Comprehensive Transaction Platform Trade and Quotation

Application Programming Interface

Version：4.2

Publish Date: Nov. 6, 2009

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

I. List of File Change history

version	date of change	memo
V4.2	2009-11-6	Create the English Version

Index

Chapter 1. Introduction

Comprehensive Transaction Platform (CTP) ,a future trade and broker information management system, contains trade server, risk management server,settlement information management subsystem.

The API   is used to communicate with the CTP trade server. From the API, investor can receive quotation data from SHFE, DCE and CZCE, send trading directive to the three exchanges, receive corresponding response and trade status return.

1.1. Background

In 2006, after shanghai future information technology coporation completed the New Generation Exchange System (NGES) development for SHFE, we brought in the success experience to our CTP development.

In April 2007, we obtained the first order of CTP from the future broker system field in China. With our great efforts in recent three years, investors trading via CTP have expanded all over the world and the broker quantity has increased to thirty in China.

1.2. Introduction of API files

The API of CTP trade server is based on C++ library and carrys out the communication between trade client and CTP trade server. Trade clients includes CTP standard trade client (such as Q7, pobo, weisoft etc. developed by third part) free used by all investor of CTP, and trade tools only used personally (developed by investors or their partners). By using the API, trade client could insert or cancel common order and condition order, contract status fire order, query order or trade record and get the current account and position status. This library contains:

File Name	**File Description**
ThostFtdcTraderApi.h	Trading interface c++ head file
ThostFtdcMdApi.h	Quotation interface c++ head file
ThostFtdcUserApiStruct.h	Defines all data type
ThostFtdcUserApiDataType.h	Defines all data structure
thosttraderapi.dll	The dynamic link library of trading interface.
thosttraderapi.lib	
thostmduserapi.dll	The dynamic link library of quotation interface.
thostmduserapi.lib	
error.dtd	The api error code and information in xml format.
error.xml	

Note: Users of compilers MS VC 6.0，MS VC.NET 2003,etc, need toturn on the multi-thread option in compile setting.

Chapter 2. Archetecture

The communication protocol between CTP API and CTP trade server is futures TradingData Exchange protocol (FTD), an information exchange protocol based on TCP.

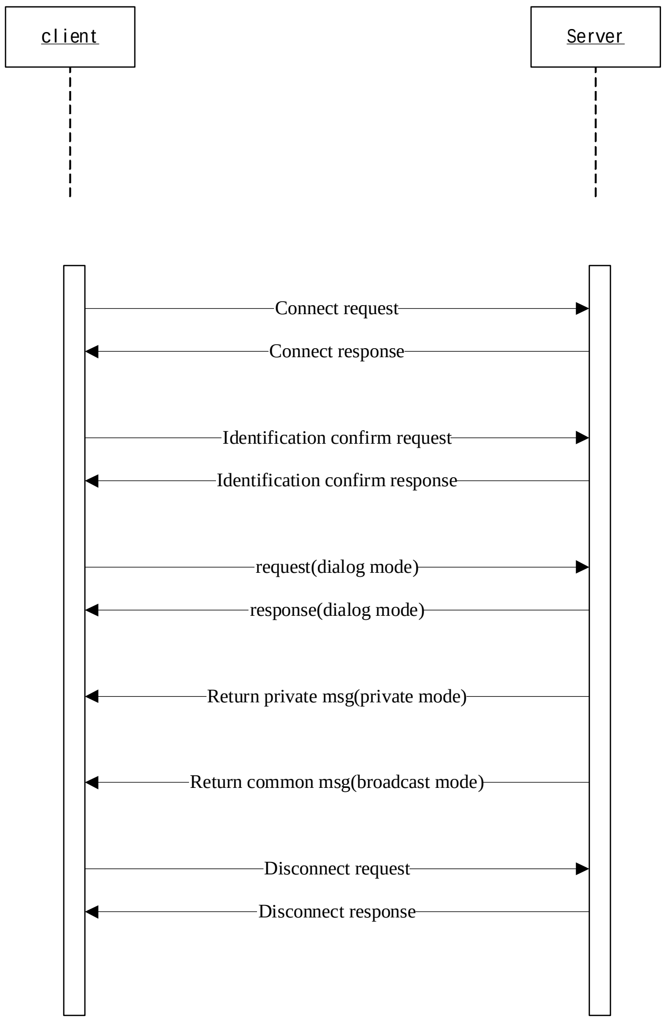
2.1. Communication Mode

In FTD protocol, communication mode includes the following three modes:

- Dialog mode, client submits a request to CTP, and CTP will return corresponding results.
- Private mode, CTP sends private messages to specific client those messages are all private notify message such as order status or trade confirmation.
- Broadcast mode, CTP publishs common information to all clients registerd to ctp.

Each communication mode is not confined to one network connection. That means, with one network connection, the client can use all the three communication modes, or several different client connection can use the same communication mode. For example, the client can use broadcast mode to receive instrument status change message, and at the same time receive its own private message such as order confirmation message.

The following diagram explains communication process of these three modes:


client
Server
Connect request
Connect response
Identification confirm request
Identification confirm response
request(dialog mode)
response(dialog mode)
Return private msg(private mode)
Return common msg(broadcast mode)
Disconnect request
Disconnect response

2.2. Data Stream

CTP support dialog, private and broadcast communication mode. With dialog communication mode, dialog data stream and query data stream could be transmitted.

Dialog and query data stream are both bi-direction data stream, the client application submit request and CTP server return response. CTP server doesn’t maintain the status of dialog and query data stream. when problems occurs, for example reconnect happens, the dialog and query data stream will be reset after the communication rebuilding and data on fly will lost .

With private communication mode, private data stream is transmitted. Private data stream is a unidirectional data stream, using it, the CTP server send private message to the corresponding client application. Private message includes risk notice, order status, order confirmation, trade confirmation. The private data stream is reliable, when the client application lost connection with CTP server, at any time in the same trading day, the client application can reconnect the CTP server with specified sequence number of its own private data flow and without any risk of lost those private trading data.

With the broadcast communication mode, public data stream is transmitted. It is a unidirectional and reliable data stream just like the private data stream, the only difference between them is the broad cast communication data will broadcast to all connecting client application. Its main useage is pulic instrument status or any public important message.

Chapter 3. Progamming Interface Types

CTP trade API provides the two interfaces, CThostFtdcTraderApi and CThostFtdcTraderSpi. The CTP quotation API provides CThostFtdcMdApi and CThostFtdcMdSpi. The four interfaces implement FTD protocol; the client could submit requests by invoking functions of the CThostFtdcXXXApi and receive the CTP response with reloaded callback functions of their own object inherited from CThostFtdcXXXSpi.

3.1. Dialog mode programming interface

Communication functions of the interface with dialog mode is usually defined as the following:

```
request：   int CThostFtdcTraderApi::ReqXXX(
                      CThostFtdcXXXField *pReqXXX,
                      int nRequestID)
            int CThostFtdcMDApi::ReqXXX(
                      CThostFtdcXXXField *pReqXXX,
                      int nRequestID)

response：  void CThostFtdcTraderSpi::OnRspXXX(
                      CThostFtdcXXXField *pRspXXX,
                      CThostFtdcRspInfoField *pRspInfo,
                      int nRequestID,
                      bool bIsLast)
```

```
void CThostFtdcMDSpi::OnRspXXX(
            CThostFtdcXXXField *pRspXXX,
            CThostFtdcRspInfoField *pRspInfo,
            int nRequestID,
            bool bIsLast)
```

The first parameter of request functions is request content and should not be empty. The second parameter is the request Id, which should be maintained by client trade application, and within one session the ID is strongly recommended be unique, when the client receive the response from the CTP server, the client could relate request and response with same request ID.

When the client receive any response from CTP server, the reloaded callback function of CThostFtdcXXXSpi will be invoked, if the response has more than one records, the reloaded callback function would be invoked repeatly until the whole message is received.

The first parameter of response functions is the data of the reponse, which usually includes the original request data. If something wrong happened or CTP can not find any record for the request, the parameter will be NULL. The second parameter is a flag used by CTP to show whether this response is one successful response. When the callback function is invoked more than one time, except the first time of the callback being invoked, this second parameter may be NULL in the following callback action. The third parameter is request ID which is same as the corresponding request. The last parameter is the end marker of the response, the value “true” manifest the current response is the last one related with the same request.

3.2. Private mode programming interface

The following example shows the usual way of defining the private interface:

*void CThostFtdcTraderSpi::OnRtnXXX(CThostFtdcXXXField *pXXX)*

*void CThostFtdcTraderSpi::OnErrRtnXXX(CThostFtdcXXXField *pXXX,*
*CThostFtdcRspInfoField *pRspInfo)*

There is no function of the quotation API interface to communicate with CTP server in private mode.

When CTP server issue return data with private data stream, the reloaded callback function of the object inherited from CThostFtdcTradeSpi will be invoked.

The first parameter of all callback functions is the return content from CTP server, the second parameter of the OnErrRtn CThostFtdcTradeSpi functions is detail error information when something is wrong.

3.3. Boadcast mode programming interface

The client application can use the following two fuctions to communication with CTP server with broadcast mode:

void CThostFtdcTraderSpi::OnRtnInstrumentStatus (

*CThostFtdcInstrumentStatusField *pInstrumentStatus)*

void CThostFtdcTraderSpi::OnRtnDepthMarketData (

*CThostFtdcDepthMarketDataField *pDepthMarketData)*

The callback function “OnRtnInstrumentStatus” is used to notify client application the status change of instruments.

The callback function “OnRtnDepthMarketData” is used by CTP to public the updated market quotation data from exchanges.

Chapter 4. Operation mode

4.1. Working thread

The CTP client process need two kind of thread, one is the application main thread and the othe is trade API working thread, if the client want to receive quotation data, another quotation API working thread is needed. API working thread links trade client and CTP server.

The trade and quotation API interface is thread-safe, the client application can use two or more working thread at the same time without need to concern about the thread conflict, the client application should process the callback message as quickly as possible to avoid any unporocessed callback message blocking this working thread. To avoid any blocked communication, the client application should use buffer layer to store all the messages received from CTP. The client application can also use such buffer to keep its own data model independence from CTP API data model.

4.2. Files in location

CTP API's dynamic link library will create some local files to store runtime data, those files has such extending file name as “.con”, The trade client application can use the first parameter of these two functions(“CreateFtdcTraderApi() or CreateFtdcMdApi()”) to specify these files’ local path.

4.3. Business terminology and interface function contrast

Type	**Business**	**Request interface**	**Response interface**	**Stream**
login	login	CThostFtdcTraderApi:: ReqUserLogin	CThostFtdcTraderSpi::OnRspUserLogin	dialog
	logout	CThostFtdcTraderApi::ReqUserLogout	CThostFtdcTraderSpi::OnRspUserLogout	dialog
	Password modification	CThostFtdcTraderApi::ReqUserPasswordUpdate	CThostFtdcTraderSpi::OnRspUserPasswordUpdate	dialog
trade	Order insertion	CThostFtdcTraderApi::ReqOrderInsert	CThostFtdcTraderSpi::OnRspOrderInsert	dialog
	Order modification	CThostFtdcTraderApi::ReqOrderAction	CThostFtdcTraderSpi::OnRspOrderAction	dialog
Private return	Trade return	N/A	CThostFtdcTraderSpi::OnRtnTrade	private
	Order return	N/A	CThostFtdcTraderSpi::OnRtnOrder	private
	Order insertion error return	N/A	CThostFtdcTraderSpi::OnErrRtnOrderInsert	private
	Order monification error return	N/A	CThostFtdcTraderSpi::OnErrRtnOrderAction	private
query	Order query	CThostFtdcTraderApi::ReqQryOrder	CThostFtdcTraderSpi::OnRspQryOrder	query
	Trade query	CThostFtdcTraderApi::ReqQryTrade	CThostFtdcTraderSpi::OnRspQryTrade	query
	Investor query	CThostFtdcTraderApi::ReqQryInvestor	CThostFtdcTraderSpi::OnRspQryInvestor	query
	Investor position query	CThostFtdcTraderApi::ReqQryInvestor Position	CThostFtdcTraderSpi::OnRspQryInvestor Position	query
	Instrument query	CThostFtdcTraderApi::ReqQryInstrument	CThostFtdcTraderSpi::OnRspQryInstrument	query

Chapter 5. CTP API specification

5.1. General rules

The client trade application follows two steps to connect and communicate with the CTP server: initialization and fuction call.

