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

CTP Client Development Guide

Improving people’s lives with high-technology

Update：2017-10-9
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 its underlying mechanisms, and outlines the general steps needed to develop a client program. It also answers some of the most-frequently asked questions by developers.

This guide summarizes the contents of existing documents and complements those of new features in the latest version of API. To make the contents of this document complete, accurate, and easy to understand, we welcome error or omission reports, or anything else that you think might help improve the document. Please feel free to contact us.

Contact Information

Financial Business Department of SFIT
apiSupport@sfit.shfe.com.cn

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

© Shanghai Future Information Technology Co., Ltd

Table of Contents

1. CTP ---------- 5

1.1. Introduction ---------- 5

1.2. FTD Communication Protocol ---------- 6

1.2.1. Communication Mode ---------- 6

1.2.2. Data Stream ---------- 7

1.3. Two Data Exchange Modes ---------- 8

1.3.1. Request/Response Mode ---------- 8

1.3.2. Publish/Subscribe Mode ---------- 8

1.4. CTP Architecture ---------- 9

2. API ---------- 12

2.1. Introduction ---------- 12

2.1.1. Interface Files ---------- 12

2.2. General Rules ---------- 13

2.2.1. Naming Conventions ---------- 13

2.2.2. Interface Classes ---------- 14

2.2.3. General Parameters ---------- 14

2.2.4. General Initialization Steps for the trade API ---------- 14

3. Demo of acquiring quotations ---------- 16

3.1. Preparation ---------- 16

3.2. Initialization for Quotation API ---------- 17

3.3. Login ---------- 18

3.4. Subscribe Quotation Data ---------- 20

© Shanghai Future Information Technology Co., Ltd

3.5. Subscribe and Receive Requests for Quotes ---------- 21

4. Demo of Trading ---------- 22

4.1. Initialization for Trade API ---------- 23

4.2. Login ---------- 23

4.3. Confirm Settlement Info ---------- 24

4.4. Order Processing Flow ---------- 26

4.5. Methods for Processing Orders ---------- 29

4.6. Place an Order ---------- 30

4.7. Place a Parked Order ---------- 37

4.8. Cancel an Order ---------- 38

4.9. RFQs and Quotes ---------- 38

4.10. Exercise an Option ---------- 40

5. Appendix ---------- 42

5.1. Data Flow files ---------- 42

5.2. Flow Control ---------- 45

5.2.1. Query Flow Control ---------- 45

5.2.2. Order Flow Control ---------- 45

5.3. Disconnection ---------- 45

© Shanghai Future Information Technology Co., Ltd

1. CTP

1.1. Introduction

The Comprehensive Transaction Platform, known as CTP, is a future brokerage management system developed specially for futures companies. It is composed of a trading system, a risk management system, and a 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. The CTP can connect four(five?) 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 inherits the experience of SHFE's "New Generation Exchange System (NGES)", which represents the top level of international futures derivatives trading systems. It adopts an innovative distributed architecture, which can precisely replay history and record every operation.

Being purely RAM based, the CTP supports 7*24 continuous trading. There is no need for operations staff to stop and start the system manually every day, because of the "One Button Operation and Maintenance" feature. This feature allows adding trading centers to expand business, without any additional operation or maintenance cost.

Currently, only the CTP has realized the "One Button Switch" for multiple active trading centers that support the FENS mechanism. In case one trading center crashes, 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 submit trading instructions. The interface uses an open interface (API), which has already become a de facto standard in the futures industry.

The CTP mini version (CTP mini) is a faster and more lightweight system. Compared with the CTP, it pursues the target of more compact configuration and less consumption of resources and equipment. Client programs developed via the CTP API are fully compatible with the CTP mini system.

© Shanghai Future Information Technology Co., Ltd

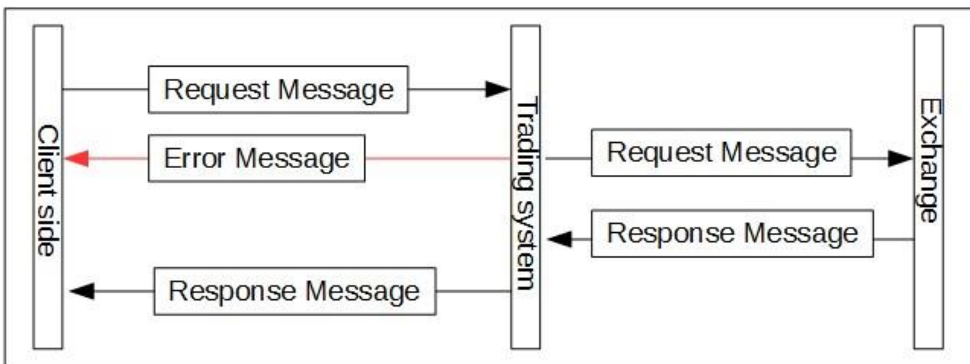
1.2. FTD Communication Protocol

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

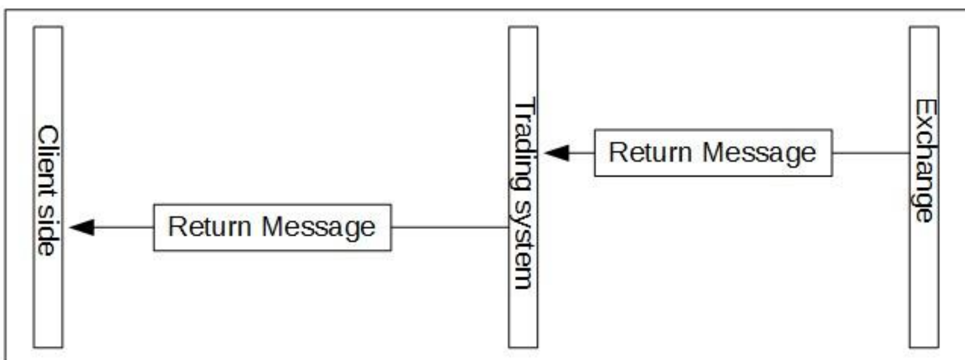
1.2.1. Communication Mode

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

Dialog Communication Mode is that the communication requests are initiated by client programs. The requests (e.g. instrument queries) 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.

