadcast communication mode are called **Return**.

1.2.2. Data Stream

In FTD protocol, we need to distinguish between communication mode and data stream. Data stream is a unidirectional or bidirectional, continuous, unique and complete datagrams' sequence, while communication mode is an interactive work mode for data streams. Each data stream corresponds to one communication mode, but one communication mode can have several data streams.

One type of communication mode may be constructed by only one data stream, thus dialog stream, private stream and broadcast stream are produced. And also one type of communication mode may be constructed by several data streams. For example, users can establish query stream and trading stream in dialog communication mode or establish notification stream and quotation stream in broadcast communication mode. FTD only designates which datagram works under which communication mode, but does not specify the division of data streams.

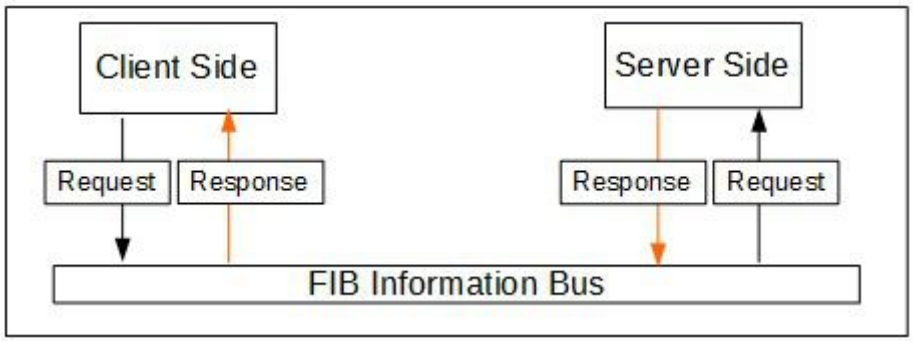
Different communication mode manages data streams in different ways. In dialog communication mode, one data stream is one connection. Messages' integrity and orderliness are maintained in the connection. After disconnected, reconnection will start a new data stream, which is totally different with the original one. If one client sends a request out and disconnects before receiving the response, the response will not be received via the new data stream after reconnection.

For private and broadcast communication modes, one data stream corresponds to all connections of certain feature in one trading day. Unless specified, clients will receive messages from last transmission rather than from beginning after reconnection.

1.3. Two Data Exchange Modes

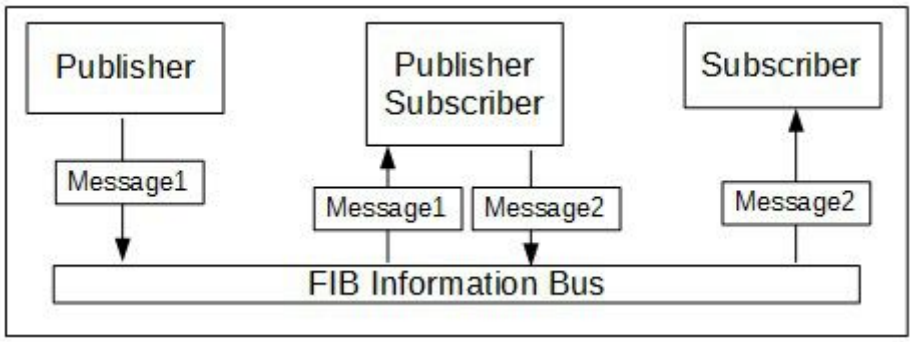
1.3.1. Request/Response Mode

The client program (Client) sends a request to the server (Server). The Server processes the request after receiving it and sends back the result to the Client.

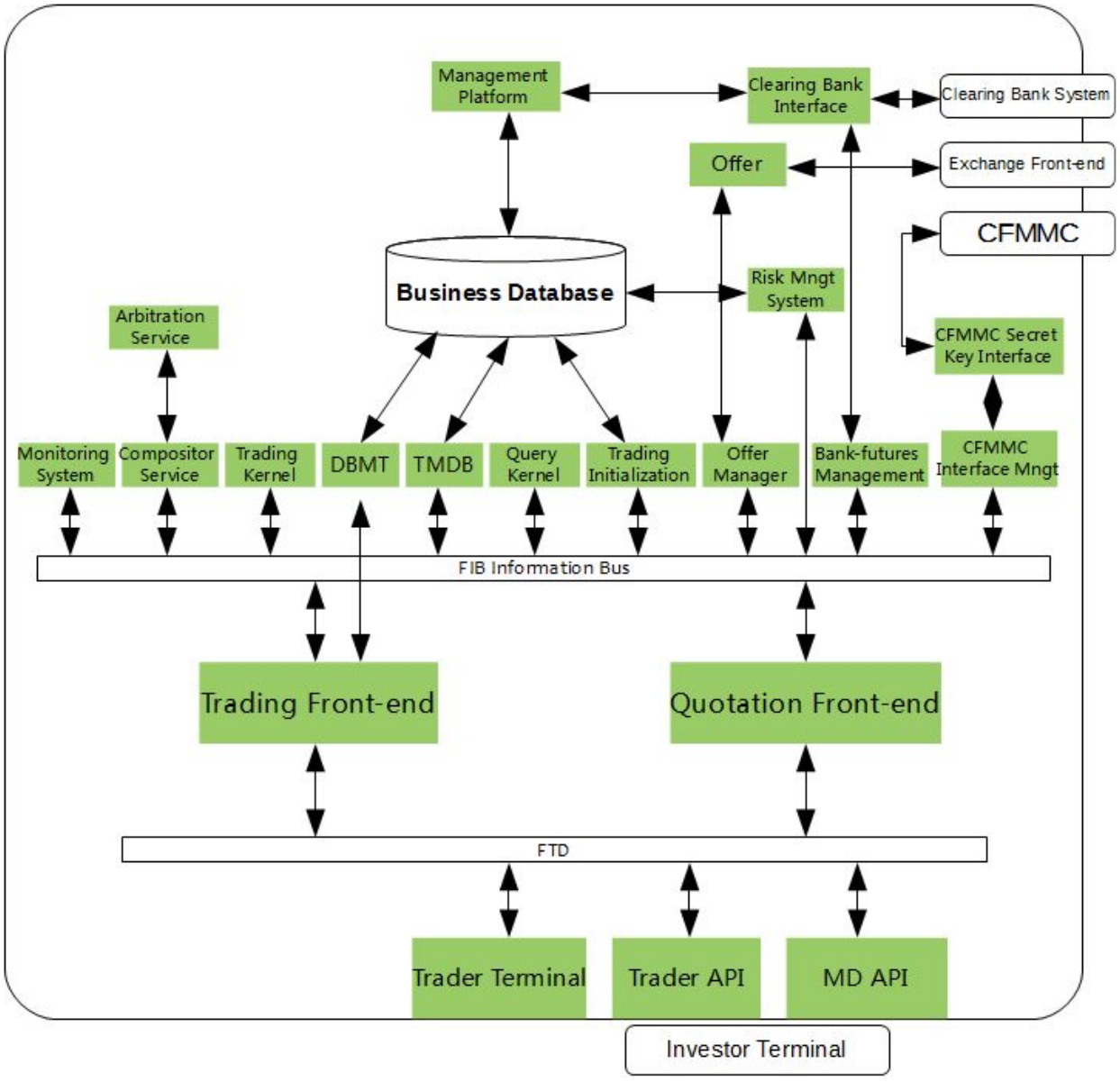


1.3.2. Publish/Subscribe Mode

Publish/subscribe mode is an asynchronous transmission mode. Publishers publish messages to topics. Subscribers subscribe messages from those topics. Publishers and subscribers are relatively independent. They need not connect directly to transmit messages. One FIB application can be both a publisher and a subscriber.