To use trade API, client trade application should program the following steps:

1. Create a “CThostFtdcTraderApi” instance.
2. Create an event hangdle instance inherited from “CThostFtdcTraderSpi” interface, and registering this instance with the “RegisterSpi” function of the “CThostFtdcTraderApi”.
3. Subscribe private stream with the “SubscribePrivateTopic” function of the “CThostFtdcTraderApi”.
4. Subscribe public stream with the “SubscribePublicTopic” function of the “CThostFtdcTraderApi”.
5. Register the trade front addresses of the CTP server with the “RegisterFront” function of the “CThostFtdcTraderApi”. The client could call the function several times, in order to establish more reliable communication; this kind of function usage is strongly recommended.
6. Start connection with CTP server using the “Init” function of the

"CThostFtdcTraderApi".

7. After the CTP server confirmed the connection, the callback function "OnFrontConnected" of the "CThostFtdcTraderSpi" interface will be invoked. In the function implementation, the client application can submit the "login" request using the "ReqUserLogin" function of the "CThostFtdcTraderApi".
8. After the CTP server confirmed the login, the callback function "OnRspUserLogin" of the "CThostFtdcTraderSpi" interface will be invoked.
9. Now, the communication between the client and CTP server is estabilished successfully, and the client trade application can use other CTP API to communicate with CTP server.

If client trade application want to use quotation API , the client application can use those steps which illustrated previous segments,except subscribing private and public stream.

There are several programming rules:

1. The parameters of all request functions should not be NULL.
2. In case the type of functions' return value is "int", value "0" means function return normally, other values represent error returns; their detail information can be foundin the "error.xml" file.

5.2. CThostFtdcTraderSpi

CTP use CThostFtdcTraderSpi as its event interface. Client trade application can inherit the function of CThostFtdcTraderSpi to receive the notification from CTP server.

5.2.1. OnFrontConnected

This function is invoked after client finished the connection with CTP server, then by inherit this function, the client could use “ReqUserLogin” to send login request.

definition：

void OnFrontConnected()；

5.2.2. OnFrontDisconnected

When the connection ended or disconnected, this function is called. If the message is left unprocessed, then the API instance will automatically reconnect with CTP server using one of the front addresses from the registed front address list.

definition：

void OnFrontDisconnected (int nReason)；

parameters：

nReason：the reason of disconnecion

0x1001 network reading failed
0x1002 network writing failed
0x2001 heartbeat receiing timeout
0x2002 heartbeat sending timeout
0x2003 received a error message

5.2.3. OnHeartBeatWarning

This function is used to indicate the long used connection is still available.

definition：

void OnHeartBeatWarning(int nTimeLapse)；

parameters：

nTimeLapse：Length of time elapsed since the last received message.

5.2.4. OnRspUserLogin

CTP server use the callback function "OnRspUserLogin" to notify the client whether the login function "OnRspUserLogin" was accepted by the server.

definition：

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

parameters：

***pRspUserLogin**：The pointer of the structure for user's login response. The following is definition of the structure:*

```
struct CThostFtdcRspUserLoginField
{
    ///trading day
    TThostFtdcDateType TradingDay;
    ///time of login
    TThostFtdcTimeType LoginTime;
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///user id
    TThostFtdcUserIDType    UserID;
```

```
        ///trade system name
        TThostFtdcSystemNameType   SystemName;
    };
```

***pRspInfo**: Pointer of the structure for system response. The following is definition of the structure:*

```
    struct CThostFtdcRspInfoField
    {
        ///error id
        TThostFtdcErrorIDType ErrorID;
        ///error information
        TThostFtdcErrorMsgType      ErrorMsg;
    };
```

5.2.5. OnRspUserLogout

CTP server use this callback function to notify the client application whether the function “OnRspUserLogout” was succeeded.

definition:

```
    void OnRspUserLogout(
                            CThostFtdcUserLogoutField *pUserLogout,
                            CThostFtdcRspInfoField *pRspInfo,
                            int nRequestID,
                            bool bIsLast);
```

parameters:

pRspUserLogout: *Pointerof the structure for user's logout response. The following is definition of the structure:*

```
    struct CThostFtdcUserLogoutField
    {
        ///broker id
        TThostFtdcBrokerIDType BrokerID;
        ///user id
        TThostFtdcUserIDType    UserID;
    };
```

5.2.6. OnRspUserPasswordUpdate

CTP server use this callback function to notify the client application whether the function “ReqUserPasswordUpdate” was succeeded.

definition:

```
void OnRspUserPasswordUpdate(
                    CThostFtdcUserPasswordUpdateField
                    *pUserPasswordUpdate,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast);
```

parameters:

pUserPasswordUpdate: *Pointer of the structure for the response of user's password modification. The following is definition of the structure:*

```
struct CThostFtdcUserPasswordUpdateField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///user id
    TThostFtdcUserIDType    UserID;
    ///old password
    TThostFtdcPasswordType OldPassword;
    ///new password
    TThostFtdcPasswordType NewPassword;
};
```

5.2.7. OnRspTradingAccountPasswordUpdate

CTP server use this callback function to notify the client application whether the function “ReqTradingAccountPasswordUpdate” has been succeeded.

definition:

```
void OnRspTradingAccountPasswordUpdate(
        CThostFtdcTradingAccountPasswordUpdateField *pTradingAccountPasswordUpdate,
        CThostFtdcRspInfoField *pRspInfo,
        int nRequestID,
        bool bIsLast)；
```

parameters：

pTradingAccountPasswordUpdate：*Pointer of the structure for the response of trading account password modification. The following is definition of the structure,*

```
struct CThostFtdcTradingAccountPasswordUpdateField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///account id
    TThostFtdcAccountIDType     AccountID;
    ///old password
    TThostFtdcPasswordType OldPassword;
    ///new password
    TThostFtdcPasswordType NewPassword;
};
```

5.2.8. OnRspError

CTP server uses this callback function to notify something is wrong in the client application’s request.

definition：

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

parameters：

pRspInfo：*Pointer of the structure for the response information. The following is definition of the structure,*

```
struct CThostFtdcRspInfoField
{
    ///error id
    TThostFtdcErrorIDType ErrorID;
    ///error information
    TThostFtdcErrorMsgType     ErrorMsg;
};
```

5.2.9. OnRspOrderInsert

CTP server use this callback function to response to the client 's "ReqOrderInsert" request.

definition:

```
void OnRspOrderInsert(
                    CThostFtdcInputOrderField *pInputOrder,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast);
```

parameters:

pInputOrder: *Pointer of the structure for the response of order inserting. The following is definition of the structure,*

```
struct CThostFtdcInputOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///order reference
    TThostFtdcOrderRefType      OrderRef;
    ///user id
    TThostFtdcUserIDType  UserID;
    /// price type of condition order
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///order direction
    TThostFtdcDirectionType     Direction;
    ///combination order's offset flag
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///combination or hedge flag
    TThostFtdcCombHedgeFlagType   CombHedgeFlag;
    ///price
    TThostFtdcPriceType     LimitPrice;
    ///volume
    TThostFtdcVolumeType  VolumeTotalOriginal;
    ///valid date
    TThostFtdcTimeConditionType     TimeCondition;
```

```
        ///GTD DATE
        TThostFtdcDateType      GTDDate;
        ///volume type
        TThostFtdcVolumeConditionType  VolumeCondition;
        ///min volume
        TThostFtdcVolumeType  MinVolume;
        ///trigger condition
        TThostFtdcContingentConditionType    ContingentCondition;
        ///stop price
        TThostFtdcPriceType     StopPrice;
        ///force close reason
        TThostFtdcForceCloseReasonType       ForceCloseReason;
        /// auto suspend flag
        TThostFtdcBoolType      IsAutoSuspend;
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
};
```

5.2.10. OnRspOrderAction

CTP server use this callback function to response to the client application 's"ReqOrderAction" request.

definition:

```
void OnRspOrderAction(
                        CThostFtdcOrderActionField *pOrderAction,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast);
```

parameters:

pOrderAction: *Pointer of the structure for the response of order action. The following is definition of the structure,*

```
struct CThostFtdcOrderActionField
{
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
```

```
///order action reference
TThostFtdcOrderActionRefType     OrderActionRef;
///order reference
TThostFtdcOrderRefType       OrderRef;
///request ID
TThostFtdcRequestIDType     RequestID;
///front ID
TThostFtdcFrontIDType FrontID;
///session ID
TThostFtdcSessionIDType      SessionID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///order system ID
TThostFtdcOrderSysIDType   OrderSysID;
///action flag
TThostFtdcActionFlagType    ActionFlag;
///price
TThostFtdcPriceType     LimitPrice;
///volume change
TThostFtdcVolumeType   VolumeChange;
///action date
TThostFtdcDateType      ActionDate;
///action time
TThostFtdcTimeType      ActionTime;
///trader ID
TThostFtdcTraderIDType       TraderID;
///install ID
TThostFtdcInstallIDType       InstallID;
///order local ID
TThostFtdcOrderLocalIDType       OrderLocalID;
///action local ID
TThostFtdcOrderLocalIDType       ActionLocalID;
///participant ID
TThostFtdcParticipantIDType       ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///business unit
TThostFtdcBusinessUnitType BusinessUnit;
///order action status
TThostFtdcOrderActionStatusType OrderActionStatus;
///user id
TThostFtdcUserIDType  UserID;
///status message
TThostFtdcErrorMsgType      StatusMsg;
```

```
};
```

5.2.11. OnRspQueryMaxOrderVolume

CTP server use this callback function to response to the client application’s “ReqQueryMaxOrderVolume” request.

definition:

```
void OnRspQueryMaxOrderVolume(
            CThostFtdcQueryMaxOrderVolumeField *pQueryMaxOrderVolume,
            CThostFtdcRspInfoField *pRspInfo,
            int nRequestID,
            bool bIsLast)；
```

parameters:

pQueryMaxOrderVolume： *Pointer of the structure for the response of ReqQueryMaxOrderVolume. The following is definition of the structure,*

```
struct CThostFtdcQueryMaxOrderVolumeField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///direction
    TThostFtdcDirectionType     Direction;
    ///offset flag
    TThostFtdcOffsetFlagType    OffsetFlag;
    ///hedge flag
    TThostFtdcHedgeFlagType     HedgeFlag;
    ///max volume
    TThostFtdcVolumeType  MaxVolume;
};
```

5.2.12. OnRspSettlementInfoConfirm

CTP server uses this callback function toresponse to the client application’s “ReqSettlementInfoConfirm” request.

definition：

```
void OnRspSettlementInfoConfirm(
            CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
            CThostFtdcRspInfoField *pRspInfo,
            int nRequestID,
            bool bIsLast)；
```

parameters：

pSettlementInfoConfirm: *Pointer of the structure for the response of ReqSettlementInfoConfirm. The following is definition of the structure,*

```
struct CThostFtdcSettlementInfoConfirmField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///confirm date
    TThostFtdcDateType      ConfirmDate;
    ///confirm time
    TThostFtdcTimeType      ConfirmTime;
};
```

5.2.13. OnRspQryOrder

CTP server uses this callback function toresponse to the client application’s “ReqQryOrder” request.