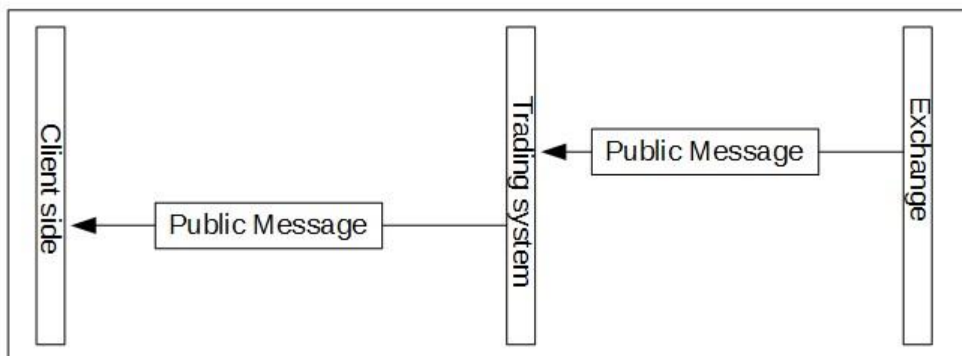


Private Communication Mode is that the trading system sends information (e.g. trade return) initiatively to a particular client.



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

© Shanghai Future Information Technology Co., Ltd



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

1.2.2. Data Stream

In FTD protocol, we need to distinguish between two important concepts, communication mode and data stream. Data stream is a unidirectional or bidirectional, continuous, unique and complete sequence of datagrams, 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 one type of communication mode may also 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 specifies which datagram works under which communication mode, but does not specify the division of data streams.

Different communication modes manage 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, a reconnection will start a new data stream, which is totally different from the original one. If one client sends a request and disconnects before receiving the response, the response will not be received via the new data stream after reconnection.

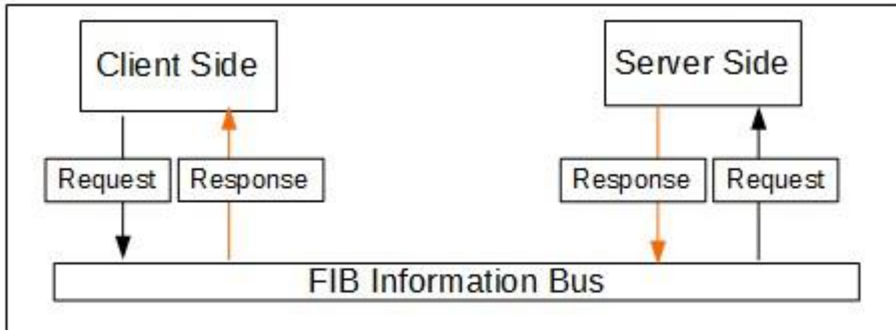
For private and broadcast communication modes, one data stream corresponds to all connections of a certain feature in one trading day. Unless specified, clients will resume from last transmission rather than from the beginning.