1.4. CTP Architecture



Investor Terminal is a trading client. It implements CTP's trade API and quotation API. It provides functions of receiving quotation, trading, bank-futures transfer, and so on.

Trader Terminal implements CTP's UserAPI. It provides functions of ordering, bank-futures transfer, querying trading data, etc. for futures companies. The UserAPI is facing futures companies and the data range it can control is larger than the TarderAPI. Considering data security and privilege partition, the UserAPI will not be opened to futures companies and investors lately.

FTD is futures trading data exchange protocol.

Trading Front-end connects trading clients via TCP protocol and also connects other background services via FIB. The trading front-end is responsible for communication work and is not related with businesses, so that it

can disperse the pressure, reduce the complexity and improve the safety for trading system. It mainly has three functions: link management, protocol conversion and data routing.

Quotation Front-end subscribes all quotation data from offer manager via FIB and forwards the quotation data to trading clients, which has subscribed the quotations of certain contracts via TCP connections.

FIB (Futures Exchange Information Bus) is trading system's communication infrastructure. It provides data encapsulation, request/response communication mode and publish/subscribe communication mode for upper application level.

Monitoring System obtains part of trading data from the trading system to monitor both the application data and physical components on the capacity, performance, etc.

Arbitration Service manages status switch for compositor service.

Compositor Service is responsible for serializing transaction requests and publishing transaction sequence as the data source for the trading kernel.

Trading Kernel takes care of real-time funds and positions calculation based on investors' positions, orders, trades and funds deposit and withdrawal, so that the pre-trade risk management can be implemented. Meanwhile, it validates all orders, drives exchange offers, and publishes real-time trading result to FIB.

Query Kernel has the same memory database structure and business implementation with the trading kernel. Based on real-time calculation for investors, query kernel updates memory database to provide query service for trading clients via FIB.

DBMT interacts with business database in real time. It sends business data need to be updated to trading kernel via the trading front-end.

TMDB subscribes trading kernel's processing result via FIB. It writes back relative business data into physical database in real time for settlement use after the trading time.

Trading Initialization mainly has two functions: 1. Generates necessary data for trading kernel's initialization based on the database. 2. Sends the trading preparation instruction to system to start a new trading session.

Offer Manager manages trade and quotation offers. It makes the trading kernel not need to process the complex communication between offers and exchange front-ends directly to simplify the trading kernel's processing logic.

Offer implements exchange's quotation API and trade API, places orders via remote trading seat provided by exchanges, receives order return, trade return, quotation data, and so on.

Risk Management System obtains the transaction sequence published by compositor and trading result published by the trading kernel to monitor trading data in real-time, and provides risk trial calculation and forced position liquidation functions as well.

Bank-futures Management is used to manage bank-futures transfer interfaces.

Clearing Bank Interface (Bank-futures Interface) implements bank-futures transfer API from each bank, provides data exchange channel between the trading system and bank systems.

Futures Margin Monitoring Center (CFMMC) Secret Key Interface is used for querying the secret key of futures companies from the Futures Margin Monitoring Center.

Futures Margin Monitoring Center(CFMMC) Interface Management is used for managing secret key interface provided by the Futures Margin Monitoring Center.

Business Database provides the physical data storage data source for the whole system.

Management Platform provides entries of variety business operations for futures companies.

2. API

2.1. Introduction

Our APIs consist of Trading System API, Risk API and Settlement API (CSV). By using the three APIs, users can dock with CTP system to process trading, risk control and saving settlement data.

The Trading System API is used for receiving exchanges' quotation data and sending trading instructions, such as: quotation subscription, placing an order, canceling an order, placing a parked order, futures-bank transfer, information query, etc. Trade API is the most widely used interface. It's target users are mainly client software developers (e.g. Q7) and individual, institutional and proprietary investors having special requirements for trading clients.

The Risk API is used to obtain real-time data (e.g. funds, positions, margins, etc.) to do trial calculation for investors' risks and provide the function of forced position liquidation to decrease risks when necessary. It provides risk enclosure and risk management for risk control personnel in futures companies.

RcWin platform, provided by SFIT for risk control personnel in futures companies, is developed by using the Risk API. Because data managed by the risk API involves all investors' data, the target users for the risk API are futures companies with special requirements for risk management system, but not individual investors.

The Settlement API mainly contains a series of database instructions. It is deployed on the computers, which can access CTP physical database. By executing corresponding instructions, operators can obtain relevant data, such as investors' information, trades, positions, etc. The main target users are also futures companies, not individual investors.

2.1.1. Interface Files

Developers can ask their futures company for the Trader API, or download the official Trade API from our website. Versions of these two may be different.

Official Website Address: *http://www.sfit.com.cn/*

API Files list:

File Name	File Description
ThostFtdcTraderApi.h	Trade interface C++ head file
ThostFtdcMdApi.h	Quotation interface C++ head file

ThostFtdcUserApiStruct.h	Defines all data structures
ThostFtdcUserApiDataType.h	Defines all data types
thosttraderapi.dll	Dynamic link library of Trade interface.
thosttraderapi.lib	
thostmduserapi.dll	Dynamic link library of quotation interface.
thostmduserapi.lib	
error.dtd	API error codes and information in xml format.
error.xml	

2.2. General Rules

2.2.1. Naming Conventions

Rule	Format	Example
Request	Req----	ReqUserLogin
Reply	OnRsp----	OnRspUserLogin
Query	ReqQry----	ReqQryInstrument
Reply of Query	OnRspQry----	OnRspQryInstrument
Return	OnRtn----	OnRtnOrder
Error Return	OnErrRtn----	OnErrRtnOrderInsert

2.2.2. Interface Classes

------Spi (e.g. CThostFtdcTraderSpi) includes all response and return functions. Developers should derive the interface and implement corresponding virtual functions.

------Api (e.g. CThostFtdcTraderApi) contains interfaces for sending requests and subscriptions initiatively. Developers can invoke it directly.

2.2.3. General Parameters

nRequestID: Clients should designate a RequestID when sending the request. The trade API will return the same RequestID in the response or return. When client operates frequently, one response method may be invoked for several times continuously. In this case, requests and responses are matched through the RequestID.