definition：

```
void OnRspQryOrder(
                CThostFtdcOrderField *pOrder,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast)；
```

parameters：

pOrder：*Pointer of the structure for the response of ReqQryOrder. The following* *is definition of the structure,*

```
struct CThostFtdcOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///order reference
    TThostFtdcOrderRefType      OrderRef;
    ///user id
    TThostFtdcUserIDType  UserID;
    ///order price type
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///direction
    TThostFtdcDirectionType     Direction;
    ///combination order's offset flag
    TThostFtdcCombOffsetFlagType    CombOffsetFlag;
    ///combination or hedge flag
    TThostFtdcCombHedgeFlagType   CombHedgeFlag;
    ///price
    TThostFtdcPriceType     LimitPrice;
    ///volume
    TThostFtdcVolumeType  VolumeTotalOriginal;
    ///valid date type
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD DATE
    TThostFtdcDateType      GTDDate;
    ///volume condition
    TThostFtdcVolumeConditionType   VolumeCondition;
    ///min volume
    TThostFtdcVolumeType  MinVolume;
    ///trigger condition
    TThostFtdcContingentConditionType    ContingentCondition;
    ///stop price
    TThostFtdcPriceType     StopPrice;
    ///force close reason
    TThostFtdcForceCloseReasonType       ForceCloseReason;
    /// auto suspend flag
    TThostFtdcBoolType      IsAutoSuspend;
    ///business unit
    TThostFtdcBusinessUnitType BusinessUnit;
    ///request ID
```

```
TThostFtdcRequestIDType     RequestID;
///order local ID
TThostFtdcOrderLocalIDType       OrderLocalID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///participant ID
TThostFtdcParticipantIDType       ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///exchange instrument ID
TThostFtdcExchangeInstIDType    ExchangeInstID;
///trader ID
TThostFtdcTraderIDType      TraderID;
///install ID
TThostFtdcInstallIDType      InstallID;
///order submit status
TThostFtdcOrderSubmitStatusType       OrderSubmitStatus;
///order notify sequence
TThostFtdcSequenceNoType  NotifySequence;
///trading day
TThostFtdcDateType      TradingDay;
///settlement ID
TThostFtdcSettlementIDType SettlementID;
///order system ID
TThostFtdcOrderSysIDType   OrderSysID;
///order source
TThostFtdcOrderSourceType OrderSource;
///order status
TThostFtdcOrderStatusType   OrderStatus;
///order type
TThostFtdcOrderTypeType     OrderType;
///volume traded
TThostFtdcVolumeType   VolumeTraded;
/// total volume
TThostFtdcVolumeType   VolumeTotal;
///insert date
TThostFtdcDateType      InsertDate;
///insert time
TThostFtdcTimeType      InsertTime;
///active time
TThostFtdcTimeType      ActiveTime;
///suspend time
TThostFtdcTimeType      SuspendTime;
///update time
```

```
        TThostFtdcTimeType      UpdateTime;
        ///cancel time
        TThostFtdcTimeType      CancelTime;
        ///active trader ID
        TThostFtdcTraderIDType      ActiveTraderID;
        ///clear participant ID
        TThostFtdcParticipantIDType       ClearingPartID;
        ///sequence No.
        TThostFtdcSequenceNoType  SequenceNo;
        ///front ID
        TThostFtdcFrontIDType FrontID;
        ///session ID
        TThostFtdcSessionIDType     SessionID;
        ///user product information
        TThostFtdcProductInfoType  UserProductInfo;
        ///status message
        TThostFtdcErrorMsgType     StatusMsg;
    };
```

5.2.14. OnRspQryTrade

CTP server uses this callback function to response to the client application’s “ReqQryTrade”request.

definition:

```
    void OnRspQryTrade(
                        CThostFtdcTradeField *pTrade,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

parameters:

pTrade: *Pointer of the structure for the response of ReqQryTrade. The following is definition of the structure,*

```
    struct CThostFtdcTradeField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///instrument ID
```

```
TThostFtdcInstrumentIDType InstrumentID;
///order reference
TThostFtdcOrderRefType      OrderRef;
///user id
TThostFtdcUserIDType  UserID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///trade ID
TThostFtdcTradeIDType TradeID;
///direction
TThostFtdcDirectionType     Direction;
///order system ID
TThostFtdcOrderSysIDType   OrderSysID;
///participant ID
TThostFtdcParticipantIDType      ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///trading role
TThostFtdcTradingRoleType  TradingRole;
///exchange instrument ID
TThostFtdcExchangeInstIDType    ExchangeInstID;
///offset flag
TThostFtdcOffsetFlagType    OffsetFlag;
///hedge flag
TThostFtdcHedgeFlagType    HedgeFlag;
///price
TThostFtdcPriceType    Price;
///volume
TThostFtdcVolumeType  Volume;
///trade date
TThostFtdcDateType     TradeDate;
///trade time
TThostFtdcTimeType     TradeTime;
///trade type
TThostFtdcTradeTypeType    TradeType;
///price source
TThostFtdcPriceSourceType  PriceSource;
///trader ID
TThostFtdcTraderIDType      TraderID;
///order local ID
TThostFtdcOrderLocalIDType      OrderLocalID;
///clear participant ID
TThostFtdcParticipantIDType      ClearingPartID;
///business unit
```

```
        TThostFtdcBusinessUnitType BusinessUnit;
        ///sequence No.
        TThostFtdcSequenceNoType  SequenceNo;
        ///trading day
        TThostFtdcDateType      TradingDay;
        ///settlement ID
        TThostFtdcSettlementIDType SettlementID;
    };
```

5.2.15. OnRspQryInvestor

CTP server uses this callback function to response to the client application's "ReqQryInvestor"request.

definition：

```
    void OnRspQry Investor (
                        CThostFtdcInvestorField *pInvestor,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast)；
```

parameters：

pInvestor：*Pointer of the structure for the response of ReqQryInvestor. The following is definition of the structure,*

```
    struct CThostFtdcInvestorField
    {
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor group ID
        TThostFtdcInvestorIDType    InvestorGroupID;
        ///investor name
        TThostFtdcPartyNameType   InvestorName;
        ///Identified Card Type
        TThostFtdcIdCardTypeType   IdentifiedCardType;
        ///Identified Card No.
        TThostFtdcIdentifiedCardNoType  IdentifiedCardNo;
        ///is active
        TThostFtdcBoolType      IsActive;
    };
```

5.2.16. OnRspQryInvestorPosition

CTP server uses this callback function to response to the client application's "ReqQryInvestorPosition"request.

definition:

```
    void OnRspQry InvestorPosition(
                                CThostFtdcInvestorPositionField *pInvestorPosition,
                                CThostFtdcRspInfoField *pRspInfo,
                                int nRequestID,
                                bool bIsLast);
```

parameters:

pInvestorPosition: *Pointer of the structure for the response of ReqQryInvestorPosition. The following is definition of the structure,*

```
    struct CThostFtdcInvestorPositionField
    {
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///position direction
        TThostFtdcPosiDirectionType       PosiDirection;
        ///hedge flag
        TThostFtdcHedgeFlagType    HedgeFlag;
        ///position date
        TThostFtdcPositionDateType PositionDate;
        ///position of last trading day
        TThostFtdcVolumeType  YdPosition;
```

```
///position
TThostFtdcVolumeType  Position;
///long frozen
TThostFtdcVolumeType  LongFrozen;
///short frozen
TThostFtdcVolumeType  ShortFrozen;
///long frozen amount
TThostFtdcMoneyType   LongFrozenAmount;
///short frozen amount
TThostFtdcMoneyType   ShortFrozenAmount;
///open volume
TThostFtdcVolumeType  OpenVolume;
///close volume
TThostFtdcVolumeType  CloseVolume;
///open amount
TThostFtdcMoneyType   OpenAmount;
///close amount
TThostFtdcMoneyType   CloseAmount;
///position cost
TThostFtdcMoneyType   PositionCost;
///previous margin
TThostFtdcMoneyType   PreMargin;
///used margin
TThostFtdcMoneyType   UseMargin;
///frozen margin
TThostFtdcMoneyType   FrozenMargin;
///frozen cash
TThostFtdcMoneyType   FrozenCash;
///frozen commission
TThostFtdcMoneyType   FrozenCommission;
///cash in
TThostFtdcMoneyType   CashIn;
///commission
TThostFtdcMoneyType   Commission;
///close profit
TThostFtdcMoneyType   CloseProfit;
///position profit
TThostFtdcMoneyType   PositionProfit;
///previous settlement price
TThostFtdcPriceType   PreSettlementPrice;
///settlement price
TThostFtdcPriceType   SettlementPrice;
///trading day
TThostFtdcDateType    TradingDay;
```

```
    ///settlement ID
    TThostFtdcSettlementIDType SettlementID;
};
```

5.2.17. OnRspQryTradingAccount

CTP server uses this callback function to response to the client application 's "ReqQryTradingAccount"request.

definition:

```
void OnRspQryTradingAccount(
                CThostFtdcTradingAccountField *pTradingAccount,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast);
```

parameters:

pTradingAccount: *Pointer of the structure for the response of ReqQryTradingAccount. The following is definition of the structure,*

```
struct CThostFtdcTradingAccountField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///account id
    TThostFtdcAccountIDType     AccountID;
    ///previous mortgage
    TThostFtdcMoneyType   PreMortgage;
    ///previous credit
    TThostFtdcMoneyType   PreCredit;
    ///previous deposit
    TThostFtdcMoneyType   PreDeposit;
    ///previous balance
    TThostFtdcMoneyType   PreBalance;
```

```
///premargin
TThostFtdcMoneyType   PreMargin;
///interest base
TThostFtdcMoneyType   InterestBase;
///interest
TThostFtdcMoneyType   Interest;
///deposit
TThostFtdcMoneyType   Deposit;
///withdraw
TThostFtdcMoneyType   Withdraw;
///frozen margin
TThostFtdcMoneyType   FrozenMargin;
///frozen cash
TThostFtdcMoneyType   FrozenCash;
///frozen commission
TThostFtdcMoneyType   FrozenCommission;
///current margin
TThostFtdcMoneyType   CurrMargin;
///cash in
TThostFtdcMoneyType   CashIn;
///commission
TThostFtdcMoneyType   Commission;
///close profit
TThostFtdcMoneyType   CloseProfit;
///position profit
TThostFtdcMoneyType   PositionProfit;
///balance
TThostFtdcMoneyType   Balance;
///available
TThostFtdcMoneyType   Available;
```

```
        ///withdraw quota
        TThostFtdcMoneyType   WithdrawQuota;
        ///reserve
        TThostFtdcMoneyType   Reserve;
        ///trading day
        TThostFtdcDateType     TradingDay;
        ///settlement ID
        TThostFtdcSettlementIDType SettlementID;
        ///credit
        TThostFtdcMoneyType   Credit;
        ///Mortgage
        TThostFtdcMoneyType   Mortgage;
        ///excahnge margin
        TThostFtdcMoneyType   ExchangeMargin;
    };
```

5.2.18. OnRspQryTradingCode

CTP server uses this callback function toresponse to the client application's "ReqQryTradingCode"request.

definition:

```
        void OnRspQryTradingCode(
                        CThostFtdcTradingCodeField *pTradingCode,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast) ;
```

parameters:

pTradingCode: *Pointer of the structure for the response of ReqQryTradingCode. The following is definition of the structure,*

```
    struct CThostFtdcTradingCodeField
```

```
{
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
    ///trading code
    TThostFtdcClientIDTypeClientID;
    ///is active
    TThostFtdcBoolType      IsActive;
};
```

5.2.19. OnRspQryExchange

CTP server uses this callback function to reponse to the client application's "ReqQryExchange"request.

definition:

```
void OnRspQryExchange(
                CThostFtdcExchangeField *pExchange,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast) ;
```

parameters:

pExchange: *Pointer of the structure for the response of ReqQryExchange. The following is definition of the structure,*

```
struct CThostFtdcExchangeField
{
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
```

```
        ///exchange name
        TThostFtdcExchangeNameType     ExchangeName;
        ///exchange property
        TThostFtdcExchangePropertyTypeExchangeProperty;
    };
```

5.2.20. OnRspQryInstrument

CTP server uses this callback function toreponse tto he client application’s “ReqQryInstrument”request.

definition:

```
    void OnRspQryInstrument(
                        CThostFtdcInstrumentField *pInstrument,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast) ;
```

parameters:

pInstrument: *Pointer of the structure for the response of ReqQryInstrument. The following is definition of the structure,*

```
    struct CThostFtdcInstrumentField
    {
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///exchange ID
        TThostFtdcExchangeIDType  ExchangeID;
        ///instrument name
        TThostFtdcInstrumentNameType    InstrumentName;
        ///exchange instrument ID
        TThostFtdcExchangeInstIDType    ExchangeInstID;
        ///product ID
```

```
TThostFtdcInstrumentIDType ProductID;
///product class
TThostFtdcProductClassType ProductClass;
///delivery year
TThostFtdcYearType      DeliveryYear;
///delivery month
TThostFtdcMonthType     DeliveryMonth;
///max volume for market order
TThostFtdcVolumeType    MaxMarketOrderVolume;
///min volume for market order
TThostFtdcVolumeType    MinMarketOrderVolume;
///max volume for limit order
TThostFtdcVolumeType    MaxLimitOrderVolume;
///min volume for limit order
TThostFtdcVolumeType    MinLimitOrderVolume;
///volume multiple of instrument
TThostFtdcVolumeMultipleType    VolumeMultiple;
///price tick
TThostFtdcPriceType     PriceTick;
///create date
TThostFtdcDateType      CreateDate;
///open date
TThostFtdcDateType      OpenDate;
///expire date
TThostFtdcDateType      ExpireDate;
///start delivery date
TThostFtdcDateType      StartDelivDate;
///end delivery date
TThostFtdcDateType      EndDelivDate;
///instrument life phase
```

```
        TThostFtdcInstLifePhaseTypeInstLifePhase;
        ///is trading
        TThostFtdcBoolType      IsTrading;
        ///position type
        TThostFtdcPositionTypeType PositionType;
        ///position date type
        TThostFtdcPositionDateTypeType  PositionDateType;
        ///long margin ratio
        TThostFtdcRatioType     LongMarginRatio;
        ///short margin ratio
        TThostFtdcRatioType     ShortMarginRatio;
    };
```

5.2.21. OnRspQryDepthMarketData

CTP server uses this callback function to reponse the client application’s “ReqQryDepthMarketData” request.

definition：