© Shanghai Future Information Technology Co., Ltd

1.3. Two Data Exchange Modes

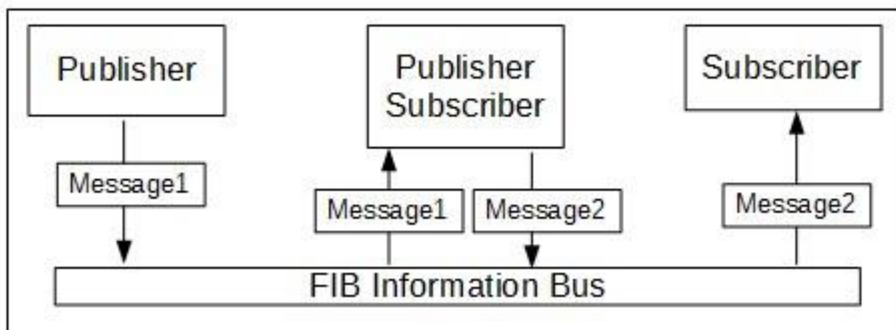
1.3.1. Request/Response Mode

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



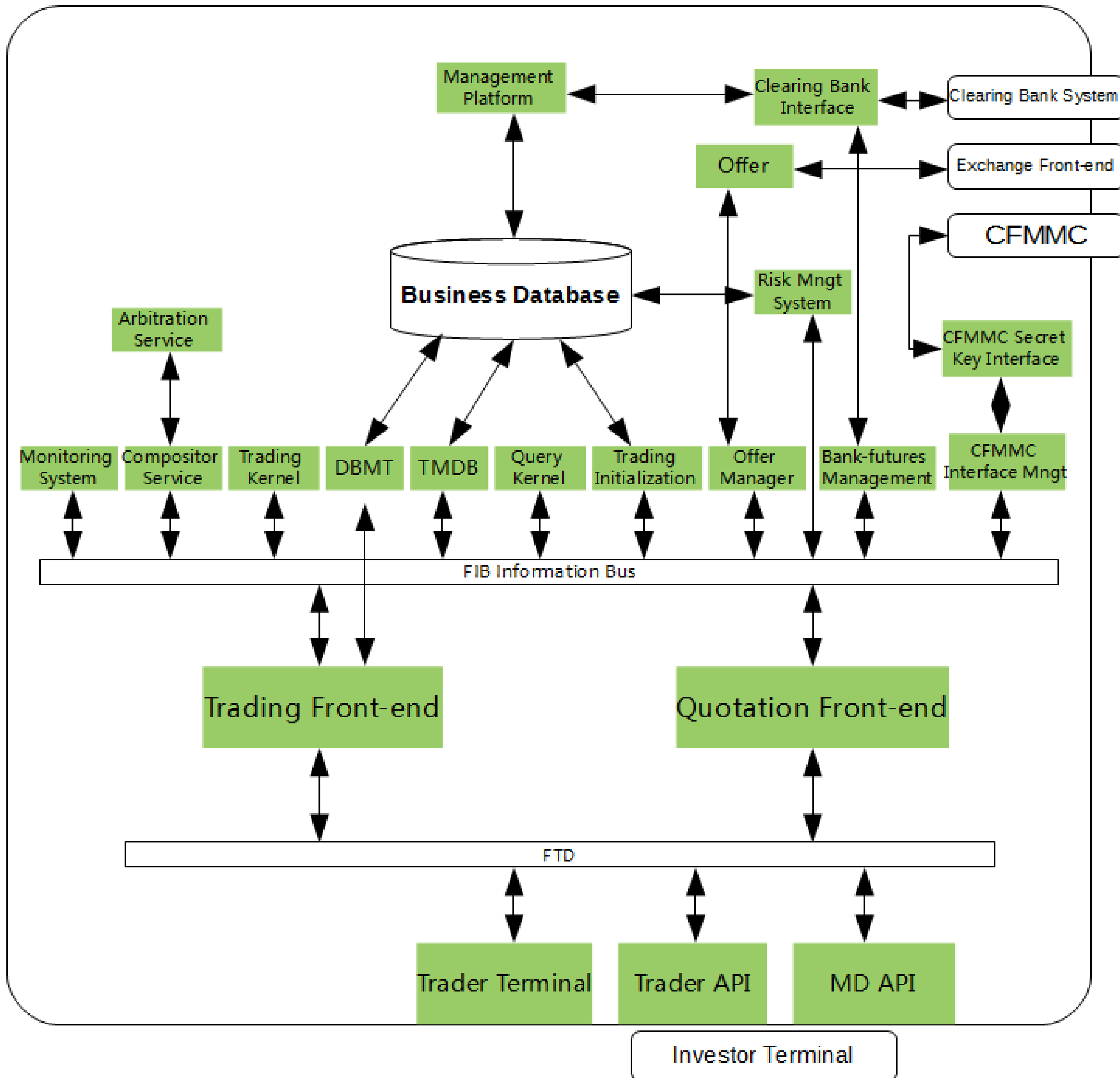
1.3.2. Publish/Subscribe Mode

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 to each other directly to transmit messages. One FIB application can be both a publisher and a subscriber.



© Shanghai Future Information Technology Co., Ltd

1.4. CTP Architecture



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

Trader Terminal uses CTP's UserAPI. It provides functions like ordering, bank-futures transfer, querying trading data, etc. for futures companies. The UserAPI is for futures companies and the data range it can control is larger than the trade API. Considering data security and privilege partition, the UserAPI will not be disclosed to futures companies and investors for now. (the audience should not know about Trader Terminal or UserAPI)

FTD is futures trading data exchange protocol.

© Shanghai Future Information Technology Co., Ltd

Trading Front-end connects to trading clients via TCP protocol and connects to other background services via FIB. The trading front-end is responsible for communication and is irrelevant to business, so that it can balance the load, 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 the Offer Manager via FIB and forwards the quotation data to trading clients, which have subscribed the quotations of certain contracts via TCP connections.

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

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

Arbitration Service manages the status switch for the compositor service.

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

Trading Kernel calculates real-time funds and positions based on investors' positions, orders, trades and funds deposit and withdrawal, for the purpose of pre-trade risk management. 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 logic implementation as the trading kernel. Based on real-time calculation for investors, query kernel updates the memory database to provide query service for trading clients via FIB.

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

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

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

Offer Manager manages trading and quotation offers. It allows the trading kernel to avoid processing the complex communication between Offers and exchange front-ends , so as to simplify the trading kernel's processing logic.

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

© Shanghai Future Information Technology Co., Ltd

Risk Management System obtains the transaction sequence published by the compositor and trading results published by the trading kernel to monitor trading data in real-time, and provides trial risk 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, and provides data exchange channels between the trading system and bank systems.

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

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

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

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

© Shanghai Future Information Technology Co., Ltd

2. API

2.1. Introduction

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

The Trading System API is used for receiving exchanges' quotation data and sending trading instructions, such as subscribing quotations, placing an order, canceling an order, placing a parked order, processing futures-bank transfer, querying information, etc. The Trade API is the most widely used interface. It's target users are mainly client software providers (e.g. Q7) and individual, institutional and proprietary investors that have 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 mitigate risks when necessary. It provides risk disclosure and risk management for risk control personnel in futures companies.

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

The Settlement API consists of a series of database instructions. It is deployed on computers that can access the 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.

(should Risk API and Settlement API be mentioned here?)