IsLast indicates whether current packet is the last packet with respect to the nRequestID, if a large packet is divided into several packets. In this case, the response method will be invoked for several times. For example, if there is one packet, isLast of the response is true. If two, IsLast of the first response is false, while the second is true.

RspInfo indicates whether there is an error during execution. ErrorId 0 implies that the request is processed by the trading kernel successfully. Otherwise this parameter describes the error message returned by the trading kernel.

All possible values for error messages are described in **error.xml.**

2.2.4. General Initialization Steps for the trade API

Following steps describe the process of creating and initiating trade API and quotation API of the trading system and start the work thread for the trade API.

Currently, the Risk API and Settlement API are not contained in the document.

1. Create “SPI” and “API” instances
   The “SPI” is a user-defined class deriving SPI interfaces (CThostFtdcTraderSpi or CThostFtdcMdSpi). The “API” here is CThostFtdcMdApi or CThostFtdcTraderApi in the interface.
2. Register SPI instance to API instance.
3. Register the front-end network addresses for API. Please register trading front-end addresses for the trade API, whereas quotation front-end addresses for the quotation API.
4. Subscribe public stream (only for trade API) to receive public data, such as trading statuses in the market. By default, data published by exchanges is received from the last disconnection (resume mode). Developers can also designate to receive from beginning (restart mode) or from the login time (quick mode).
5. Subscribe private stream (only for trade API) to receive private data, such as order returns. By default, data published by exchanges is received from the last disconnection (resume mode). Developers can also designate to receive from beginning (restart mode) or from the login time (quick mode).
6. Initialize (Init).

7. Wait for interface thread to terminate (Join).

For example code and more details please refer “Develop a DEMO” chapter.

Develop a Demo

Next part will introduce the steps and key points for using trade API and quotation API of the trading system. The attached code fragment is just for reference.

This document does not explain programming details and does not contain visual interface development guide.

3. Demo of acquiring quotations

The quotation API is easy to use. By using it, developers can quickly learn the basic usage of CTP APIs.

3.1. Preparation

Currently, CTP's trade API has released windows, linux, Android and iOS versions. This guide takes windows version on PC in C/C++ language for example.

Following lists three recommended C/C++ development IDE:

- **Visual Studio**
- **Code Blocks**
- **QT Creator**

Import API Files

Developers should copy all API files described earlier to their project's directory and import all head files and static or dynamic link libraries.

Implement SPI Interface

Developer should derive the **CThostFtdcMdSpi** interface and implement necessary virtual functions.

OnFrontConnected	This function is invoked after establishing the connection to the server but before the logging in to the trading system. Usually it's where client log in the system.
OnFrontDisconnected	This function is invoked after the client disconnects from the server. API will try to reconnect automatically. Parameter of this function contains the reason of the disconnection.

OnRspUserLogin	Return response from log in request. Error message such as “illegal login” usually means wrong password.
OnRspSubMarketData	Return response from request of subscribing market data. Usually it is invoked when error happens to the request of subscribing market data. Reason of error and instrument can be found in the parameters.
OnRspUnSubMarketData	Return response from request of unsubscribing market data.
OnRtnDepthMarketData	Return price quotation with a specific instrument. If no error happens to the request, this function is invoked instead of OnRspSubMarketData
OnRspError	This function is invoked when CTP is not able to identify the request sent by clients. For example, developers might get a notification if they register invalid IP to the front.
OnHeartBeatWarning	This function is invoked if no messages have been received for a preset period of time. To inform client that the connection is alive.

3.2. Initialization for Quotation API

```
1    CThostFtdcMdApi* pUserApi =
2          CThostFtdcMdApi::CreateFtdcMdApi(pathOfLocalFiles, true);
3    QCTPMdSpi* pUserSpi=new QCTPMdSpi(pUserApi);
4    pUserApi->RegisterSpi(pUserSpi);
5    pUserApi->RegisterFront(ipAddressOfmdFront);
6    pUserApi->Init();
7    pUserApi->Join();
```

Line 1&2

Create an instance of **CThostFtdcMdApi** using **CreateFtdcMdApi** method. The first parameter is the directory of local stream files. Stream files are local files generated by quotation API or trade API, postfixed with .con, recording the number of all packets received by the client. The second parameter means the communication mode is TCP (false) or UDP (true).

If developers need to use multicast quotation, they should also set the third parameter **bIsMulticast**. Only when the second and third parameters are both set to be true, the quotation data will be transmitted via multicast.

Note: The multicast quotation can only be used within intranet.

For detailed introduction of flow files, please refer to the “Appendix” chapter.

Line 3

Create a SPI instance. The QCTPMdSpi is a user created entity class deriving **CThostFtdcMdSpi.**

Line 4

Register SPI instance to API instance.

Line 5

Register front-end address to API instance. The front-end address is like tcp://127.0.0.1:17001. The “tcp” is the beginning string, which is not the communication mode and cannot be changed. The “127.0.0.1” is the address for quotation front-end hosted server. The “17001” is the port number of quotation front-end.

Note: By CTP API, quotation data can be transmitted in three ways: TCP, UDP and multicast. One front-end address can use either one of the three ways. It is not necessary to designate a transmission address for each front-end address.

Line 6&7

This is the working thread to initialize the quotation API. After initialization, the thread starts automatically and uses the address registered in the last step to request establishing connection with the server.

All CTP APIs have independent working threads. When developing visual programs, developers should pay attention to avoid threads conflict.

3.3. Login

In previous section, after initialization, quotation working thread will use the registered front-end address to establish a no authenticated connection with the server. After connected, **OnFrontConnected** will be invoked. If not connected or disconnected during running progress, **OnFrontDisconnected** will be invoked.

After connected, the client program can request login using **ReqUserLogin**. The main data structure is **CThostFtdcReqUserLoginField**.

```
CThostFtdcReqUserLoginField req;
memset(&req, 0, sizeof(req));
strcpy(req.BrokerID, BROKERID);
strcpy(req.UserID, USERID);
strcpy(req.Password, PASSWORD);
int ret = pUserApi->ReqUserLogin(&req, ++requestId);
```

BrokerID is the member ID for the futures company.

UserID is the client code for an investor in the futures company.

Password is the investor's password.

An integer value (**ret** in the code fragment) returned by this function indicates the request is sent out successfully or not, rather than the request is processed by the server or not.

Return Value

- 0, represents sending out successfully.
- -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.

Return values of most requests in CTP APIs are the same as described above.

```
void QCTPMdSpi::OnRspUserLogin(
        CThostFtdcRspUserLoginField *pRspUserLogin,
        CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
```

After receiving the login request successfully, the server validates the BrokerID, UserID and Password , and then return the response via **OnRspUserLogin** method.