```
    void OnRspQryDepthMarketData(
                        CThostFtdcDepthMarketDataField *pDepthMarketData,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast) ；
```

parameters：

pDepthMarketData： *Pointer of the structure for the response of ReqQryDepthMarketData. The following is definition of the structure,*

```
    struct CThostFtdcDepthMarketDataField
    {
        ///trading day
        TThostFtdcDateType      TradingDay;
```

```
///instrument ID
TThostFtdcInstrumentIDType InstrumentID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///exchange instrument ID
TThostFtdcExchangeInstIDType    ExchangeInstID;
///last price
TThostFtdcPriceType     LastPrice;
///previous settlement price
TThostFtdcPriceType     PreSettlementPrice;
///previous close price
TThostFtdcPriceType     PreClosePrice;
///previous open volume
TThostFtdcLargeVolumeType PreOpenInterest;
///open price
TThostFtdcPriceType     OpenPrice;
///highest price
TThostFtdcPriceType     HighestPrice;
///lowest price
TThostFtdcPriceType     LowestPrice;
///trade volume
TThostFtdcVolumeType  Volume;
///turnover
TThostFtdcMoneyType   Turnover;
///open interest
TThostFtdcLargeVolumeType OpenInterest;
///close Price
TThostFtdcPriceType     ClosePrice;
///settlement price
TThostFtdcPriceType     SettlementPrice;
```

```
///upper limit price
TThostFtdcPriceType     UpperLimitPrice;
///lower limit price
TThostFtdcPriceType     LowerLimitPrice;
///pre-delta
TThostFtdcRatioType     PreDelta;
///current delta
TThostFtdcRatioType     CurrDelta;
///update time
TThostFtdcTimeType      UpdateTime;
///Update Millisecond
TThostFtdcMillisecType UpdateMillisec;
///the first bid price
TThostFtdcPriceType     BidPrice1;
///the first bid volume
TThostFtdcVolumeType  BidVolume1;
///the first ask price
TThostFtdcPriceType     AskPrice1;
///the first ask volume
TThostFtdcVolumeType  AskVolume1;
///the second bid price
TThostFtdcPriceType     BidPrice2;
///the second bid volume
TThostFtdcVolumeType  BidVolume2;
///the second ask price
TThostFtdcPriceType     AskPrice2;
///the second ask volume
TThostFtdcVolumeType  AskVolume2;
///the third bid price
TThostFtdcPriceType     BidPrice3;
```

```
    ///the third bid volume
    TThostFtdcVolumeType  BidVolume3;
    ///the third ask price
    TThostFtdcPriceType   AskPrice3;
    ///the third ask volume
    TThostFtdcVolumeType  AskVolume3;
    ///the forth bid price
    TThostFtdcPriceType   BidPrice4;
    ///the forth bid volume
    TThostFtdcVolumeType  BidVolume4;
    ///the forth ask price
    TThostFtdcPriceType   AskPrice4;
    ///the forth ask volume
    TThostFtdcVolumeType  AskVolume4;
    ///the fifth bid price
    TThostFtdcPriceType   BidPrice5;
    ///the fifth bid volume
    TThostFtdcVolumeType  BidVolume5;
    ///the fifth ask price
    TThostFtdcPriceType   AskPrice5;
    ///the fifth ask volume
    TThostFtdcVolumeType  AskVolume5;
};
```

5.2.22. OnRspQrySettlementInfo

CTP server uses this callback function to response to the client application's "ReqQrySettlementInfo" request.

definition:

```
void OnRspQrySettlementInfo(
                        CThostFtdcSettlementInfoField *pSettlementInfo,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast) ;
```

parameters:

pSettlementInfo: *Pointer of the structure for the response of ReqQrySettlementInfo. The following is definition of the structure,*

```
struct CThostFtdcSettlementInfoField
{
    ///trading day
    TThostFtdcDateType      TradingDay;
    ///settlement ID
    TThostFtdcSettlementIDType SettlementID;
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///sequence No.
    TThostFtdcSequenceNoType  SequenceNo;
    ///content
    TThostFtdcContentType Content;
};
```

5.2.23. OnRspQryInvestorPositionDetail

CTP server uses this callback function toresponse to the client application's “ReqQryInvestorPositionDetail” request.

definition:

```
void OnRspQryInvestorPositionDetail(
                        CThostFtdcInvestorPositionDetailField *pInvestorPositionDetail,
                        CThostFtdcRspInfoField *pRspInfo,
                        int nRequestID,
                        bool bIsLast) ;
```

parameters:

pInvestorPositionDetail: *Pointer of the structure for the response of ReqQryInvestorPositionDetail. The following is definition of the structure,*

```
struct CThostFtdcInvestorPositionDetailField
{
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///hedge flag
    TThostFtdcHedgeFlagType     HedgeFlag;
    ///direction
    TThostFtdcDirectionType     Direction;
    ///open date
    TThostFtdcDateType      OpenDate;
    ///trade ID
    TThostFtdcTradeIDType TradeID;
    ///volume
    TThostFtdcVolumeType  Volume;
    ///open price
    TThostFtdcPriceType     OpenPrice;
    ///trading day
    TThostFtdcDateType      TradingDay;
    ///settlement ID
    TThostFtdcSettlementIDType SettlementID;
    ///trade type
    TThostFtdcTradeTypeType     TradeType;
    ///combination instrument ID
    TThostFtdcInstrumentIDType CombInstrumentID;
};
```

5.2.24. OnRspQryNotice

CTP server uses this callback function to reponse to the client application’s “ReqQryNotice” request.

definition：

```
void OnRspQryNotice(
                    CThostFtdcNoticeField *pNotice,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast)；
```

parameters：

pNotice：*Pointer of the structure for the response of ReqQryNotice. The following is definition of the structure,*

```
struct CThostFtdcNoticeField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///content
    TThostFtdcContentType Content;
    ///Sequence Label of broker notice
    TThostFtdcSequenceLabelType      SequenceLabel;
};
```

5.2.25. OnRspQryInstrument

CTP server uses this callback function to reponse to the client application's "ReqQryInstrument" request.

definition:

```
void OnRspQryInstrument(
                    CThostFtdcInstrumentField *pInstrument,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID,
                    bool bIsLast);
```

parameters:

pRspInstrument: *Pointer of the structure for the response of ReqQryInstrument. The following is definition of the structure,*

```
struct CThostFtdcInstrumentField
{
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
    ///instrument name
    TThostFtdcInstrumentNameType   InstrumentName;
    ///exchange instrument ID
    TThostFtdcExchangeInstIDType   ExchangeInstID;
    ///product ID
    TThostFtdcInstrumentIDType ProductID;
    ///product class
    TThostFtdcProductClassType ProductClass;
    ///delivery year
    TThostFtdcYearType     DeliveryYear;
    ///delivery month
    TThostFtdcMonthType   DeliveryMonth;
    ///max volume for market order
    TThostFtdcVolumeType  MaxMarketOrderVolume;
    ///min volume for market order
    TThostFtdcVolumeType  MinMarketOrderVolume;
    ///max volume for limit order
    TThostFtdcVolumeType  MaxLimitOrderVolume;
    ///min volume for limit order
    TThostFtdcVolumeType  MinLimitOrderVolume;
```

```
        ///volume multiple of instrument
        TThostFtdcVolumeMultipleType    VolumeMultiple;
        ///price tick
        TThostFtdcPriceType    PriceTick;
        ///create date
        TThostFtdcDateType    CreateDate;
        ///open date
        TThostFtdcDateType    OpenDate;
        ///expire date
        TThostFtdcDateType    ExpireDate;
        ///start delivery date
        TThostFtdcDateType    StartDelivDate;
        ///end delivery date
        TThostFtdcDateType    EndDelivDate;
        ///instrument life phase
        TThostFtdcInstLifePhaseTypeInstLifePhase;
        ///is trading
        TThostFtdcBoolType    IsTrading;
        ///position type
        TThostFtdcPositionTypeType PositionType;
        ///position date type
        TThostFtdcPositionDateTypeType PositionDateType;
    };
```

5.2.26. OnRtnTrade

CTP server uses this callback function tonotify the client application when trade has been finished.

definition：

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

parameters：

pTrade：*Pointer of the structure for the trade information. The following is definition of the structure,*

```
    struct CThostFtdcTradeField
    {
        ///broker id
        TThostFtdcBrokerIDType    BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
```

```
///instrument ID
TThostFtdcInstrumentIDType InstrumentID;
///order reference
TThostFtdcOrderRefType      OrderRef;
///user id
TThostFtdcUserIDType  UserID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///trade ID
TThostFtdcTradeIDType TradeID;
///direction
TThostFtdcDirectionType      Direction;
///order system ID
TThostFtdcOrderSysIDType   OrderSysID;
///participant ID
TThostFtdcParticipantIDType       ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///trading role
TThostFtdcTradingRoleType  TradingRole;
///exchange instrument ID
TThostFtdcExchangeInstIDType    ExchangeInstID;
///offset flag
TThostFtdcOffsetFlagType     OffsetFlag;
///hedge flag
TThostFtdcHedgeFlagType    HedgeFlag;
///price
TThostFtdcPriceType     Price;
///volume
TThostFtdcVolumeType  Volume;
///trade date
TThostFtdcDateType      TradeDate;
///trade time
TThostFtdcTimeType      TradeTime;
///trade type
TThostFtdcTradeTypeType     TradeType;
///price source
TThostFtdcPriceSourceType  PriceSource;
///trader ID
TThostFtdcTraderIDType      TraderID;
///order local ID
TThostFtdcOrderLocalIDType       OrderLocalID;
///clear participant ID
TThostFtdcParticipantIDType       ClearingPartID;
```

```
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///sequence No.
        TThostFtdcSequenceNoType  SequenceNo;
        ///trading day
        TThostFtdcDateType      TradingDay;
        ///settlement ID
        TThostFtdcSettlementIDType SettlementID;
    };
```

5.2.27. OnRtnOrder

CTP server uses this callback function to notify the client application about change of order status

definition:

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

parameters:

pOrder: *Pointer of the structure for the order information. The following is definition of the structure,*

```
    struct CThostFtdcOrderField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///order reference
        TThostFtdcOrderRefType      OrderRef;
        ///user id
        TThostFtdcUserIDType  UserID;
        ///order price type
        TThostFtdcOrderPriceTypeType    OrderPriceType;
        ///direction
        TThostFtdcDirectionType      Direction;
        ///combination order's offset flag
        TThostFtdcCombOffsetFlagType   CombOffsetFlag;
        ///combination or hedge flag
        TThostFtdcCombHedgeFlagType   CombHedgeFlag;
```

```
///price
TThostFtdcPriceType     LimitPrice;
///volume
TThostFtdcVolumeType  VolumeTotalOriginal;
///    valid date
TThostFtdcTimeConditionType      TimeCondition;
///GTD DATE
TThostFtdcDateType      GTDDate;
///volume condition
TThostFtdcVolumeConditionType  VolumeCondition;
///min volume
TThostFtdcVolumeType  MinVolume;
///trigger condition
TThostFtdcContingentConditionType    ContingentCondition;
///stop price
TThostFtdcPriceType     StopPrice;
///force close reason
TThostFtdcForceCloseReasonType       ForceCloseReason;
/// auto suspend flag
TThostFtdcBoolType      IsAutoSuspend;
///business unit
TThostFtdcBusinessUnitType BusinessUnit;
///request ID
TThostFtdcRequestIDType     RequestID;
///order local ID
TThostFtdcOrderLocalIDType      OrderLocalID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///participant ID
TThostFtdcParticipantIDType       ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///exchange instrument ID
TThostFtdcExchangeInstIDType    ExchangeInstID;
///trader ID
TThostFtdcTraderIDType      TraderID;
///install ID
TThostFtdcInstallIDType       InstallID;
///order submit status
TThostFtdcOrderSubmitStatusType       OrderSubmitStatus;
///notify sequence
TThostFtdcSequenceNoType  NotifySequence;
///trading day
TThostFtdcDateType      TradingDay;
```

```
};