2.1.1. Interface Files

Developers can ask their futures company for the Trade 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++ header file

© Shanghai Future Information Technology Co., Ltd

<table>
<tr><td>ThostFtdcMdApi.h</td><td>Quotation interface C++ header file</td></tr>
<tr><td>ThostFtdcUserApiStruct.h</td><td>Defines all data structures</td></tr>
<tr><td>ThostFtdcUserApiDataType.h</td><td>Defines all data types</td></tr>
<tr><td>thosttraderapi.dll</td><td rowspan="2">Dynamic link library of the Trade interface.</td></tr>
<tr><td>thosttraderapi.lib</td></tr>
<tr><td>thostmduserapi.dll</td><td rowspan="2">Dynamic link library of the quotation interface.</td></tr>
<tr><td>thostmduserapi.lib</td></tr>
<tr><td>error.dtd</td><td rowspan="2">API error codes and information in xml format.</td></tr>
<tr><td>error.xml</td></tr>
</table>

2.2. General Rules

2.2.1. Naming Conventions

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

© Shanghai Future Information Technology Co., Ltd

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 actively. 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 multiple times continuously. In such a case, requests and responses are correlated with the RequestID.

IsLast indicates whether the 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 multiple times. For example, if there is one packet, isLast of the response is true. If there are two packets, IsLast of the first response is false, while that of 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 indicates the error type returned by the trading kernel.

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

2.2.4. General Initialization Steps for the trade API

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

For now, the Risk API and Settlement API are not covered in the document.

1. Create “SPI” and “API” instances
   The “SPI” is a user-defined class that derives 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 different front-end network addresses for corresponding APIs. 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 the trade API, not necessary for the quotation API) to receive public data, such as trading status in the market. By default, data published by exchanges is resumed from the last disconnection (resume mode). Developers can also designate to receive from the beginning (restart mode) or from the login time (quick mode).

5. Subscribe private stream (only for the trade API, not necessary for the quotation API) to receive private data, such as order returns. By default, data published by exchanges is resumed from the last disconnection (resume mode). Developers can also designate to receive from the beginning (restart mode) or from the login time (quick mode).
6. Initialize (Init).
7. Wait for the interface thread to terminate (Join).

For sample code and more details please refer to the “Develop a DEMO” chapter.

© Shanghai Future Information Technology Co., Ltd

Develop a Demo

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

This document does not explain programming details and does not contain graphical user 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

Up till now, we have released the CTP's trade API for Windows, Linux, Android and iOS. This guide takes the Windows version on PC in C/C++ language as an example.

Here are three recommended C/C++ IDEs:

- **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 header 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 the connection to the server is established but before logging in to the trading system. Usually it's where you put your login code.
OnFrontDisconnected	This function is invoked after the client disconnects from the server. API will try to reconnect automatically. Parameter of this function indicates the reason of the connection failure.

© Shanghai Future Information Technology Co., Ltd

OnRspUserLogin	Return response from the login request. Error message such as “illegitimate login” usually means wrong password.
OnRspSubMarketData	Return response from a request of subscribing market data. Usually it is invoked when an error happens to the request of subscribing market data. Reason of error and instrument can be found in the parameters.
OnRspUnSubMarketData	Return response from a request of unsubscribing market data.
OnRtnDepthMarketData	Return price quotation of 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 an invalid IP to the front-end.
OnHeartBeatWarning	This function is invoked if no message has been received for a preset period of time. It can be used to send heart beats to suggest that the connection between the client and the server is still alive. The interface is no longer valid.

（缺行情接口工作原理简图）

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 the **CreateFtdcMdApi** method. The first parameter is the directory of local flow files. Flow files are local files generated by quotation API or trade API, postfixed with .con, which record the number of all packets received by the client. The second parameter indicates the transmission 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 an intranet.

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

Line 3

Create an SPI instance. The QCTPMdSpi is a sample entity class deriving **CThostFtdcMdSpi.**

Line 4

Register the SPI instance to an API instance.

Line 5

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

Note: With 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 to initialize the working thread of the quotation API. After initialization, the thread starts automatically and uses the address registered in the last step to connect to the server.

All CTP APIs have independent working threads. When developing GUI programs, developers should pay attention to potential conflicts between threads.

3.3. Login

After initialization, the quotation working thread will use the registered front-end address to establish a non-authenticated connection with the server. After connection, **OnFrontConnected** will be invoked. If not connected or disconnected during program execution, **OnFrontDisconnected** will be invoked.

After connection, the client program can request login using **ReqUserLogin(or ReqUserLogin2)**. The key data structure is **CThostFtdcReqUserLoginField**.

Note: ReqUserLogin2 (login request 2) and ReqUserPasswordUpdate2 (user password update request 2) are two enhanced APIs, which demands the CTP backend version to be P20170915 or later.