Developers can judge whether the login is successful via **ErrorID** in the second parameter **pRspInfo**. If ErrorID is 0, the login is successful, otherwise failed.

Generally, the error message "Illegal login" will be returned for password error.

After login successfully, the first parameter **pRspUserLogin** contains some basic data from the server, such as SessionID, frontID, maxOrderRef and the time on each exchange server.

3.4. Subscribe Quotation Data

```
int ret=pUserApi->SubscribeMarketData(arrayOfContracts, sizeOfArray);
```

After login successfully, the client can subscribe quotations now.

The **SubscribeMarketData** is invoked to subscribe quotations. The first parameter is an array of all subscribed contract. The second parameter is the length of this array.

```
void QCTPMdSpi::OnRspSubMarketData(
        CThostFtdcSpecificInstrumentField *pSpecificInstrument,
        CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
```

```
void QCTPMdSpi::OnRtnDepthMarketData(
          CThostFtdcDepthMarketDataField *pDepthMarketData)
```

After subscribing quotation, OnRspSubMarketData or OnRtnDepthMarketData may be invoked.

OnRspSubMarketData

If the subscription request is illegal, error message (pRspInfo) is returned via this method. Even if the request is legal, this function will also be invoked and the returned message is “CTP: No Error”.

OnRtnDepthMarketData

If subscription request is legal, the server will return quotations in the frequency of **twice per second.**

ReqUserLogout and UnSubscribeMarketData are used to logout and unsubscribe quotations. The usage is the same with login and subscribing quotation.

Note: Currently, if logout via ReqUserLogout, the API will disconnect current connection. After re-login, the API will establish a new connection, so the SessionID will be reset and the MaxOrderRef will also start from 0.

3.5. Subscribe and Receive Requests for Quotes

Market makers can subscribe investors' requests for quotes (**RFQ**s) via quotation API.

Currently, exchanges take different ways to implement the functions of investors making RFQs and market makers receiving RFQs. And also they implement variously in processing quotes, RFQs, and other functions. For development efficiency, CTP realizes the functions of investors making RFQs and market makers receiving RFQs in quotation API temporarily.

RFQ Implementation for Four Exchanges

SHFE&CFFEX

The RFQ stream transmits via trade API. And the trade API provides interfaces for investors making RFQs and market makers receiving RFQs. Market makers need not subscribe to receive RFQs. The server authenticates their accounts when market makers login the trade API, and sends corresponding contracts' RFQs according to their privileges in exchanges.

DCE&CZCE

The RFQ stream transmits via quotation API. Investors still make RFQ via trade API, while market makers should subscribe investors' RFQs before the server transmits the RFQ stream.

CTP API insists on being consistent with exchanges. If exchanges improve their implementation for processing RFQs, CTP API will also be adjusted.

API Methods

Quotation API

Request	Response/Return	Description
SubscribeForQuoteRsp	OnRspSubForQuoteRsp	Market makers subscribe investors' RFQs
UnSubscribeForQuoteRsp	OnRspUnSubForQuoteRsp	Market makers unsubscribe investors' RFQs
	OnRtnForQuoteRsp	Market makers receive investors' RFQs

Trade API

Request/Return	Description
OnRtnForQuoteRsp	Market makers receive investors' RFQs
ReqForQuoteInsert	Investors send requests of RFQs

The usage of subscribing RFQs in quotation API is the same with subscribing quotations. In trade API, there is no need to do subscribing.

How to make RFQs for Investors will be described in detail in the next "Demo of Trading" Chapter.

4. Demo of Trading

Comparing with the quotation API, the trade API is more functional and more complex. The trade API provides methods need to be invoked during investors' trading process, e.g. placing an order, canceling an order, bank-futures transfer, information query, etc.

The usage is similar with the quotation API.

4.1. Initialization for Trade API

Just as the quotation API, developers should derive **CThostFtdcTraderSpi** class and implement necessary virtual functions.

```
CThostFtdcTraderApi* pUserTraderApi =
                CThostFtdcTraderApi::CreateFtdcTraderApi(pathOfLocalFiles);
QCTPTradingSpi* pUserTraderSpi = new QCTPTradingSpi(pUserTraderApi);
pUserTraderApi->RegisterSpi(pUserTraderSpi);
pUserTraderApi->RegisterFront(ipAddressOfTradeFront);
pUserTraderApi->SubscribePublicTopic(THOST_TERT_RESTART);
pUserTraderApi->SubscribePrivateTopic(THOST_TERT_RESUME);
pUserTraderApi->Init();
pUserTraderApi->Join();
```

Steps and code are almost the same with quotation API (3.2 section).

Differences:

1. When creating API instance, transmit protocol need not to be designated, so the method in line 2 only needs one parameter (stream files directory).
2. Public stream and private stream need to be subscribed.

Public Stream contains information published by exchanges for all clients, e.g. contracts' trading statuses in the market.
Private Stream contains information sent by exchanges to specified clients, e.g. order return, trade return.

There are three subscription modes:

- Restart: to re-transmit from the first message of the same type sent by exchanges in current trading day.
- Resume: to re-transmit by resuming and continuing from last transmission;
- Quick: to transmit from current login.

3. The registered front-end address is trading front-end address.

4.2. Login

Authentication

Before login into the trading system, if configured, authentication is required. Login can be requested only after a successful authentication.

The authentication can be configured in the settlement platform used by business people in futures companies. It can be configured to be turned off, so that clients need not to be authenticated. Otherwise, futures companies need maintain the authentication code (AuthCode) for client programs in the settlement platform.

The authentication uses ReqAuthenticate (request an authentication) and OnRspAuthenticate (return the authentication response from the server) methods.

Login

Requesting login in trade API is the same with that in quotation API.

After login successfully, the pRspUserLogin parameter in OnRspUserLogin method will contain front ID (FrontID), session ID (SessionID), and the maximum order reference number (MaxOrderRef).

- FrontID is the ID of the front-end which the client connects with.
- SessionID is the session's ID for the connection between the client and the front-end server.
- MaxOrderRef is the maximum number for OrderRef of current session. The OrderRef is a unique number for orders in one session. If the client does not assign a value for OrderRef, server will assign it automatically. If the client assigns a value, the value can be incremented from MaxOrderRef to prevent duplication with other orders.

Note: Currently, if user uses ReqUserLogout to logout, current connection will be disconnected. After re-login, a new connection will be established, the SessionID will be reset and the MaxOrderRef will start from 0.