    ///settlement ID
    TThostFtdcSettlementIDType SettlementID;
    ///order system ID
    TThostFtdcOrderSysIDType   OrderSysID;
    ///order source
    TThostFtdcOrderSourceType OrderSource;
    ///order status
    TThostFtdcOrderStatusType   OrderStatus;
    ///order type
    TThostFtdcOrderTypeType     OrderType;
    ///volume traded
    TThostFtdcVolumeType   VolumeTraded;
    ///volume total
    TThostFtdcVolumeType   VolumeTotal;
    ///insert date
    TThostFtdcDateType      InsertDate;
    ///insert time
    TThostFtdcTimeType      InsertTime;
    ///active time
    TThostFtdcTimeType      ActiveTime;
    ///suspend time
    TThostFtdcTimeType      SuspendTime;
    ///update time
    TThostFtdcTimeType      UpdateTime;
    ///cancel time
    TThostFtdcTimeType      CancelTime;
    ///active trader ID
    TThostFtdcTraderIDType       ActiveTraderID;
    ///clear participant ID
    TThostFtdcParticipantIDType        ClearingPartID;
    ///sequence No.
    TThostFtdcSequenceNoType  SequenceNo;
    ///front ID
    TThostFtdcFrontIDType FrontID;
    ///session ID
    TThostFtdcSessionIDType      SessionID;
    ///user product information
    TThostFtdcProductInfoType  UserProductInfo;
    ///status message
    TThostFtdcErrorMsgType      StatusMsg;
};
```

5.2.28. OnErrRtnOrderInsert

.

This callback function is used to notify the client application about the failure of the validation of ctp server or exchange.

definition:

```
void OnErrRtnOrderInsert(
                        CThostFtdcInputOrderField *pInputOrder,
                        CThostFtdcRspInfoField *pRspInfo);
```

parameters:

pInputOrder: *Pointer of the structure for the order insertion information including the response from server. The following is definition of the structure,*

```
struct CThostFtdcInputOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///order reference
    TThostFtdcOrderRefType      OrderRef;
    ///user id
    TThostFtdcUserIDType  UserID;
    ///order price type
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///direction
    TThostFtdcDirectionType      Direction;
    ///combination order's offset flag
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///combination or hedge flag
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///price
    TThostFtdcPriceType    LimitPrice;
    ///volume
    TThostFtdcVolumeType  VolumeTotalOriginal;
    ///valid date
    TThostFtdcTimeConditionType      TimeCondition;
```

```
        ///GTD DATE
        TThostFtdcDateType      GTDDate;
        ///volume condition
        TThostFtdcVolumeConditionType   VolumeCondition;
        ///min volume
        TThostFtdcVolumeType  MinVolume;
        ///trigger condition
        TThostFtdcContingentConditionType    ContingentCondition;
        ///stop price
        TThostFtdcPriceType     StopPrice;
        ///force close reason
        TThostFtdcForceCloseReasonType       ForceCloseReason;
        /// auto suspend flag
        TThostFtdcBoolType      IsAutoSuspend;
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
    };
```

5.2.29. OnErrRtnOrderAction

This callback function is used to notify the client application about the failure of the validation of ctp server or exchange.

definition:

```
    void OnErrRtnOrderAction (
                                CThostFtdcOrderActionField *pOrderAction,
                                CThostFtdcRspInfoField *pRspInfo);
```

parameters:

pOrderAction: *Pointer of the structure for the order action information including the response from server. The following is definition of the structure,*

```
    struct CThostFtdcOrderActionField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
```

```
TThostFtdcInvestorIDType	InvestorID;
///order action reference
TThostFtdcOrderActionRefType	OrderActionRef;
///order reference
TThostFtdcOrderRefType	OrderRef;
///request ID
TThostFtdcRequestIDType	RequestID;
///front ID
TThostFtdcFrontIDType FrontID;
///session ID
TThostFtdcSessionIDType	SessionID;
///exchange ID
TThostFtdcExchangeIDType	ExchangeID;
///order system ID
TThostFtdcOrderSysIDType	OrderSysID;
///action flag
TThostFtdcActionFlagType	ActionFlag;
///price
TThostFtdcPriceType	LimitPrice;
///volume change
TThostFtdcVolumeType	VolumeChange;
///action date
TThostFtdcDateType	ActionDate;
///action time
TThostFtdcTimeType	ActionTime;
///trader ID
TThostFtdcTraderIDType	TraderID;
///install ID
TThostFtdcInstallIDType	InstallID;
///order local ID
TThostFtdcOrderLocalIDType	OrderLocalID;
///action local ID
TThostFtdcOrderLocalIDType	ActionLocalID;
///participant ID
TThostFtdcParticipantIDType	ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///business unit
TThostFtdcBusinessUnitType BusinessUnit;
///order action status
TThostFtdcOrderActionStatusType OrderActionStatus;
///user id
TThostFtdcUserIDType	UserID;
///status message
```

```
        TThostFtdcErrorMsgType      StatusMsg;
    };
```

5.2.30. OnRspQrySettlementInfoConfirm

CTP server uses this callback function to notify the client application the sucess of “ReqQrySettlementInfoConfirm”

definition:

```
void OnRspQrySettlementInfoConfirm(
                CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID,
                bool bIsLast);
```

parameters:

pSettlementInfoConfirm: *Pointer of the structure for the response of ReqQrySettlementInfoConfirm. The following is definition of the structure,*

```
  struct CThostFtdcSettlementInfoConfirmField
  {
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///confirm date
    TThostFtdcDateType      ConfirmDate;
    ///confirm time
    TThostFtdcTimeType      ConfirmTime;
  };;
```

5.2.31. RspQryParkedOrder

CTP server uses this callback function to response to parked order query..

definition:

```
    void OnRspQryParkedOrder(CThostFtdcParkedOrderField *pParkedOrder,
                 CThostFtdcRspInfoField *pRspInfo,
                 int nRequestID,
                    bool bIsLast);
```

parameters:

pParkedOrder: *Pointer of the structure for the response of ReqQryParkedOrder. The following is definition of the structure,*

```
    struct CThostFtdcParkedOrderField
    {
        ///broker id
        TThostFtdcBrokerIDType       BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType     InvestorID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///order reference
        TThostFtdcOrderRefType       OrderRef;
        ///user id
        TThostFtdcUserIDType  UserID;
        ///order price type
        TThostFtdcOrderPriceTypeType    OrderPriceType;
        ///direction
        TThostFtdcDirectionType      Direction;
        ///combination order's offset flag
        TThostFtdcCombOffsetFlagType    CombOffsetFlag;
        ///combination or hedge flag
        TThostFtdcCombHedgeFlagType   CombHedgeFlag;
        ///price
        TThostFtdcPriceType     LimitPrice;
        ///volume
        TThostFtdcVolumeType  VolumeTotalOriginal;
        ///valid date
        TThostFtdcTimeConditionType      TimeCondition;
        ///GTD DATE
        TThostFtdcDateType      GTDDate;
        ///volume condition
        TThostFtdcVolumeConditionType   VolumeCondition;
        ///min volume
        TThostFtdcVolumeType  MinVolume;
        ///trigger condition
        TThostFtdcContingentConditionType    ContingentCondition;
        ///stop price
        TThostFtdcPriceType     StopPrice;
        ///force close reason
        TThostFtdcForceCloseReasonType       ForceCloseReason;
        /// auto suspend flag
        TThostFtdcBoolType      IsAutoSuspend;
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
        ///user force close flag
        TThostFtdcBoolType      UserForceClose;
        ///exchange ID
        TThostFtdcExchangeIDType  ExchangeID;
        ///parked order system ID
        TThostFtdcParkedOrderIDType     ParkedOrderID;
        ///user type
        TThostFtdcUserTypeType      UserType;
        ///parked order status
        TThostFtdcParkedOrderStatusType     Status;
    };
```

5.2.32. RspQryParkedOrderAction

CTP server use this callback function to response to the query of "RspQryParkedOrderAction".

definition:

```
    void OnRspQryParkedOrderAction(
                CThostFtdcParkedOrderActionField *pParkedOrderAction,
                CThostFtdcRspInfoField *pRspInfo,
                  int nRequestID, bool bIsLast);
```

parameters:

pParkedOrderAction: *Pointer of the structure for the response of ReqQryParkedOrderAction. The following is definition of the structure,*

```
    struct CThostFtdcParkedOrderActionField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///order action reference
        TThostFtdcOrderActionRefType    OrderActionRef;
        ///order reference
        TThostFtdcOrderRefType      OrderRef;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
        ///front ID
        TThostFtdcFrontIDType FrontID;
        ///session ID
        TThostFtdcSessionIDType     SessionID;
        ///exchange ID
        TThostFtdcExchangeIDType  ExchangeID;
        ///order system ID
        TThostFtdcOrderSysIDType   OrderSysID;
        ///action flag
        TThostFtdcActionFlagType    ActionFlag;
        ///price
        TThostFtdcPriceType     LimitPrice;
        ///volume change
        TThostFtdcVolumeType  VolumeChange;
        ///user id
        TThostFtdcUserIDType  UserID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///parked order action ID
        TThostFtdcParkedOrderActionIDType  ParkedOrderActionID;
        ///user type
        TThostFtdcUserTypeType      UserType;
        ///parked order action status
        TThostFtdcParkedOrderStatusType      Status;
    };
```

5.2.33. RspQryInvestorPositionCombineDetail

CTP server uses this callback function to response to the query of investor combination instrument ‘s position..

definition:

```
void OnRspQryInvestorPositionCombineDetail(
            CThostFtdcInvestorPositionCombineDetailField *pInvestorPositionCombineDetail,
            CThostFtdcRspInfoField *pRspInfo,
            int nRequestID, bool bIsLast);
```

parameters:

pInvestorPositionCombineDetail: *Pointer of the structure for the response of*

ReqQryInvestorPositionCombineDetail. The following is definition of the structure,

```
struct CThostFtdcInvestorPositionCombineDetailField
{
    ///trading day
    TThostFtdcDateType      TradingDay;
    ///open date
    TThostFtdcDateType      OpenDate;
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
    ///settlement ID
    TThostFtdcSettlementIDType SettlementID;
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///combination trade ID
    TThostFtdcTradeIDType ComTradeID;
    ///trade ID
    TThostFtdcTradeIDType TradeID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///hedge flag
    TThostFtdcHedgeFlagType    HedgeFlag;
    ///direction
    TThostFtdcDirectionType      Direction;
    ///total amount
    TThostFtdcVolumeType  TotalAmt;
    ///margin
    TThostFtdcMoneyType   Margin;
    ///excahnge margin
    TThostFtdcMoneyType   ExchMargin;
    ///margin rate by money
    TThostFtdcRatioType     MarginRateByMoney;
    ///margin rate by volume
    TThostFtdcRatioType     MarginRateByVolume;
    ///combination instrument ID
    TThostFtdcInstrumentIDType CombInstrumentID;
};
```

5.2.34. RspParkedOrderInsert

CTP server use this callback function to notify the client application about the sucess of “ReqParkedOrderInsert”.

definition:

```
void OnRspParkedOrderInsert(CThostFtdcParkedOrderField *pParkedOrder,
                    CThostFtdcRspInfoField *pRspInfo,
                    int nRequestID, bool bIsLast);
```

parameters:

pParkedOrder : *Pointer of the structure for the response of ReqParkedOrderInsert. The following is definition of the structure,*

```
struct CThostFtdcParkedOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///order reference
    TThostFtdcOrderRefType      OrderRef;
    ///user id
    TThostFtdcUserIDType  UserID;
    ///order price type
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///direction
    TThostFtdcDirectionType      Direction;
    ///combinationorder’s offset flag
    TThostFtdcCombOffsetFlagType   CombOffsetFlag;
    ///combination or hedge flag
    TThostFtdcCombHedgeFlagType   CombHedgeFlag;
    ///price
    TThostFtdcPriceType     LimitPrice;
    ///volume
    TThostFtdcVolumeType  VolumeTotalOriginal;
    /// Valid date   TThostFtdcTimeConditionType      TimeCondition;
    ///GTD DATE
    TThostFtdcDateType      GTDDate;
    ///volume condition
    TThostFtdcVolumeConditionType  VolumeCondition;
    ///min volume
    TThostFtdcVolumeType  MinVolume;
    ///trigger condition
```

```
        TThostFtdcContingentConditionType    ContingentCondition;
        ///stop price
        TThostFtdcPriceType     StopPrice;
        ///force close reason
        TThostFtdcForceCloseReasonType       ForceCloseReason;
        /// auto suspend flag
        TThostFtdcBoolType      IsAutoSuspend;
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
        ///user force close flag
        TThostFtdcBoolType      UserForceClose;
        ///exchange ID
        TThostFtdcExchangeIDType  ExchangeID;
        ///parked order system ID
        TThostFtdcParkedOrderIDType     ParkedOrderID;
        ///user type
        TThostFtdcUserTypeType      UserType;
        ///parked order status
        TThostFtdcParkedOrderStatusType       Status;
    };
```

5.2.35. RspParkedOrderAction

CTP server uses this callback function to notify the client application the success of "ReqParkedOrderAction".

definition:

```
void OnRspParkedOrderAction(
                CThostFtdcParkedOrderActionField *pParkedOrderAction,
                CThostFtdcRspInfoField *pRspInfo,
                int nRequestID, bool bIsLast);
```

parameters:

pParkedOrderAction : *Pointer of the structure for the response of ReqParkedOrderAction. The following is definition of the structure,*

```
    struct CThostFtdcParkedOrderActionField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType     InvestorID;
        ///order action reference
        TThostFtdcOrderActionRefType     OrderActionRef;
        ///order reference
        TThostFtdcOrderRefType       OrderRef;
        ///request ID
        TThostFtdcRequestIDType     RequestID;
        ///front ID
        TThostFtdcFrontIDType FrontID;
        ///session ID
        TThostFtdcSessionIDType     SessionID;
        ///exchange ID
        TThostFtdcExchangeIDType  ExchangeID;
        ///order system ID
        TThostFtdcOrderSysIDType   OrderSysID;
        ///action flag
        TThostFtdcActionFlagType    ActionFlag;
        ///price
        TThostFtdcPriceType     LimitPrice;
        ///volume change
        TThostFtdcVolumeType   VolumeChange;
        ///user id
        TThostFtdcUserIDType  UserID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///parked order action ID
        TThostFtdcParkedOrderActionIDType   ParkedOrderActionID;
        ///user type
        TThostFtdcUserTypeType       UserType;
        ///parked order action status
        TThostFtdcParkedOrderStatusType       Status;
    };
```

5.2.36. RspRemoveParkedOrder

CTP server use this callback function to notify the client application whether the success of “ReqRemoveParkedOrder”.

definition:

```
void OnRspRemoveParkedOrder(
                CThostFtdcRemoveParkedOrderField *pRemoveParkedOrder,
```

*CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast);*

parameters：

pRemoveParkedOrder： *Pointer of the structure for the response of ReqRemoveParkedOrder. The following is definition of the structure,*

```
struct CThostFtdcRemoveParkedOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///parked order system ID
    TThostFtdcParkedOrderIDType    ParkedOrderID;
};
```

5.2.37. RspRemoveParkedOrderAction

CTP server use this callback function to notify the client application about the success of "ReqRemoveParkedOrderAction".

definition：

```
void OnRspRemoveParkedOrderAction(
            CThostFtdcRemoveParkedOrderActionField *pRemoveParkedOrderAction,
            CThostFtdcRspInfoField *pRspInfo,
            int nRequestID, bool bIsLast);
```

parameters：

pRemoveParkedOrderAction： *Pointer of the structure for the response of ReqRemoveParkedOrderAction. The following is definition of the structure,*

```
struct CThostFtdcRemoveParkedOrderActionField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///parked order action trade ID
    TThostFtdcParkedOrderActionIDType  ParkedOrderActionID;
};
```

5.3. CThostFtdcTraderApi

CThostFtdcTraderApi interface’s functions incluede order insertion, order action, order and trade query, and other information query such as client information, investor account, and investor position, instrument information, instrument status, exchange publication, etc..

5.3.1. CreateFtdcTraderApi

The CTP client application uses this function to create a CThostFtdcTradeApi instance. Please note that do not use “new” to create any instance. .

definition:

*static CThostFtdcTradeApi *CreateFtdcTradeApi(const char *pszFlowPath = "");*

parameters:

pszFlowPath: *Pointer of a constant string, point to one special file directory which used to store notified information sent from CTP server, if not specified, the cuurent file directory is the default one.*

return value:

A pointer of an instance of CThostFtdcTradeAp.

5.3.2. Release

The CTP client application uses this function to delete a CThostFtdcTradeApi instance, but please do not use “delete”to delete any instance.

definition：

void Release()；

5.3.3. Init

The CTP client application uses this function to create the connection with CTP server, after this user can login in.

definition：

void Init()；

5.3.4. Join

The CTP client application uses this function to waiting the close of a CThostFtdcTradeApi instance.

definition：

void Join()；

5.3.5. GetTradingDay

The CTP client application uses this function to get the current traing?(what is this?) day, the return value will be valid only when the connection between client and CTP server is created successfully.

definition：

*const char *GetTradingDay()；*

return value：

a pointer of a constant string identifies the current trading date.

5.3.6. RegisterSpi

The CTP client application uses this function to register an instance inherited from the CThostFtdcTraderSpi interface.

definition:

*void RegisterSpi(CThostFtdcTraderSpi *pSpi) ;*

parameters:

pSpi: the pointer of the CThostFtdcTraderSpi instance.

5.3.7. RegisterFront

The CTP client application uses this function to register the front address of the CTP server, the function could be invocated more than one times to register more front addresses, and the API would selected one until the connection is created successfully.

definition:

void RegisterFront(char *pszFrontAddress);

parameters:

pszFrontAddress: *Pointer of the structure for the front address of the CTP server. The address format just like : "protocol://ipaddress:port", for example, "tcp://127.0.0.1:17001", "tcp" means the communication protocol, "127.0.0.1" identifies the front address."17001" identifies the server port.*

5.3.8. SubscribePrivateTopic

The CTP client application uses this function to subscribe the private topic from CTP server. The function must be called before the invocation of "init" function; otherwise the client application wouldn't receive its private stream.

definition：

void SubscribePrivateTopic(TE_RESUME_TYPE nResumeType);

parameters：

nResumeType： *the re-transmit mode of the private stream.*

TERT_RESTART: re-transmit from the begin of the current trading day.

TERT_RESUME: resume transmitting from the last received data.

TERT_QUICK: transmitting the new private stream data from the login time.

5.3.9. SubscribePublicTopic

The CTP client application uses this function to subscribe the public topic from CTP server. The function must be called before the invocation of “init” function; otherwise the client application wouldn’t receive its public stream.

definition：

void SubscribePublicTopic(TE_RESUME_TYPE nResumeType);

parameters：

nResumeType： *the re-transmit mode of the public stream.*

TERT_RESTART: re-transmit from the begin of the current trading day.

TERT_RESUME: resume transmitting from the last received data.

TERT_QUICK: transmitting the new public stream data from the login time.

5.3.10. ReqUserLogin

The CTP client application uses this function to send the login in request to the CTP server.

definition：

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

parameters：

pReqUserLoginField：*The pointer of the structure for user's login request. The following is definition of the structure,*

```
struct CThostFtdcReqUserLoginField
{
    ///trading day
    TThostFtdcDateType      TradingDay;
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///user id
    TThostFtdcUserIDType  UserID;
    ///password
    TThostFtdcPasswordType      Password;
    ///user product information
    TThostFtdcProductInfoType   UserProductInfo;
    ///interface product information
    TThostFtdcProductInfoType   InterfaceProductInfo;
    ///protocol information
    TThostFtdcProtocolInfoType  ProtocolInfo;
};
```

return value：

0，success.

-1，net connection failure.

-2，over the max quantity of unhandled requests.

-3，over the max requests per second.

5.3.11. ReqUserLogout

The CTP client application uses this function to send the login out request to the CTP server.

definition：

```
int ReqUserLogout(
                    CThostFtdcUserLogoutField *pUserLogout,
                    int nRequestID)；
```

parameters：

pReqUserLogout：*Pointer of the structure for user's logout request. The following is definition of the structure,*

```
struct CThostFtdcUserLogoutField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///user id
    TThostFtdcUserIDType  UserID;
};
```

return value：

0，success.

-1，net connection failure.

-2，over the max quantity of unhandled requests.

-3，over the max requests per second.

5.3.12. ReqUserPasswordUpdate

The CTP client application uses this function to send the user password update request to the CTP server.

definition：

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

parameters：

pUserPasswordUpdate：*Pointer of the structure for user password updation request. The following is definition of the structure,*

```
struct CThostFtdcUserPasswordUpdateField
```

```
{
      ///broker id
      TThostFtdcBrokerIDType      BrokerID;
      ///user id
      TThostFtdcUserIDType  UserID;
      ///old password
      TThostFtdcPasswordType      OldPassword;
      ///new password
      TThostFtdcPasswordType      NewPassword;
};
```

5.3.13. ReqTradingAccountPasswordUpdate

The CTP client application uses this function to send the account password update request to the CTP server.

definition:

```
int ReqTradingAccountPasswordUpdate(
                              CThostFtdcTradingAccountPasswordUpdateField
                              *pTradingAccountPasswordUpdate,
                              int nRequestID) ;
```

parameters:

pUserPasswordUpdate: *Pointer of the structure for account password updation request. The following is definition of the structure,*

```
struct CThostFtdcTradingAccountPasswordUpdateField
{
      ///broker id
      TThostFtdcBrokerIDType      BrokerID;
      ///account id
      TThostFtdcAccountIDType    AccountID;
      ///old password
      TThostFtdcPasswordType      OldPassword;
      ///new password
      TThostFtdcPasswordType      NewPassword;
};
```

5.3.14. ReqOrderInsert

The CTP client application uses this function to send the order insertion request to the CTP server.

definition:

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

parameters:

pInputOrder: *Pointer of the structure for order insertion request. The following is definition of the structure,*

```
struct CThostFtdcInputOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///order reference
    TThostFtdcOrderRefType      OrderRef;
    ///user id
    TThostFtdcUserIDType  UserID;
    ///order price type
    TThostFtdcOrderPriceTypeType    OrderPriceType;
    ///direction
    TThostFtdcDirectionType     Direction;
    ///combination order's offset flag
    TThostFtdcCombOffsetFlagType    CombOffsetFlag;
    ///combination or hedge flag
    TThostFtdcCombHedgeFlagType  CombHedgeFlag;
    ///price
    TThostFtdcPriceType     LimitPrice;
    ///volume
    TThostFtdcVolumeType  VolumeTotalOriginal;
    ///valid date
    TThostFtdcTimeConditionType     TimeCondition;
    ///GTD DATE
    TThostFtdcDateType      GTDDate;
```

```
        ///volume condition
        TThostFtdcVolumeConditionType   VolumeCondition;
        ///min volume
        TThostFtdcVolumeType  MinVolume;
        ///trigger condition
        TThostFtdcContingentConditionType    ContingentCondition;
        ///stop price
        TThostFtdcPriceType      StopPrice;
        ///force close reason
        TThostFtdcForceCloseReasonType       ForceCloseReason;
        /// auto suspend flag
        TThostFtdcBoolType       IsAutoSuspend;
        ///business unit
        TThostFtdcBusinessUnitType BusinessUnit;
        ///request ID
        TThostFtdcRequestIDType      RequestID;
    };
```

OrderRef： *order reference，which should increase monotonically. In the response of eachOnRspUserLogin, the client application could get the MaxOrderRef. Other worth mention, the CTP server compares the orderref as string, so staffing all placet of TThostFtdcOrderRefType is needed.*

5.3.15. ReqOrderAction

The CTP client application uses this function to send the order cancellation request to the CTP server.

definition：

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

parameters：

pOrderAction： *Pointer of the structure for order delettion request. The following is definition of the structure,*

```
    struct CThostFtdcOrderActionField
    {
        ///broker id
        TThostFtdcBrokerIDType       BrokerID;
```

```
///investor ID
TThostFtdcInvestorIDType     InvestorID;
///order action reference
TThostFtdcOrderActionRefType     OrderActionRef;
///order reference
TThostFtdcOrderRefType       OrderRef;
///request ID
TThostFtdcRequestIDType     RequestID;
///front ID
TThostFtdcFrontIDType FrontID;
///session ID
TThostFtdcSessionIDType      SessionID;
///exchange ID
TThostFtdcExchangeIDType  ExchangeID;
///order system ID
TThostFtdcOrderSysIDType   OrderSysID;
///action flag
TThostFtdcActionFlagType    ActionFlag;
///price
TThostFtdcPriceType     LimitPrice;
///volume change
TThostFtdcVolumeType   VolumeChange;
///action date
TThostFtdcDateType      ActionDate;
///action time
TThostFtdcTimeType      ActionTime;
///trader ID
TThostFtdcTraderIDType       TraderID;
///install ID
TThostFtdcInstallIDType       InstallID;
///order local ID
TThostFtdcOrderLocalIDType       OrderLocalID;
///action local ID
TThostFtdcOrderLocalIDType       ActionLocalID;
///participant ID
TThostFtdcParticipantIDType       ParticipantID;
///trading code
TThostFtdcClientIDTypeClientID;
///business unit
TThostFtdcBusinessUnitType BusinessUnit;
///order action status
TThostFtdcOrderActionStatusType OrderActionStatus;
///user id
TThostFtdcUserIDType  UserID;
```