© Shanghai Future Information Technology Co., Ltd

```
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 of 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 sample code) returned by this function indicates whether the request was sent out successfully or not, rather than whether the request was processed by the server or not.

Return Value

- 0, sent successfully.
- -1, failed to send;
- -2, unprocessed requests exceeded the allowed volume;
- -3, the number of requests sent per second exceeds the allowed volume.

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 returns the response via the **OnRspUserLogin** method.

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

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

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

3.4. Subscribe Quotation Data

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

After logging in successfully, the client can subscribe quotations now.

The **SubscribeMarketData** is invoked to subscribe quotations. The first parameter is an array of all contracts to subscribe. 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 quotations, OnRspSubMarketData or OnRtnDepthMarketData may be invoked.

OnRspSubMarketData

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

OnRtnDepthMarketData

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

ReqUserLogout and UnSubscribeMarketData are used to log out and unsubscribe quotations. The usage is the same as logging in and subscribing quotations.

Note: For now, if logging out via ReqUserLogout, the API will disconnect the current connection. After re-login, the API will establish a new connection. The SessionID will be reset and the MaxOrderRef will restart from 0.

3.5. Subscribe and Receive Requests for Quotes

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

For now, exchanges take different ways to handle RFQs from investors and subscriptions from market makers. Also, they implement differently in processing quotes, RFQs, and other functions. Considering development efficiency, CTP realizes all the functions for investors to make RFQs and market makers to receive RFQs in the quotation API for now.

RFQ Implementation for Four Exchanges

SHFE&CFFEX

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

DCE&CZCE

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

CTP API sticks to the principle of being consistent with different exchanges. If exchanges improve their implementation for processing RFQs, CTP API will be adjusted accordingly.

API Methods

Quotation API

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

Trade API

© Shanghai Future Information Technology Co., Ltd

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

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

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

3.6. Explanation of Rule of Quotation Push

The CTP rule of quotation is :

√ twice quotation snapshots per second

√ push only when quotation update happens

√ push initial quotation after first connection

The basic principal for the CTP quotation push is pushing twice per second, but the push time (in milliseconds) is not strictly 000、500. That is, the push time could probably be 300、800.

There are two reasons for this: one is that quotations from exchanges (in milliseconds) are not constant; the other is that the push only occurs when the quotation update happens.

Besides, after a client is connected to CTP through CTP API, CTP will conventionally push the latest quotation, to notify the client if the current market has any quotation.

[notice]: The exchange also pushes quotations during the auction and matching period before opening. The timestamp of quotation at the open time could probably be equal to or greater than 500, or less than 500.

[notice]: In internet environment, after opening, CTP will push the states of contracts (for example, the state of a contract is updated to continuous trading after opening), and the push time of this message is almost identical with the first quotation time after opening.

4. Demo of Trading

© Shanghai Future Information Technology Co., Ltd

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

The usage is similar to the quotation API.

4.1. Initialization for Trade API

Just like the quotation API, developers should derive from the **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 as those of the quotation API (3.2 section).

Differences:

1. When creating an API instance, transmit protocol needs not be designated, so the method in line 2 only needs one parameter (flow 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 on current trading day.