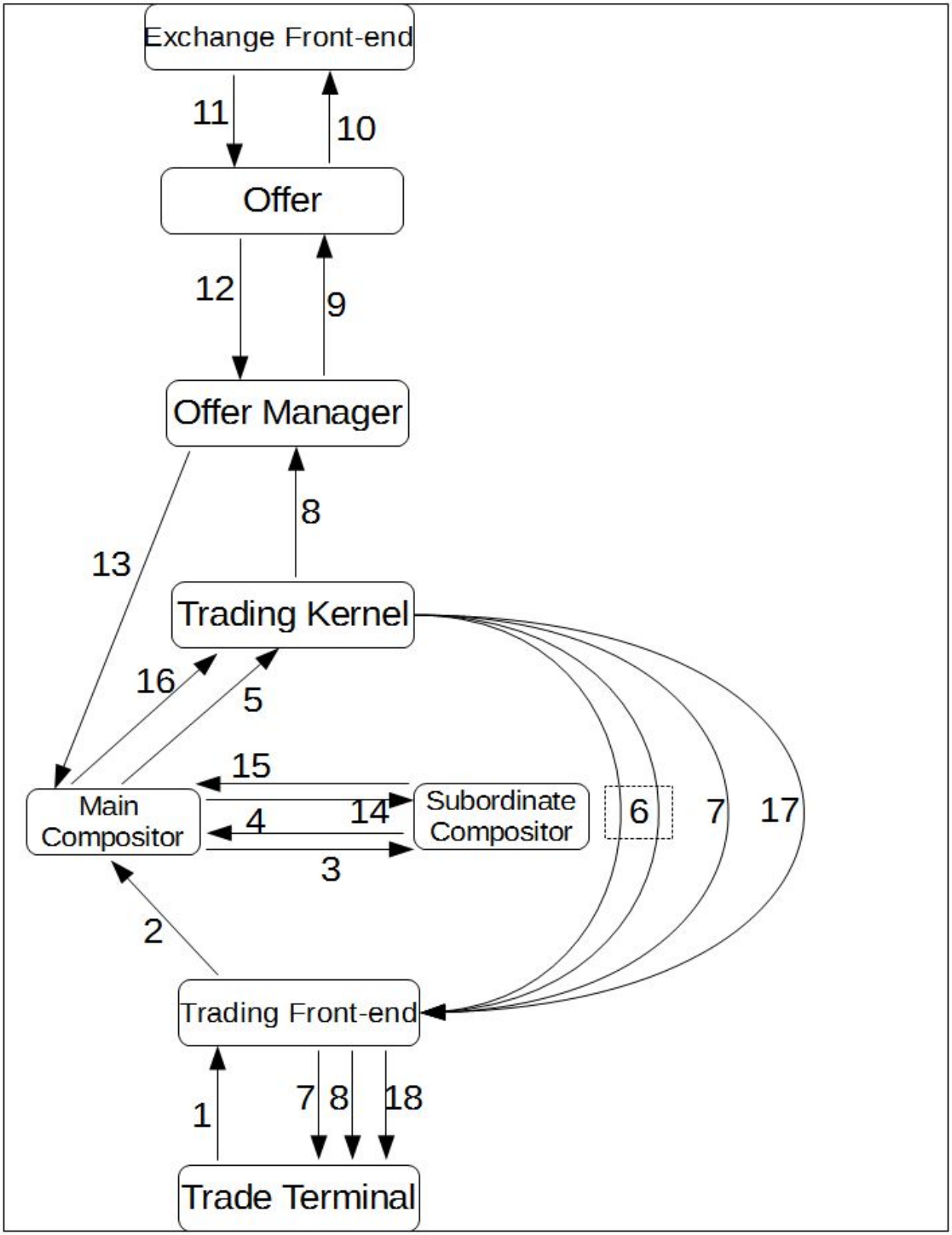
4.3. Confirm Settlement Info

In order to let investors know their trading statuses (e.g. available funds, positions, margin occupancy, etc.) timely and accurately and understand their risk statuses, CTP requires investors confirming the settlement info of the last trading day, when the first time they login in current trading day.

Methods for Settlement Confirmation

Request	Reply	Description
ReqQrySettlementInfo	OnRspQrySettlementInfo	Query settlement info
ReqSettlementInfoConfirm	OnRspSettlementInfoConfirm	Confirm settlement info
ReqQrySettlementInfoConfirm	OnRspQrySettlementInfoConfirm	Query date of confirmation

4.4. Order Processing Flow



1. The Trade Terminal submits order request to the Trading Front-End via trade API.
2. The Main Compositor subscribes transaction request from the Trading Front-End.
3. The Main Compositor sends the transaction sequence to the Subordinate Compositor for confirmation.
4. After receiving the packet to be confirmed, the Subordinate Compositor writes relative packets into the flow file and returns the confirmation message immediately.
5. The trading kernel subscribes transaction sequence from the Main Compositor.
6. The trading kernel validates the transaction sequence after receiving it. If any error, the trading kernel will return an order response with error message to the Trading Front-End. Then the Trading Front-End transfers the message to the Trade Terminal immediately. If the request is a legal transaction request, the trading kernel will also return a response to the Trading Front-End, but this response will not be returned to the Trade Terminal.
7. There are two cases in this step: 1. If the Trading Front-End gets an error order response, it will send the package to the Trade Terminal via dialog mode. 2. If the response message is correct, the Trading Front-end returns the corresponding order return to the Trade Terminal immediately via private mode (step 8, from the Trading Front-End to the Trade Terminal).
8. There are two progresses in this step: 1. After receiving order return from the trading kernel, the Trading Front-End sends the order return to the Trade Terminal via private mode. 2. After sending the first order return to the Trading Front-End, the trading kernel will generate a request package for requesting an order insert and send it to the exchange. The package will be subscribed by the Offer Manager.
9. After subscribing the order insert request, the Offer Manager forwards the package to corresponding offer.
10. After receiving the package from the Offer Manager, the Offer sends the order to the exchange via the exchange's trade API.
11. The Offer receives order return, trade return, or error order response from the exchange's trading front-end via exchange's trade API.
12. The Offer gathers all order returns, trader returns and order responses from the exchange and sends them to the Offer Manager.
13. The Main Compositor subscribes order packages from the Offer Manager.
14. After serializing the packages, the Main Compositor sends the transaction sequence to the Subordinate Compositor for confirmation.
15. After receiving the packets to be confirmed, the Subordinate Compositor writes the packets into the flow file and returns the confirmation messages immediately
16. The trading kernel subscribes all order and trade returns from the Main Compositor.
17. The Trading Front-End subscribes all trading result from the trading kernel.
18. The Trading Front-End distributes the trading result to each Trade Terminal.

4.5. Methods for Processing Orders

ReqOrderInsert：Request an order insert (above step 1).

OnRspOrderInsert：Return an order response containing error message returned by the trading kernel (the first case in step 7).

OnRtnOrder: Return the order status from an exchange. Each time the order status is changed, this method will be invoked once. It may be invoked for several times during one order progress, e.g. when the trading system submits an order to the exchange (the second progress in step 8), when the exchange cancels or accepts the order, when the order is filled, and so on.