```
        ///status message
        TThostFtdcErrorMsgType      StatusMsg;
    };
```

5.3.16. ReqQueryMaxOrderVolume

The CTP client application uses this function to send the request of query the max order volume to the CTP server.

definition:

```
        int ReqQueryMaxOrderVolume(
                CThostFtdcQueryMaxOrderVolumeField *pQueryMaxOrderVolume,
                int nRequestID);
```

parameters:

pQueryMaxOrderVolume: *Pointer of the structure for the request of query the max order volume. The following is definition of the structure,*

```
    struct CThostFtdcQueryMaxOrderVolumeField
    {
        ///broker id
        TThostFtdcBrokerIDType      BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType    InvestorID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///direction
        TThostFtdcDirectionType     Direction;
        ///offset flag
        TThostFtdcOffsetFlagType    OffsetFlag;
        ///hedge flag
        TThostFtdcHedgeFlagType     HedgeFlag;
        ///max volume
        TThostFtdcVolumeType   MaxVolume;
    };
```

5.3.17. ReqSettlementInfoConfirm

The CTP client application uses this function to confirm the settlement information fromthe CTP server.

definition：

```
 int ReqSettlementInfoConfirm(
            CThostFtdcSettlementInfoConfirmField *pSettlementInfoConfirm,
            int nRequestID)；
```

parameters：

pSettlementInfoConfirm：*Pointer of the structure for settlement information confirmation request. The following is definition of the structure,*

```
struct CThostFtdcSettlementInfoConfirmField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///confirm date
    TThostFtdcDateType      ConfirmDate;
    ///confirm time
    TThostFtdcTimeType      ConfirmTime;
};
```

5.3.18. ReqQryOrder

The CTP client application uses this function to send the order query request to the CTP server.

definition：

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

parameters：

pQryOrder：*Pointer of the structure for order query request. The following is definition of the structure,*

```
struct CThostFtdcQryOrderField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
```

```
        ///investor ID
        TThostFtdcInvestorIDType     InvestorID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///exchange ID
        TThostFtdcExchangeIDType   ExchangeID;
        ///order system ID
        TThostFtdcOrderSysIDType   OrderSysID;
};
```

5.3.19. ReqQryTrade

The CTP client application uses this function to send the trade query request to the CTP server.

definition:

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

parameters:

pQryTrade：*Pointer of the structure for trade query request. The following is definition of the structure,*

```
struct CThostFtdcQryTradeField
{
        ///broker id
        TThostFtdcBrokerIDType       BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType     InvestorID;
        ///instrument ID
        TThostFtdcInstrumentIDType InstrumentID;
        ///exchange ID
        TThostFtdcExchangeIDType   ExchangeID;
        ///trade ID
        TThostFtdcTradeIDType TradeID;
};
```

5.3.20. ReqQry Investor

The CTP client application uses this function to send the investor query request to the CTP server.

definition:

```
int ReqQry Investor (
                CThostFtdcQryInvestorField *pQryInvestor,
                int nRequestID);
```

parameters:

pQry Investor: *Pointer of the structure for investor query request. The following is definition of the structure,*

```
struct CThostFtdcQryInvestorField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
};
```

5.3.21. ReqQryInvestorPosition

The CTP client application uses this function to send the investor position query request to the CTP server.

definition:

```
int ReqQryInvestorPosition(
                CThostFtdcQryInvestorPositionField *pQryInvestorPosition,
                int nRequestID);
```

parameters:

pQryInvestorPosition: *Pointer of the structure for investor position query request. The following is definition of the structure,*

```
struct CThostFtdcQryInvestorPositionField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
};
```

5.3.22. ReqQryTradingAccount

The CTP client application uses this function to send the trading account query request to the CTP server.

definition：

```
    int ReqQryTradingAccount(
                CThostFtdcQryTradingAccountField *pQryTradingAccount,
                int nRequestID)；
```

parameters：

pQryTradingAccount：*Pointer of the structure for trading account query request. The following is definition of the structure,*

```
struct CThostFtdcQryTradingAccountField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
};
```

5.3.23. ReqQryTradingCode

The CTP client application uses this function to send the trading code query request to the CTP server.

definition：

```
  int ReqQryTradingCode(
                CThostFtdcQryTradingCodeField *pQryTradingCode,
                int nRequestID)；
```

parameters:

pQryTradingCode: *Pointer of the structure for trading code query request. The following is definition of the structure,*

```
struct CThostFtdcQryTradingCodeField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
    ///trading code
    TThostFtdcClientIDTypeClientID;
};
```

5.3.24. ReqQryExchange

The CTP client application uses this function to send the exchange query request to the CTP server.

definition:

```
int ReqQryExchange(
        CThostFtdcQryExchangeField *pQryExchange,
        int nRequestID);
```

parameters:

pQryExchange: *Pointer of the structure for exchange query request. The following is definition of the structure,*

```
struct CThostFtdcQryExchangeField
{
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
};
```

5.3.25. ReqQryInstrument

The CTP client application uses this function to send the instrument query request to the CTP server.

definition:

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

parameters:

pQryInstrument：*Pointer of the structure for instrument query request. The following is definition of the structure,*

```
struct CThostFtdcQryInstrumentField
{
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
    ///exchange ID
    TThostFtdcExchangeIDType  ExchangeID;
    ///exchange instrument ID
    TThostFtdcExchangeInstIDType    ExchangeInstID;
    ///product ID
    TThostFtdcInstrumentIDType ProductID;
};
```

5.3.26. ReqQryDepthMarketData

The CTP client application uses this function to send the market quotation query request to the CTP server.

definition:

```
int ReqQryDepthMarketData(
                CThostFtdcQryDepthMarketDataField *pQryDepthMarketData,
                int nRequestID)；
```

parameters:

pQryDepthMarketData：*Pointer of the structure for market quotation query* *request. The following is definition of the structure,*

```
struct CThostFtdcQryDepthMarketDataField
{
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
};
```

5.3.27. ReqQrySettlementInfo

The CTP client application uses this function to send the settlement information query request to the CTP server.

definition：

```
        int ReqQrySettlementInfo(
                    CThostFtdcQrySettlementInfoField *pQrySettlementInfo,
                    int nRequestID)；
```

parameters：

pQrySettlementInfo：*Pointer of the structure for settlement information query request. The following is definition of the structure,*

```
struct CThostFtdcQrySettlementInfoField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///trading day
    TThostFtdcDateType      TradingDay;
};
```

5.3.28. ReqQryInvestorPositionDetail

The CTP client application uses this function to send the investor position detail query request to the CTP server.

definition:

```
int ReqQryInvestorPositionDetail(
            CThostFtdcQryInvestorPositionDetailField *pQryInvestorPositionDetail,
            int nRequestID);
```

parameters:

pQryInvestorPositionDetail: *Pointer of the structure for investor position detail query request. The following is definition of the structure,*

```
struct CThostFtdcQryInvestorPositionDetailField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType InstrumentID;
};
```

5.3.29. ReqQryNotice

The CTP client application uses this function to send the notice query request to the CTP server.

definition:

```
int ReqQryNotice(
            CThostFtdcQryNoticeField *pQryNotice,
            int nRequestID);
```

parameters:

pQryNotice: *Pointer of the structure fornotice query request. The following is definition of the structure,*

```
struct CThostFtdcQryNoticeField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
};
```

5.3.30. ReqQrySettlementInfoConfirm

The CTP client application uses this function to send the settlement information confirmation query request to the CTP server.

definition:

```
int ReqQrySettlementInfoConfirm(
                    CThostFtdcQrySettlementInfoConfirmField
                    *pQrySettlementInfoConfirm,
                    int nRequestID);
```

parameters:

pQrySettlementInfoConfirm: *Pointer of the structure for settlement information confirmation query request. The following is definition of the structure,*

```
struct CThostFtdcQrySettlementInfoConfirmField
{
    ///broker id
    TThostFtdcBrokerIDType      BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
};
```

5.3.31. ReqQryParkedOrder

The CTP client application uses this function to send the parked order query request to the CTP server.

definition:

```
int ReqQryParkedOrder(CThostFtdcQryParkedOrderField *pQryParkedOrder,
                      int nRequestID);
```

parameters:

pQryParkedOrder: *Pointer of the structure for parked order query request. The following is definition of the structure,*

```
struct CThostFtdcQryParkedOrderField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType     InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType  InstrumentID;
    ///exchange ID
    TThostFtdcExchangeIDType   ExchangeID;
};
```

5.3.32. ReqQryParkedOrderAction

The CTP client application uses this function to send the parked order action query request to the CTP server.

definition:

```
int ReqQryParkedOrderAction(
            CThostFtdcQryParkedOrderActionField *pQryParkedOrderAction,
```

int nRequestID)；

parameters：

pQryParkedOrderAction：*Pointer of the structure for parked order action query request. The following is definition of the structure,*

```
struct CThostFtdcQryParkedOrderActionField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType     InvestorID;
    ///exchange ID
    TThostFtdcExchangeIDType   ExchangeID;
};
```

5.3.33. ReqQryInvestorPositionCombineDetail

The CTP client application uses this function to send the investor combination position detail query request to the CTP server.

definition：

```
int ReqQryInvestorPositionCombineDetail(
        CThostFtdcQryInvestorPositionCombineDetailField *pQryInvestorPositionCombineDetail,
          int nRequestID);；
```

parameters：

pQryInvestorPositionCombineDetail：*Pointer of the structure for investor combination position detail query request. The following is definition of the structure,*

```
struct CThostFtdcQryInvestorPositionCombineDetailField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType      InvestorID;
    ///combination instrument ID
    TThostFtdcInstrumentIDType  CombInstrumentID;
};
```

5.3.34. ReqParkedOrderInsert

The CTP client application uses this function to send the parked order insertion request to the CTP server.

definition：

```
int ReqParkedOrderInsert (CThostFtdcParkedOrderField *pParkedOrder,
                          int nRequestID)；
```

parameters：

pParkedOrder：*Pointer of the structure for parked order insertion request. The following is definition of the structure,*

```
struct CThostFtdcParkedOrderField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType      InvestorID;
    ///instrument ID
    TThostFtdcInstrumentIDType  InstrumentID;
    ///order reference
    TThostFtdcOrderRefType OrderRef;
    ///user id
    TThostFtdcUserIDType    UserID;
    ///order price type
    TThostFtdcOrderPriceTypeType    OrderPriceType;
```

```
///direction
TThostFtdcDirectionType Direction;
///combination offset flag
TThostFtdcCombOffsetFlagType    CombOffsetFlag;
///combination hedge flag
TThostFtdcCombHedgeFlagType   CombHedgeFlag;
///price
TThostFtdcPriceType LimitPrice;
///volume
TThostFtdcVolumeType    VolumeTotalOriginal;
///valid date
TThostFtdcTimeConditionType     TimeCondition;
///GTD DATE
TThostFtdcDateType  GTDDate;
///volume condition
TThostFtdcVolumeConditionType   VolumeCondition;
///min volume
TThostFtdcVolumeType    MinVolume;
///trigger condition
TThostFtdcContingentConditionType   ContingentCondition;
///stop price
TThostFtdcPriceType StopPrice;
///force close reason
TThostFtdcForceCloseReasonType      ForceCloseReason;
///is auto suspend
TThostFtdcBoolType  IsAutoSuspend;
///business unit
TThostFtdcBusinessUnitType  BusinessUnit;
///request ID
TThostFtdcRequestIDType     RequestID;
```

```
        ///user force close flag
        TThostFtdcBoolType  UserForceClose;
        ///exchange ID
        TThostFtdcExchangeIDType   ExchangeID;
        ///parked order system ID
        TThostFtdcParkedOrderIDType     ParkedOrderID;
        ///user type
        TThostFtdcUserTypeType UserType;
        ///parked order status
        TThostFtdcParkedOrderStatusType      Status;
    };
```

5.3.35. ReqParkedOrderAction

The CTP client application uses this function to send the parked order action request to the CTP server.

definition:

```
int ReqParkedOrderAction(CThostFtdcParkedOrderActionField *pParkedOrderAction,
                                int nRequestID);
```

parameters:

pParkedOrderAction: *Pointer of the structure for parked order action request. The following is definition of the structure,*