   - Resume: to re-transmit by resuming and continuing from the last transmission;
   - Quick: to transmit from current login.
3. The front-end address to register is the trading front-end address.

4.2. Login

Authentication

© Shanghai Future Information Technology Co., Ltd

Before logging 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 personnel in futures companies. It can be configured to be turned off, so that clients do not need to be authenticated. Otherwise, futures companies need to 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 as that in quotation API.

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

Note: Currently, if a user uses ReqUserLogout to log out, the 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 to confirm the settlement info of the prior trading day, when they log in on current trading day for the first time.

Methods for Settlement Confirmation

Request	Reply	Description
ReqQrySettlementInfo	OnRspQrySettlementInfo	Query settlement info
ReqSettlementInfoConfirm	OnRspSettlementInfoConfirm	Confirm settlement info

ReqQrySettlementInfoConfirm	OnRspQrySettlementInfoConfirm	Query date of confirmation

© Shanghai Future Information Technology Co., Ltd

4.4. Position Calculation

The API for querying position is as follows:

Request	Reply	Description
ReqQryInvestorPosition	OnRspQryInvestorPosition	Query position (summary) info
ReqQryInvestorPositionDetail	OnRspQryInvestorPositionDetail	Query position detail info
ReqQryInvestorPositionCombineDetail	OnRspQryInvestorPositionCombineDetail	Query position combination detail info

[position detail]

CTP generates position detail records based on trading records from exchanges; each trading record corresponds to a position detail record.

[position summary]

CTP summarizes the position detail records according to contract, position direction, open date (only for SHFE, different for old position and new position).

In position summary:

YdPosition position amount at yesterday close time (≠current yesterday position amount, static, not vary for open and close during the day)

Position current position amount

TodayPosition today's new open position amount

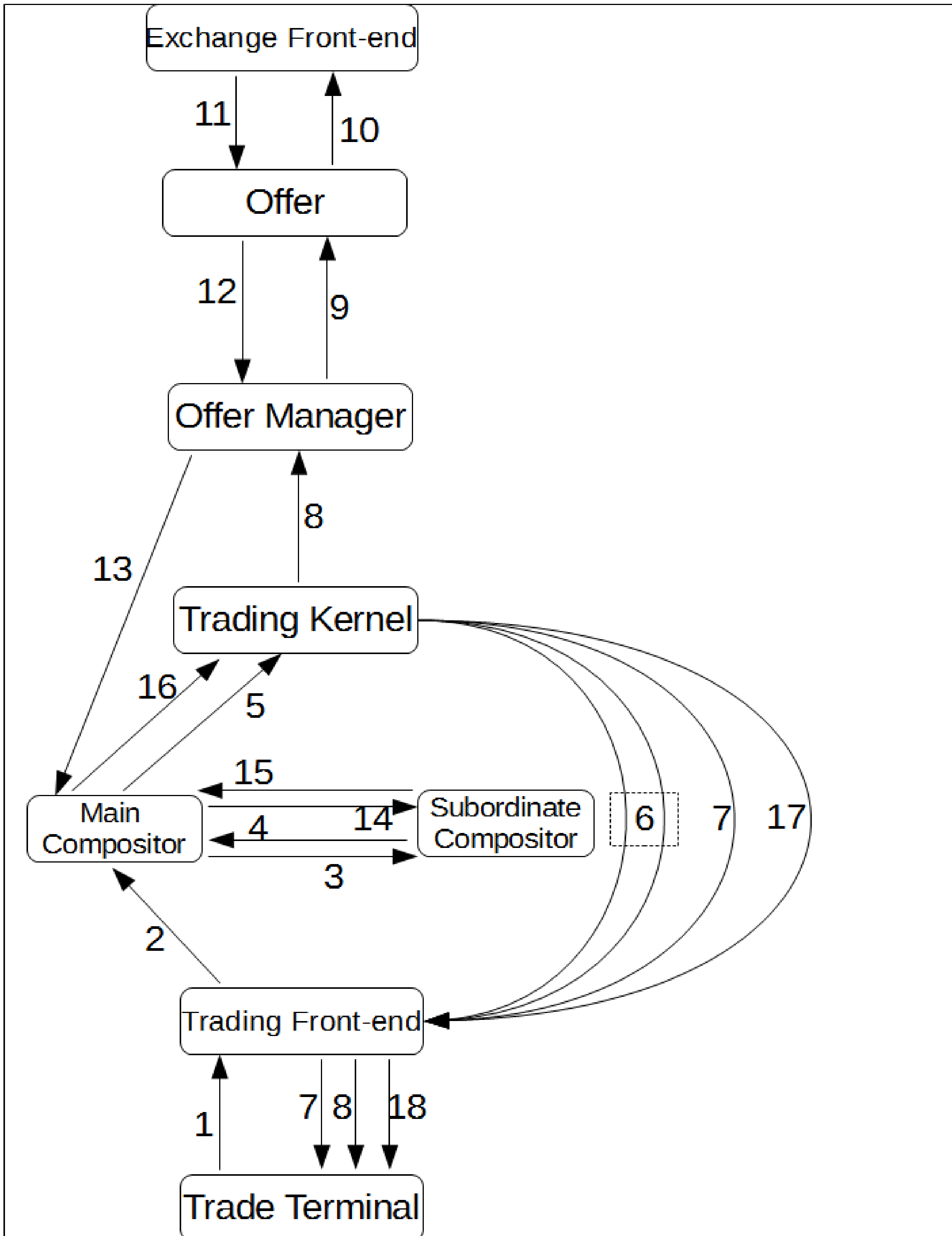
Current yesterday position = ∑Position − ∑TodayPosition

[position combination details]

Only for combination contracts' position detail records

4.5. Order Processing Flow

© Shanghai Future Information Technology Co., Ltd



1. The Trade Terminal submits an order request to the Trading Front-End via trade API.
2. The Main Compositor subscribes transaction requests 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 relevant 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 legitimate 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 processes in this step: 1. After receiving an 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 to request an order insertion and send it to the exchange. The package will be posted to the Offer Manager.
9. After receiving the order insertion request, the Offer Manager forwards the package to the 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 receives 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.

© Shanghai Future Information Technology Co., Ltd

4.6. Methods for Processing Orders

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

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

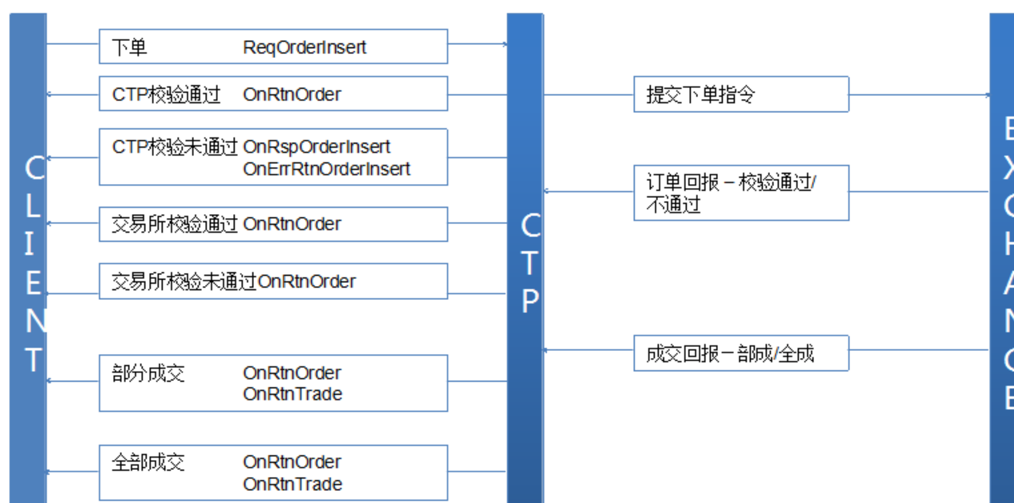
OnRtnOrder: Return the order status from an exchange. Each time the order status is changed, this method will be invoked once. It may be invoked several times during one order process, e.g. when the trading system submits an order to the exchange (the second process in step 8), when the exchange cancels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

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

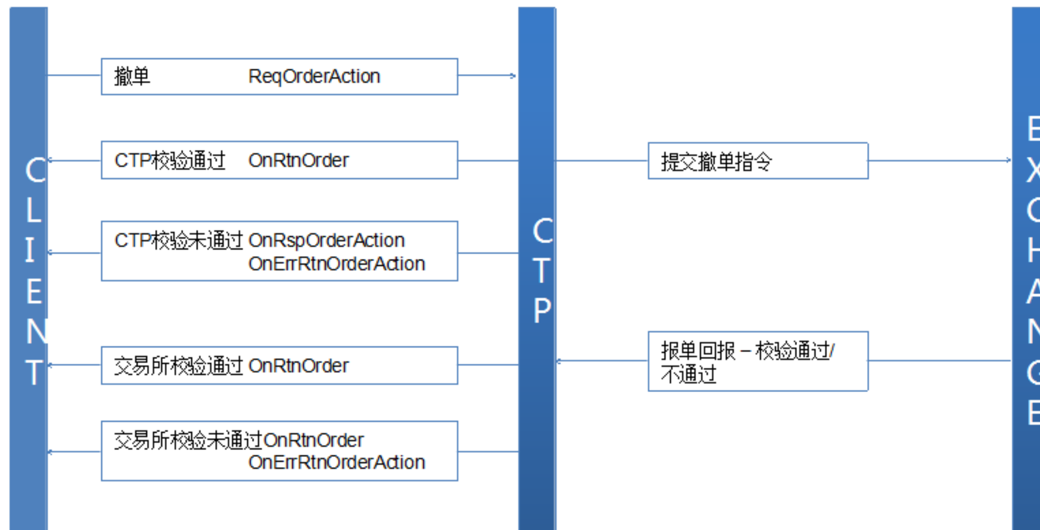
OnRtnTrade： If the order is filled, the exchange will return the filled order status and will return the trade information via this method.

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 will be "filled". But because CTP's trading kernel only updates the order status after receiving the trade return, we recommend that clients 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 fail. The reason is that the trading kernel receives the closing position instruction before the trade return, and the order status is not “filled” yet in the trading kernel.

报单处理流程-1



© Shanghai Future Information Technology Co., Ltd



4.7. Place an Order

ReqOrderInsert method is used to place 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;
```