OnErrRtnOrderInsert: When an exchange receives the order validated by the trading kernel from an offer, the exchange will validate the order again. If the order is illegal, the exchange will cancel the order, send error message to the offer and return the latest order status. After a client receives the error message, OnErrRtnOrderInsert method will be invoked and the latest order status (canceled) will be returned via OnRtnOrder method. If the order is legal, only the order status (un-triggered) is returned.

OnRtnTrade：If the order is filled, the exchange returns the filled order status and returns the trade information via this method.

After an order is filled, both an order return (OnRtnOrder) and a trade return (OnRtnTrade) will be sent to the client. And the order status in the order return is "filled". But because CTP's trading kernel updates the order status only after receiving the trade return, we recommend clients to use the trade return to judge whether an order is filled or not. For example, if a client receives the order return with "filled" status and sends the closing position instruction immediately, the closing position operation may be failed. Because the trading kernel receives the closing position instruction before the trade return, and the order status is not “filled” yet in the trading kernel.

4.6. Place an Order

ReqOrderInsert method is used for placing an order. The main data structure is CThostFtdcInputOrderField.

General assignment code is as follows:

```
CThostFtdcInputOrderField orderInsert;
memset(&orderInsert, 0, sizeof(orderInsert));
strcpy(orderInsert.BrokerID, BROKERID);
strcpy(orderInsert.InvestorID, INVESTORID);
strcpy(orderInsert.InstrumentID, INSTRUMENTID);
strcpy(orderInsert.OrderRef, orderRef);

orderInsert.Direction = THOST_FTDC_D_Buy;
orderInsert.CombOffsetFlag[0] = THOST_FTDC_OF_Open;
orderInsert.CombHedgeFlag[0] = THOST_FTDC_HF_Speculation;
// volume of the instrument
orderInsert.VolumeTotalOriginal = 1;
// type of volume condition
// THOST_FTDC_VC_AV : any volume
// THOST_FTDC_VC_MV : minimum volume
// THOST_FTDC_VC_CV : all the volume
orderInsert.VolumeCondition = THOST_FTDC_VC_AV;
// minimum volume
orderInsert.MinVolume = 1;
// reason of forcible close
// see TThostFtdcForceCloseReasonType in head file for details
orderInsert.ForceCloseReason =
        THOST_FTDC_FCC_NotForceClose;
// is auto suspend
// 0:no,1:yes
orderInsert.IsAutoSuspend = 0;
// is forcible close
// 0:no, 1:yes
orderInsert.UserForceClose = 0;
```

Different orders should be designated with different values for corresponding fields.

Limit Order

```
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_AnyPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
// price
orderInsert.LimitPrice = YourPrice;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_IOC;
orderInsert.TimeCondition = THOST_FTDC_TC_GFD;
```

Market Order

```
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_LimitPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_AnyPrice;
// price
orderInsert.LimitPrice = 0;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_GFD;
orderInsert.TimeCondition = THOST_FTDC_TC_IOC;
```

Conditional Order

```
// type of condition
// THOST_FTDC_CC_Immediately : immediately match with current
market price
// see TThostFtdcContingentConditionType in head files for details
orderInsert.ContingentCondition = THOST_FTDC_CC_Immediately;
// stop price
// triggered when price falls or rises to this price
orderInsert.StopPrice = 2150;
// price type of order
// see TThostFtdcOrderPriceTypeType in head file for details
//THOST_FTDC_OPT_LastPrice;
//THOST_FTDC_OPT_AnyPrice;
orderInsert.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
// price
orderInsert.LimitPrice = YourPrice;
// valid type
// see TThostFtdcTimeConditionType in head file for details
//THOST_FTDC_TC_IOC;
orderInsert.TimeCondition = THOST_FTDC_TC_GFD;
```

Conditional Order Introduction

A conditional order is an order, which is executed only when a specified condition is satisfied. The condition can be based on the latest quotation or on the time. For example, if an investor has one short position of IF1410, he hopes to buy one to close the position when the latest price is lower than 2200, then he can place a conditional order. When the quotation data fluctuates and the condition is satisfied, the order will be sent out automatically. By using conditional order, he needs not stare at the computer for monitoring the quotations. For more effective use of conditional orders, investors can submit stop-and-limit orders and market-if-touched orders.

4.6.1. FOK & FAK

FOK（Fill or Kill）is a special conditional order. After the order is accepted by the exchange, if the order can be fulfilled at the moment, the order will be fulfilled immediately; otherwise the entire order will be canceled.

FAK（Fill and Kill）is also a special conditional order. After the order is accepted by the exchange, if the order can be partially filled at the moment, part of the order will be filled and remaining of the order will be canceled immediately.

CTP API realizes FAK and FOK instructions by the combination of fields.

	FAK	FOK
TThostFtdcOrderPriceTypeType	THOST_FTDC_OPT_LimitPrice	THOST_FTDC_OPT_LimitPrice
TThostFtdcTimeConditionType	THOST_FTDC_TC_IOC	THOST_FTDC_TC_IOC
TThostFtdcVolumeConditionType	THOST_FTDC_VC_AV / THOST_FTDC_VC_MV	THOST_FTDC_VC_CV

MinVolume Field

For FAK instruction, THOST_FTDC_VC_AV means any volume, while THOST_FTDC_VC_MV means the minimum volume. If THOST_FTDC_VC_MV is set, investors should specify the minimum volume that needs to be filled. If the volume can be filled is less than the MinVolume, the entire order will be canceled and no volume will be filled.

CombOffsetFlag Field

CombOffsetFlag is used to specify the offset attribute (open, close, closetoday) for the order.

```
req.CombOffsetFlag[0] = THOST_FTDC_OF_OPEN
```

The field is a character array in length of 5. It can be used to describe the order offset attribute for both common and combination orders. For a common order, only the first item should be designated, while for a combination order, the first and second items should be designated. Please refer to ThostFtdcUserApiStruct.h

for allowed values.

```
// open position
#define THOST_FTDC_OF_Open '0'
// close position
#define THOST_FTDC_OF_Close '1'
// force close position
#define THOST_FTDC_OF_ForceClose '2'
// close position which was opened within current trading day
#define THOST_FTDC_OF_CloseToday '3'
// close position which was opened before current trading day
#define THOST_FTDC_OF_CloseYesterday '4'
// Force off
#define THOST_FTDC_OF_ForceOff '5'
// Local Force close
#define THOST_FTDC_OF_LocalForceClose '6'

typedef char TThostFtdcOffsetFlagType;
```

Positions of SHFE are divided into today positions and yesterday positions. When closing positions, it is necessary to specify the instruction is to close today or yesterday positions.

If using THOST_FTDC_OF_Close for SHFE's positions, the result is the same with THOST_FTDC_OF_CloseYesterday. If using THOST_FTDC_OF_CloseToday or THOST_FTDC_OF_CloseYesterday for other exchanges, the result is the same with THOST_FTDC_OF_Close.

Note: Currently, four exchanges are all first open, first close when closing positions. Besides the first open and first close rule, DCE and CZCE have the rule of closing the common orders first, then the combination orders.