```
    struct CThostFtdcParkedOrderActionField
    {
        ///broker id
        TThostFtdcBrokerIDType BrokerID;
        ///investor ID
        TThostFtdcInvestorIDType     InvestorID;
        ///order action reference
    TThostFtdcOrderActionRefType    OrderActionRef;
    ///order reference
    TThostFtdcOrderRefType OrderRef;
    ///request ID
    TThostFtdcRequestIDType     RequestID;
    ///front ID
    TThostFtdcFrontIDType   FrontID;
    ///session ID
    TThostFtdcSessionIDType     SessionID;
    ///exchange ID
    TThostFtdcExchangeIDType   ExchangeID;
    ///order system ID
    TThostFtdcOrderSysIDType    OrderSysID;
    ///action flag
    TThostFtdcActionFlagType    ActionFlag;
    ///price
    TThostFtdcPriceType LimitPrice;
    ///volume change
    TThostFtdcVolumeType    VolumeChange;
    ///user id
    TThostFtdcUserIDType    UserID;
    ///instrument ID
    TThostFtdcInstrumentIDType  InstrumentID;
    ///parked order action ID
    TThostFtdcParkedOrderActionIDType ParkedOrderActionID;
    ///user type
    TThostFtdcUserTypeType UserType;
    ///parked order action status
    TThostFtdcParkedOrderStatusType     Status;
};
```

5.3.36. ReqRemoveParkedOrder

The CTP client application uses this function to send the parked ordercancel request to the CTP server.

definition:

```
int ReqRemoveParkedOrder(CThostFtdcRemoveParkedOrderField *pRemoveParkedOrder,
                                        int nRequestID)；
```

parameters:

pRemoveParkedOrder: *Pointer of the structure for parked order removing request. The following is definition of the structure,*

```
struct CThostFtdcRemoveParkedOrderField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType    InvestorID;
    ///parked order system ID
    TThostFtdcParkedOrderIDType    ParkedOrderID;
};
```

5.3.37. ReqRemoveParkedOrderAction

The CTP client application uses this function to send the parked order actioncancel request to the CTP server.

definition:

```
Int ReqRemoveParkedOrderAction(
                    CThostFtdcRemoveParkedOrderActionField *pRemoveParkedOrderAction,
                    int nRequestID);；
```

parameters:

pRemoveParkedOrderAction: *Pointer of the structure for parked order removing request. The following is definition of the structure,*

```
struct CThostFtdcRemoveParkedOrderActionField
{
    ///broker id
    TThostFtdcBrokerIDType BrokerID;
    ///investor ID
    TThostFtdcInvestorIDType     InvestorID;
    ///parked order action trade ID
    TThostFtdcParkedOrderActionIDType ParkedOrderActionID;
};
```

Chapter 6. example

7.1 trade API example

```
    // tradeapitest.cpp :
    // A simple example demonstrate how to use CThostFtdcTraderApi
and CThostFtdcTraderSpi interface application.
    // This example will demonstrates the procedure of an order
insertion.

    #include <stdio.h>
    #include <windows.h>
    #include "FtdcTraderApi.h"

    // Flag of the order insertion finished or not.
    // Create a manual reset event with no signal
    HANDLE g_hEvent = CreateEvent(NULL, true, false, NULL);

    // participant ID
    TThostFtdcBrokerIDType g_chBrokerID;
    // user id
    TThostFtdcUserIDType g_chUserID;

    class CSimpleHandler : public CThostFtdcTraderSpi
    {
    public:
        // constructor, which need a valid pointer to a
CThostFtdcMduserApi instance
        CSimpleHandler(CThostFtdcTraderApi *pUserApi) :
m_pUserApi(pUserApi) {}

        ~CSimpleHandler() {}

        // After making a succeed connection with the CTP server, the
 client should send the login request to the CTP server.
        virtual void OnFrontConnected()
        {
            CThostFtdcReqUserLoginField reqUserLogin;
            // get BrokerID
            printf("BrokerID:");
            scanf("%s", &g_chBrokerID);
            strcpy(reqUserLogin. BrokerID, g_chBrokerID);
            // get user id
            printf("userid:");
            scanf("%s", &g_chUserID);
            strcpy(reqUserLogin.UserID, g_chUserID);
            // get password
            printf("password:");
            scanf("%s", &reqUserLogin.Password);
            // send the login request
            m_pUserApi->ReqUserLogin(&reqUserLogin, 0);
        }
	//When the connection between client and the CTP server disconnected,the follwing function will be called.
	virtual void OnFrontDisconnected(int nReason)
	{
		// Inthis case, API willreconnect，the client application can ignore this.
		printf("OnFrontDisconnected.\n");
	}

	// After receiving the login request from the client，the CTP server will send the following response to notify the client whether the login success or not.
	virtual void OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin, CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
	{
		printf("OnRspUserLogin:\n");
		printf("ErrorCode=[%d], ErrorMsg=[%s]\n", pRspInfo->ErrorID, pRspInfo->ErrorMsg);
		printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);

		if (pRspInfo->ErrorID != 0) {
			// in case any login failure, the client should handle this error.
			printf("Failed to login, errorcode=%d errormsg=%s requestid=%d chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
			exit(-1);
		}

		// login success, then send order insertion request.
    CThostFtdcInputOrderField ord;
		memset(&ord, 0, sizeof(ord));
 //broker id
 strcpy(ord.BrokerID, g_chBrokerID);
 //investor ID
 strcpy(ord.InvestorID, "12345");
 // instrument ID
 strcpy(ord.InstrumentID, "cn0601");
///order reference
strcpy(ord.OrderRef, "000000000001");
 // user id
 strcpy(ord.UserID, g_chUserID);
 // order price type
 ord.OrderPriceType = THOST_FTDC_OPT_LimitPrice;
 // direction
 ord.Direction = THOST_FTDC_D_Buy;
 // combination order’ s offset flag
 strcpy(ord.CombOffsetFlag, "0");
 // combination or hedge flag
 strcpy(ord.CombHedgeFlag, "1");
 // price
 ord.LimitPrice = 50000;
 // volume
 ord.VolumeTotalOriginal = 10;
 // valid date
 ord.TimeCondition = THOST_FTDC_TC_GFD;
 // GTD DATE
 strcpy(ord.GTDDate, "");
 // volume condition
 ord.VolumeCondition = THOST_FTDC_VC_AV;
 // min volume
 ord.MinVolume = 0;
 // trigger condition
```

```
            ord.ContingentCondition = THOST_FTDC_CC_Immediately;
            // stop price
            ord.StopPrice = 0;
            // force close reason
            ord.ForceCloseReason = THOST_FTDC_FCC_NotForceClose;
            // auto suspend flag
            ord.IsAutoSuspend = 0;

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

        // order insertion response
        virtual void OnRspOrderInsert(CThostFtdcInputOrderField
*pInputOrder, CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool
bIsLast)
        {
            // output the order insertion result
            printf("ErrorCode=[%d], ErrorMsg=[%s]\n",
pRspInfo->ErrorID, pRspInfo->ErrorMsg);

            // inform the main thread order insertion is over
            SetEvent(g_hEvent);
        };

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

        // the error notification caused by client request
        virtual void OnRspError(CThostFtdcRspInfoField *pRspInfo,
int nRequestID, bool bIsLast) {
          printf("OnRspError:\n");
          printf("ErrorCode=[%d], ErrorMsg=[%s]\n",
pRspInfo->ErrorID, pRspInfo->ErrorMsg);
          printf("RequestID=[%d], Chain=[%d]\n", nRequestID,
bIsLast);
          // the client should handle the error
          {error handle code}
       }

    private:
       // a pointer of CThostFtdcMduserApi instance
       CThostFtdcTraderApi *m_pUserApi;
    };



    int main()
    {
       // create a CThostFtdcTraderApi instance
       CThostFtdcTraderApi *pUserApi =
CThostFtdcTraderApi::CreateFtdcTraderApi();
       // create an event handler instance
       CSimpleHandler sh(pUserApi);
       // register an event handler instance
       pUserApi->RegisterSpi(&sh);

       // subscribe private topic
       pUserApi->SubscribePrivateTopic(TERT_RESUME);

       // subscribe public topic
       pUserApi->SubscribePublicTopic(TERT_RESUME);

       // register the CTP front address and port
    pUserApi->RegisterFront("tcp://172.16.0.31:26205");
    // make the connection between client and CTP server
    pUserApi->Init();

    // waiting for the order insertion.
    WaitForSingleObject(g_hEvent, INFINITE);

    // release the API instance
    pUserApi->Release();

    return 0;
}
```

7.2 quotation API example

```
    // tradeapitest.cpp :
    #include <stdio.h>
    #include <windows.h>
    #include "ThostFtdcMdApi.h"
    // the flag whether the quotation data received or not.
    // Create a manual reset event with no signal
    HANDLE g_hEvent = CreateEvent(NULL, true, false, NULL);
    // participant ID
    TThostFtdcBrokerIDType g_chBrokerID;
    // user id
    TThostFtdcUserIDType g_chUserID;
    class CSimpleHandler : public CThostFtdcMdSpi
    {
    public:
        // constructor,which need a valid pointer of a CThostFtdcMdApi instance
    CSimpleHandler(CThostFtdcMdApi *pUserApi) : m_pUserApi(pUserApi) {}
        ~CSimpleHandler() {}

        //  when  the  connection  between  client  and  CTP  server  is  created
successfully, the client would send the login request to the CTP server.
        virtual void OnFrontConnected()
        {
            CThostFtdcReqUserLoginField reqUserLogin;
            // get BrokerID
            printf("BrokerID:");
            scanf("%s", &g_chBrokerID);
            strcpy(reqUserLogin. BrokerID, g_chBrokerID);
            // get userid
            printf("userid:");
            scanf("%s", &g_chUserID);
            strcpy(reqUserLogin.UserID, g_chUserID);
            // get password
            printf("password:");
            scanf("%s", &reqUserLogin.Password);
            // send the login request
            m_pUserApi->ReqUserLogin(&reqUserLogin, 0);
        }

        // when client and CTP server disconnected,the follwing function will be
called.
        virtual void OnFrontDisconnected(int nReason)
        {
            // inhtis case, API will reconnect, the client application canignore
this.
```

```
            printf("OnFrontDisconnected.\n");
        }
        // after receiving the login request from the client, the CTP server will
send the following response to notify the client whether the login success or not.
        virtual void OnRspUserLogin(CThostFtdcRspUserLoginField *pRspUserLogin,
CThostFtdcRspInfoField *pRspInfo, int nRequestID, bool bIsLast)
        {
            printf("OnRspUserLogin:\n");
            printf("ErrorCode=[%d],    ErrorMsg=[%s]\n",    pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
            printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
            if (pRspInfo->ErrorID != 0) {
                // login failure, the client should handle this error.
                printf("Failed to login, errorcode=%d errormsg=%s requestid=%d
chain=%d", pRspInfo->ErrorID, pRspInfo->ErrorMsg, nRequestID, bIsLast);
                exit(-1);
            }
            // login success, then subscribe the quotation information
           char  * Instrumnet[]={“IF0809”,”IF0812”};
            pUserApi->SubscribeMarketData (Instrumnet,2);
           //or unsubscribe the quotation
           pUserApi->UnSubscribeMarketData (Instrumnet,2);
        }
        // quotation return
        virtual    void    OnRtnDepthMarketData(CThostFtdcDepthMarketDataField
*pDepthMarketData)
        {
         //output the order insert result
            printf("ErrorCode=[%d],    ErrorMsg=[%s]\n",    pRspInfo->ErrorID,
pRspInfo->ErrorMsg);

            // set the flag when the quotation data received.
            SetEvent(g_hEvent);
        };



        // the error notification caused by client request
        virtual void OnRspError(CThostFtdcRspInfoField *pRspInfo, int nRequestID,
bool bIsLast) {
            printf("OnRspError:\n");
            printf("ErrorCode=[%d],    ErrorMsg=[%s]\n",    pRspInfo->ErrorID,
pRspInfo->ErrorMsg);
            printf("RequestID=[%d], Chain=[%d]\n", nRequestID, bIsLast);
            // the client should handle the error
            {error handle code}
        }
    private:
        // a pointer to CThostFtdcMdApi instance
        CThostFtdcMdApi *m_pUserApi;
    };
    int main()
    {
        // create a CThostFtdcMdApi instance
        CThostFtdcMdApi *pUserApi = CThostFtdcMdApi::CreateFtdcMdApi();
        // create an event handler instance
        CSimpleHandler sh(pUserApi);
        // register an event handler instance
        pUserApi->RegisterSpi(&sh);




        // register the CTP front address and port
        pUserApi->RegisterFront("tcp://172.16.0.31:26213");
        // start the connection between client and CTP server
        pUserApi->Init();

        // waiting for the quotation data
        WaitForSingleObject(g_hEvent, INFINITE);
        // release API instance
        pUserApi->Release();
        return 0;
}
```