© Shanghai Future Information Technology Co., Ltd

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, and he hopes to buy one to close the position when the 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 automatically. By using conditional order, he needs not stare at the computer to monitor the quotations. For more effective use of conditional orders, investors can submit stop-and-limit orders and market-if-touched orders.

4.7.1. 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 the remaining quantity 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 instructions, 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 5 character array. It can be used to describe the order offset attribute for both common and combinational orders. For a common order, only the first item should be designated, while for a combinational 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 (opened today) and yesterday positions (history positions). When closing positions, it is necessary to specify whether the instruction is to close today or yesterday positions.

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

Note: Currently, the 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 combinational orders.

4.7.2. Order Sequence Numbers

In CTP and exchange systems, each order has 3 groups of unique sequence numbers to avoid duplication.

FrontID + SessionID + OrderRef

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

The OrderRef is a field in the CThostFtdcInputOrderField structure. To assure the uniqueness, developers can make it increment 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 code of the exchange that 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.

4.7.3. Order Type Combination

The API supports all order instructions of the five exchanges. Various instructions take effect by various combinations.

		Limit Order	Market Order	Conditional Order
OrderPriceType	Price Type	THOST_FTDC_OPT_LimitPrice	~AnyPrice	LimitPrice
LimitPrice	Price	DIYPrice	0	DIYPrice
TimeCondition	Valid Date	THOST_FTDC_TC_GFD	IOC	GFD
VolumeCondition	Volume Type	THOST_FTDC_VC_AV	AV	AV

ContingentCondition	Contingent Condition	THOST_FTDC_CC_Immediately	Immediately	Assign Contingent Condition
Additional conditions				StopPrice

		FAK	FOK	Stop-DCE
OrderPriceType	Price Type	THOST_FTDC_OPT_LimitPrice	LimitPrice	LimitPrice
LimitPrice	Price	DIYPrice	DIYPrice	DIYPrice
TimeCondition	Valid Date	THOST_FTDC_TC_IOC	IOC	GFD
VolumeCondition	Volume Type	THOST_FTDC_VC_AV/MV	CV	AV
ContingentCondition	Contingent Condition	THOST_FTDC_CC_Immediately	Immediately	Touch/Touch Profit
Additional conditions		MinVolume		StopPrice

The CFFEX five-range instructions' combination rule：

CFFEX-AnyPrice to LimitPrice

- THOST_FTDC_OPT_AnyPrice
- THOST_FTDC_TC_GFD
- THOST_FTDC_VC_AV

CFFEX- five-range market Price

- THOST_FTDC_OPT_FiveLevelPrice
- THOST_FTDC_TC_IOC
- THOST_FTDC_VC_AV

CFFEX- five-range market Price to LimitPrice

- THOST_FTDC_OPT_FiveLevelPrice

© Shanghai Future Information Technology Co., Ltd

- THOST_FTDC_TC_GFD
- THOST_FTDC_VC_AV

CFFEX- BestPrice

- THOST_FTDC_OPT_BestPrice
- THOST_FTDC_TC_GFD
- THOST_FTDC_VC_AV

CFFEX- BestPrice to LimitPrice

- THOST_FTDC_OPT_BestPrice
- THOST_FTDC_TC_GFD
- THOST_FTDC_VC_AV

4.7.4. 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 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 an error message will be returned.

OrderStatus

- 0 fully filled
- 1 partially filled, order is still in the matching queue in exchange
- 3 not filled, order is still in the matching queue in exchange
- 5 already cancelled
- 6 unknown - order has already been submitted to the exchange, but not yet received the confirm information

© Shanghai Future Information Technology Co., Ltd

4.7.5. Trade Return

The method onRtnTrade is used to return a trade return. Each time the order is partially filled or fully filled, this method will be invoked once. Trade return only contains the information of contract, filled volume, and price, etc. It has only information about the trade, but not the investor's positions and funds after the trade.