4.6.2. Order Sequence Numbers

In CTP and exchange systems, each order has 3 groups of sequence numbers to assure the uniqueness.

FrontID + SessionID + OrderRef

After login, the trading kernel will return the FrontID and current connection's SessionID. These two values are unchanged in this connection.

The OrderRef is a field in CThostFtdcInputOrderField structure. To assure the uniqueness, developers can make it incremented by one from MaxOrderRef for each order in one login session. If it is not designated, the trading kernel will assign a unique value to it automatically.

Client programs can maintain their own sequence numbers and can cancel an order **at any time** via this group of sequence numbers.

ExchangeID + TraderID + OrderLocalID

After the trading kernel submits the order to the offer manager, the trading kernel generates the OrderLocalID and returns it to the client program. The ExchangeID is the exchange's code, which the order's contract belongs to. The TraderID is designated by the trading kernel. The client program can also use this group of sequence numbers to cancel an order.

Unlike the first group, this group is maintained by the trading kernel.

ExchangeID + OrderSysID

After receiving an order, the exchange will generate the OrderSysID for the order. And this group of sequence numbers will be returned to the client program via CTP. The client program can also use this group of sequence numbers to cancel an order.

This group of sequence numbers is maintained by exchanges.

4.6.3. Order Return

The method onRtnOrder is used to return an order return. The data structure is CThostFtdcOrderField.

Order return is used to notify the client program about an order's latest status, e.g. submitted, canceled, not-queuing, and filled, etc. Each time the order status is changed, this method will be invoked once.

VolumeTotalOriginal & VolumeTraded & VolumeTotal

The above three fields are the original volume, filled volume and remaining unfilled volume of an order.

If one order is filled for several times, each transaction will have one order return.

When a conditional order is executed, the trading kernel will validate the order. If it fails the validation, OnRtnErrorConditionalOrder will be invoked and error message will be returned.

4.6.4. Trade Return

The method onRtnTrade is used to return a trade return. Each time the order is partially filled for fulfilled, this method will be invoked once. Trade return only contains the information of contract, filled volume, and price, etc. It does not have investor's positions and funds after this transaction is processed.

The method ReqQryTradingAccount is used for querying the latest status of investor's funds, e.g. margin, transaction fee, position profit, available funds, etc.

Query Margin Rate

The method for querying margin rate is ReqQryInstrumentMarginRate.

It is only used for common contract. For combination contract, users can query the rates of two legs of the combination contract, and then calculate the rate according to exchanges' rules.

4.7. Place a Parked Order

The parked order is the only one order type that can be submitted to exchanges in non-trading periods (before call auction or in the rest time between trading sessions). Parked orders will be triggered at the beginning of next trading session. Users can place both parked order and parked cancellation order.

When a parked order is triggered (Parked Order Insert), a new order is submitted to the exchange.

When a parked cancellation order is triggered (Parked Order Action), an order cancellation is submitted to the exchange to cancel an existed order.

The ReqParkedOrderInsert method is used to place a parked order. The main data structure is CThostFtdcParkedOrderField. The usage of ReqParkedOrderInsert is similar to the ReqOrderInsert. Its response method is OnRspParkedOrderInsert, which returns the response from the trading kernel.

The ReqParkedOrderAction method is used to place a parked cancellation order and the corresponding response method is OnRspParkedOrderAction.

The ReqRemoveParkedOrder method is used to delete a parked order, which is submitted but not triggered.

The ReqRemoveParkedOrderAction method is used to delete a parked cancellation order, which is submitted but not triggered.

The ReqQryParkedOrder and ReqQryParkedOrderAction methods are used to query parked orders and parked cancellation orders. Their response methods are OnRspQryParkedOrder and OnRspQryParkedOrderAction.

After a parked order or parked cancellation order is triggered, the order turns into a regular order. The processing is exactly the same with regular order insertion or order cancellation. So far, after a parked order is triggered, the order reference of the submitted order is generated by CTP and cannot be controlled by client programs.

4.8. Cancel an Order

The order cancellation method is ReqOrderAction. And the data structure is CThostFtdcInputOrderActionField. Canceling an order and placing an order are very similar, but there are two points should be paid attention to:

1. ActionFlag Field
   Currently, domestic exchanges only support canceling an order and do not support amending an order, so ReqOrderAction method only supports canceling an order and the available value for ActionFlag field can only be THOST_FTDC_AF_Delete.

2. Sequence Numbers

Order cancellation needs to locate the original order via sequence numbers. The three groups of the sequence numbers introduced in the last section could all be used to cancel an order.

Response and Return for Canceling an Order

OnRspOrderAction: Return the response of the order cancellation, containing error messages returned by the trading kernel.

OnRtnOrder : After the trading kernel validates the order cancellation instruction, the instruction will be submitted to the exchange and new order status of the related order will be returned.

OnErrRtnOrderAction: The exchange will validate the order cancellation instruction again. If the exchange finds it illegal, this method will be called and an error message will be returned. And if the order is valid, new order status of the related order will be returned as well (OnRtnOrder).

4.9. RFQs and Quotes

RFQs

The ReqForQuoteInsert method is used to submit a RFQ for investors. The main data structure is CThostFtdcInputForQuoteField. Only the contract code and quote reference number need to be passed in.

The OnRspForQuoteInsert will be called to return an error message, if the RFQ fails the validation. If the RFQ is valid, no message will be returned.

Market Makers Receive RFQs

In the trade API, users with market maker's privilege will receive the investors' RFQs automatically after login. Requesting and subscribing the RFQs are not needed. The OnRtnForQuoteRsp is used for receiving RFQs.

Market Maker Insert Quotes

To insert a quote, the ReqQuoteInsert method is used and the main data structure is CThostFtdcInputQuoteField.

```
CThostFtdcInputQuoteField quote;
memset(&quote, 0, sizeof(quote));
strcpy(quote.BrokerID, IDofBrokerageFirm);
strcpy(quote.InvestorID, IDofInvestor);
strcpy(quote.InstrumentID, IDofInstrument);
strcpy(quote.QuoteRef, RefNumberOfQuote);
strcpy(quote.AskOrderRef, RefNumberOfAskOrder);
strcpy(quote.BidOrderRef, RefNumberOfBidOrder);
quote.AskPrice=PriceofAskOrder;
quote.BidPrice=PriceofBidOrder;
quote.AskVolume=VolumeofAskOrder;
quote.BidVolume=VolumeofBidOrder;

// offsetflag of ask order
quote.AskOffsetFlag=THOST_FTDC_OF_Open;

// offsetflag of bid order
quote.BidOffsetFlag=THOST_FTDC_OF_Open;

// hedgeflag of ask order
quote.AskHedgeFlag=THOST_FTDC_HF_Speculation;

//hedgeflag of bid order
quote.BidHedgeFlag=THOST_FTDC_HF_Speculation;
```

The OnRspQuoteInsert method is called when the quote fails the validation and an error message is returned by the trading kernel.

When a quote passes the validation and the trading kernel submits the quote to the exchange, the OnRtnQuote will be invoked. Two regular orders will be derived from the quote and be submitted to the exchange together with the quote. In this case, onRtnOrder will also be invoked.

From version 6.3.0, two fields AskOrderRef and BidOrdreRef are added in the API. These two values will be assigned to the OrderRef fields of the two derived orders respectively. If client programs do not designate these values, the trading kernel will assign them automatically.

Cancel a Quote

The ReqQuoteAction method is used to cancel a quote. The usage is similar with the ReqOrderAction. The QuoteRef+SessionID+FrontID is used to cancel a quote.

Canceling a quote will cancel both the quote and the remaining unfilled derived orders. A market maker can also cancel a derived order separately. The method is the same with canceling an order in previous section.

4.10. Exercise an Option

The ReqExecOrderInsert is used to declare that investors intend to exercise their options. The main data structure is CThostFtdcInputExecOrderField.

```
CthostFtdcInputExecOrderField execOrderReq;
memset(&execOrderReq, 0, sizeof(execOrderReq));
strcpy(execOrderReq.BrokerID, IDofBrokerageFirm);
strcpy(execOrderReq.InvestorID, IDofInvestor);
strcpy(execOrderReq.InstrumentID, IDofInstrument);
strcpy(execOrderReq.ExecOrderRef, RefNumberofExecOrder);
execOrderReq.volume = volumeofInstrument;

// for SHFE, it should be THOST_FTDC_OF_CloseToday or
// THOST_FTDC_OF_CloseYesterday
execOrderReq.OffsetFlag = THOST_FTDC_OF_Close;
execOrderReq.HedgeFlag = THOST_FTDC_HF_Speculation;

// to exercise or to abandon
execOrderReq.ActionType = THOST_FTDC_ACTP_Exec;

// long or short position to hold after exercising
execOrderReq.PosiDirection = THOST_FTDC_PD_Long;

// whether to hold future positions after exercising
// CFFEX: use THOST_FTDC_EOPF_UnReserve
// DCE/CZCE: use THOST_FTDC_EOPF_Reserve
execOrderReq.ReservePositionFlag =
                              THOST_FTDC_EOPF_UnReserve;

// whether to close future positions automatically
// CFFEX: use THOST_FTDC_EOCF_AutoClose
// DCE/CZCE: use THOST_FTDC_EOCF_NotToClose
execOrderReq.CloseFlag = THOST_FTDC_EOCF_AutoClose;
```

If the request fails the validation, the OnRspExecOrderInsert method will be invoked. If the request passes the validation, the OnRtnExecOrder will be invoked.

After executing an option, if a client receives an unexercised message, it means the request has been accepted by the exchange. Because options' execution is processed during clearing period, so the unexercised message will be returned by the exchange in trading time.

Four Exchanges' Rules for Exercising Options

CFFEX

In-the-money options will be exercised automatically and out-of-the-money options will be given up automatically. Forced exercise is not allowed for out-of-the-money options. After options are exercised, positions will be closed out immediately (because the option exercise date is the same with the corresponding underlying future's delivery date).

SHFE

In-the-money options will be exercised automatically and out-of-the-money options will be given up automatically. Investors can choose to execute out-of-the-money options or not. And after executing, it is optional to keep the futures' positions or not.

DCE

If futures companies set the auto-exercise flag in DCE official website, options will be exercised automatically. Otherwise, the investors need to apply to exercise or give up on their own.

CZCE

In-the-money options will be exercised automatically and out-of-the-money options will be given up automatically. Investors may apply to exercise or give up on their own.

5. Appendix

5.1. Data Flow files

When initiating, the CTP API may generate some flow files on local machine for saving the number of received packets in public, dialog and private streams of the current trading day.

Data flow files are mainly used for re-transmitting data in resume mode. And when using CTP Risk API, data flow files are also used for batch querying.

Developers need to pay attention to following issues:

1. The client program will read and write the data flow files very frequently. If the client program does not manage the number of file handlers in the system, file handles may be used up.
2. When developing the program for multiple accounts, please pay attention to not put all account's flow files in one directory, that may cause only one account can receive the returns while other accounts cannot.

Data Flow Files Generated by Quotation API

Quotation API	
DialogRsp.con	Received dialog response data flow
QueryRsp.con	Received query response data flow
TradingDay.con	Trading Day

Data Flow Files Generated by Trade API

Trade API	
DialogRsp.con	Received dialog response data flow
QueryRsp.con	Received query response data flow
TradingDay.con	Trading Day
Public.con	Received public return data flow
Private.con	Received private return data flow

5.2. Flow Control

5.2.1. Query Flow Control

Limitations for query operations in the trade API is as follows:

- Users can only send one query operation per second at most.
- Only one in-process query is allowed at the same time. If a query operation is sent out, but the response has not been received yet, then the query operation is in-process.

Above limitations are only for query operations in the trade API. Placing orders, canceling orders, submitting quotes and RFQs do not have these limitations.

5.2.2. Order Flow Control

For order flow control, some relevant parameters need to be configured in futures companies' system.

If these parameters are not configured, the default limitations are as follows:

- In one session, every client can only send 6 trading instructions (e.g. placing orders, canceling orders, etc.) per second at most.
- One account can only establish up to 6 sessions at the same time.

Note: No errors will be returned, if operations exceed the limitations. Orders will be in queuing status waiting to be handled.

5.3. Disconnection

The method OnFrontDisconnected will be invoked if a disconnection occurred. The disconnection cause is in nReason parameter.

Possible Causes:

Decimal System(10D)	Hexadecimal(16H)	Explanation
4097	0x1001	Fails to read from internet.
4098	0x1002	Fails to write into internet.
8193	0x2001	Time out receiving heart-beating.
8194	0x2002	Time out sending heart-beating.
8195	0x2003	Received error messages.

There are two situations for the disconnection between a client and a server:

- Network issues cause the disconnection.
- The server disconnects initiatively.

There are two causes for a server to disconnect initiatively.

- The client has not received packets from the server for a long time. Timeout occurs.
- The amount of the connections exceeds the limitation.

Heartbeat Mechanism

The CTP uses heartbeat mechanism to confirm that the connection between a client and a server is valid or not. If the client does not have messages to receive from the server, the server will send heartbeat packets timely to the client. Currently, the heartbeat mechanism is only implemented within the APIs, clients may not know it, so that the OnHeartBeatWarning method will not be invoked.

Timeout

Network delay may cause a server disconnects with a client initiatively.

The default timeouts are 120 seconds for reading, 60 seconds for writing, and 80 seconds for heartbeat.

When a client and a server are disconnected, the trade API will try to reconnect automatically once per 5 seconds.