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

We suggest that the client should use the trade return to judge if an order was filled, and if yes, the quantity and the price. If using order return (status being "partially filled" or "fully filled"), there's possibility that the closeout could fail, because there's theoretical time difference (very marginal) between the order return and the trade return, and the CTP back-end updates the order status based on the trade return.

Query Margin Rate

The method for querying margin rate is ReqQryInstrumentMarginRate.

It is only used for common contracts. For a combinational contract, users can query the rates of the two legs of the combinational contract, and then calculate the rate according to the exchange's rules.

4.8. Place a Parked Order

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

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 submitted but not triggered parked order.

© Shanghai Future Information Technology Co., Ltd

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

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 as 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.

4.9. 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 the ReqOrderAction method only supports canceling an order and the available value for the 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, which contains 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 a 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 illegitimate, this method will be called and an error message will be returned. And if the order is valid, a new order status of the related order will be returned as well (OnRtnOrder).

4.10. RFQs and Quotes

RFQs

The ReqForQuoteInsert method is used to submit an RFQ for investors. The main data structure is the 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 or 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 BidOrderRef are added to 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.

From version 6.3.6, the field ForQuoteSysID is added to the API. A market maker connecting to DCE and CZCE should fill in the bid region with the quote ID (ForQuoteSysID) from the quote response (OnRtnForQuoteRsp). Exchanges will count the bid finishing rate of a market maker according to this field.

Note: This field is not necessary for market makers connecting to CFFEX. The rule for CFFEX to determine the finishing rate of market makers is that if the market maker bids within a specific period of time (specified by exchanges, say 1 min) after the exchange sends the ask request to the market maker.

Cancel a Quote

The ReqQuoteAction method is used to cancel a quote. The usage is similar to 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 directly. The method is the same as canceling an order in previous section.

4.11. 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 option 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 as 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.

4.12. Margin of Combination Contract

Rules for margin of combination contract:

The margin rule of combination contracts that CFFEX is going to implement is a type of margin-reduction rule different from big-edge rules. Exchange stipulates the matching ratio (for instance1:3) between futures&futures, futures&option, or option&futures. Investors who possesse the combination can apply for margin-reduction from CFFEX. After the application is approved, two single legs will be bound to a combination contract, which can benefit from the magin-reduction rule. When closing a combination contract, the combination should be broken up first before closing.

Specific to margin of combination contract, ReqCombActionInsert is added to the API to support the rules, and the core structure is CThostFtdInputCombActionField.

CTP greatly simplifies the applying and breaking flow on the basis of margin of combination contract rules by CFFEX, and developers only need to change the code slightly to realize the margin rules of combination contracts in CFFEX.

Apply / Break:

CThostFtdInputCombActionField

{

///BrokerID

TThostFtdcBrokerIDType Broker ID;

///InvestorID

TThostFtdcInvestorIDType InvestorID;

///InstrumentID

Developers need to get combination contract like a&b and then fill in this field by querying the contract (ReqQryInstrument).

TThostFtdcInstrumentIDType InstrumentID;

///Combination referring

TThostFtdcOrderRefType CombActionRef;

///UserID

TThostFtdcUserIDType UserID;

///direction

Developers need to fill in this field the first leg in a&b like combination contract(i.e. contract a).

TThostFtdcDirectionType Direction;

///Volume

Developers need to fill in this field the volume applied.

For instance, for ratio a&b 1:3, the position of a is 2 & b is 6, the filled volume is 2.

TThostFtdcVolumeType Volume;

///Direction of Combination Contract

Fill in THOST_FTDC_CMDR_Comb （for applying）

Fill in THOST_FTDC_CMDR_UnComb （for breaking）

TThostFtdcCombDirectionType CombDirection;

///HedgeFlag

TThostFtdcHedgeFlagType HedgeFlag;

© Shanghai Future Information Technology Co., Ltd

};

4.13. Data Flow files

When initiating, the CTP API may generate some flow files on the local machine to save 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 makes large amounts of reads and writes to the data flow files. If the client program does not manage the number of file handles in the system well, file handles may be used up.
2. When developing the program for multiple accounts, please pay attention not to put all account's flow files in one directory, which may cause the situation that 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 the 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

© Shanghai Future Information Technology Co., Ltd

Private.con	Received private return data flow

4.14. Flow Control

4.14.1. Query Flow Control

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

- Users can send one query operation per second at most.
- At most 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.

4.14.2. Order Flow Control

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

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

- In one session, each client can 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 error will be returned, if operations exceed the limitations. Orders will be in queuing status waiting to be handled.

4.15. Disconnection

The method OnFrontDisconnected will be invoked if a disconnection occurs. The disconnection cause will be stated in nReason parameter.

Possible Causes:

Decimal System(10D)	Hexadecimal(16H)	Explanation
4097	0x1001	Failed to read from the network.
4098	0x1002	Failed to write to the network.
8193	0x2001	Timed out receiving heart-beat.

© Shanghai Future Information Technology Co., Ltd

8194	0x2002	Timed out sending heart-beat.
8195	0x2003	Received error messages.

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

- Network issues caused the disconnection.
- The server disconnected 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 occurred.
- The amount of connections exceeded the limitation.

Heartbeat Mechanism

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

Timeout

Network delay may cause a server to disconnect from a client initiatively.

The default timeout is 120 seconds for reading, 60 seconds for writing, and 80 seconds for heartbeats.

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

© Shanghai Future Information Technology Co., Ltd