Zhengzhou Commodity Exchange

ZCEAPI Handbook

Version 2.1

This document is applicable to ZCEAPI V2.1.

Zhengzhou Commodity Exchange
August 2021

Contents

Revision History

API Version	Date	Change	Notes
2.0.0	December 2019	Created instructions	Revise instructions for the new version of the handbook on the basis of the V1.1.3.7
2.0.1	January 20, 2021	Added instructions on the use of emergency addresses for three functions as follows: API_Init; API_Stop; API_ConnectEx	
2.1.0	August 8, 2021	1. Deleted the API_ConnectEx and added the API_Connect; 2. Added parameters on whether to automatically initiate the business-driven thread for the function API_CreateExchgConnection, and in the meantime, added the API_RunEvent; and 3. Deleted descriptions for the query stream and emergency addresses	

1. Overview

1.1 Purpose and Scope

This document is all about the instructions on how to use the remote trading API ("ZCEAPI") issued by Zhengzhou Commodity Exchange (hereinafter referred to as "Exchange"). Its intended audience are software developers who use the ZCEAPI to develop trading or quotation software that communicates with the trading system of this Exchange. As this document can provide guidance for developers who use the ZCEAPI to develop software mentioned above, all users of the ZCEAPI are therefore expected to read this document carefully.

If there is any discrepancy between what is described in this document and the reality, please contact the technical support of this Exchange in a timely manner to confirm it.

Developers shall make reasonable use of various development interfaces and shall not maliciously affect or compromise any systems of this Exchange. Moreover, they shall fully test the systems developed by them to ensure that their systems have the right functions. Furthermore, developers shall not develop functions that access the systems of this Exchange in violation of rules or regulations, including by cracking APIs or protocols or by any other similar means.

The right to interpret this document rests on Zhengzhou Commodity Exchange.

1.2 Related Documents

Released together with this document is the *ZCEAPI Reference Manual of Zhengzhou Commodity Exchange* (i.e., the *Zhengzhou Commodity Exchange Futures Trading Data Exchange Interface Specifications*, hereinafter referred to as *ZCEAPI Reference Manual*).

API Demos are released in conjunction with this document, for they demonstrate the basic method to use the ZCEAPI. Therefore, developers are expected to use the ZCEAPI with reference to the descriptions in the Demos.

1.3 Introduction

The ZCEAPI is an API used by Zhengzhou Commodity Exchange for remote trading purposes. It is used by a client's applications to access and communicate with the trading system of this Exchange. The ZCEAPI provides programmers with a set of APIs and a simple protocol for conveying messages on trading data and quotations.

The ZCEAPI encapsulates the underlying connection and communication protocol with which to communicate with the trading system of this Exchange. In using the ZCEAPI to communicate with this Exchange, a user needs only to assign the ZCEAPI data packets appropriate values and then call the right functions to send the packages with reference to the *ZCEAPI Reference Manual*.

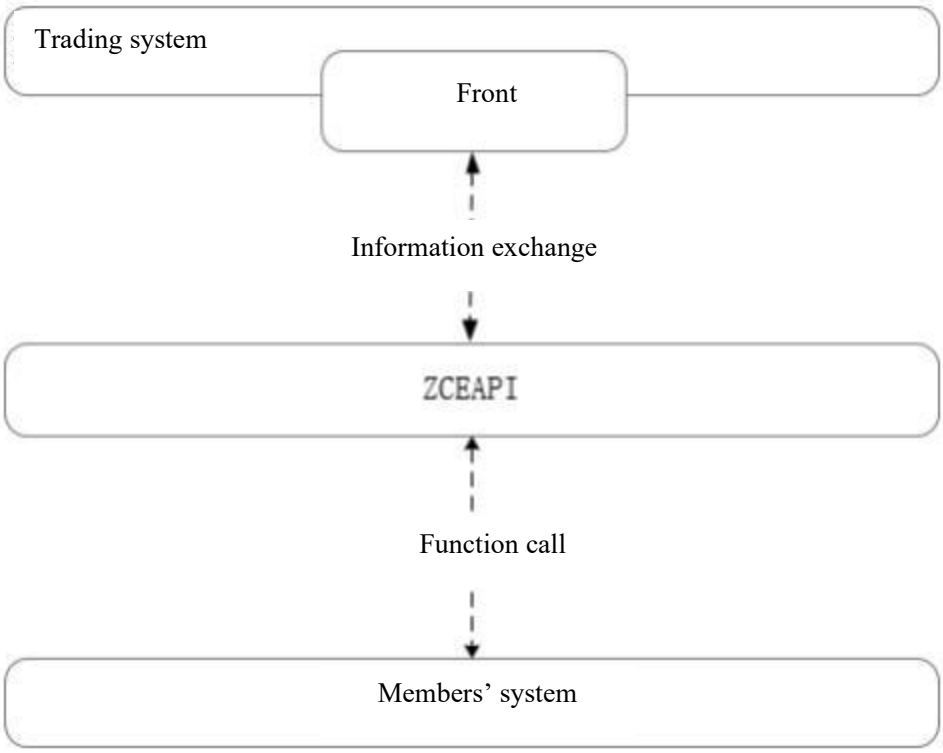
The heartbeat mechanism will be maintained by the ZCEAPI itself. The encryption, decryption, compression, and decompression of data, among other things, will be done all by the ZCEAPI. All a user has to do is to select appropriate parameters for the API.

The ZCEAPI uses callback mechanisms to notify users of link events and data receipt. In the case of a link event, such as link error, disconnection, and receipt of data packets, the ZCEAPI will automatically call back the function set by users to hand the event.

In short, with the help of the ZCEAPI, a user can conveniently communicate with, and finish transaction processing on, the trading system of this Exchange.

1.4 Location of the ZCEAPI

The ZCEAPI is used by trading software developers on the application end. As part of the client's application, it operates on the client side of the trading system (as shown in the following chart).



1.5 Communication Method of the ZCEAPI

In business terms, the communication between the ZCEAPI and this Exchange's backend system falls into three modes, namely, conservational communication, private communication, and broadcast communication (additional modes may be added in the future). In whichever mode, a user can always connect his applications to this Exchange's trading system through a reliable TCP link created by the ZCEAPI so as to perform his trading business or obtain market data.

The conversational mode (the resulting data stream is thus conversational stream) enables all transaction processing and part of the query operations and supports all the functions of data exchange between the trading system of this Exchange and the systems of its members.

The private mode (the resulting data stream is private stream) is a reliable data link that serves as a supplement to the conversational mode. Its primary functions include:

1) when an entry in the order book undergoes a change in status or quantity, the trading system will send order status confirmation, SPD arbitrage confirmation, hedging confirmation, option exercise confirmation, or quote response confirmation messages through the private stream.

2) when a record of market match undergoes a change, the trading system will send the TradeInsertSingle (PID=0x11001) message through the private stream.

The broadcast mode (the resulting data stream is broadcast stream) is a supplement to the conversational mode, too. When public information, e.g., system status, contract information, exchange bulletins, and market developments, undergoes any change, the trading system will circulate such change to the systems of members through the broadcast mode to reduce the number of queries made by members and accelerate the feedback of information to them.

On the underlying level, the ZCEAPI offers optional links, i.e., TCP and UDP, to communicate with this Exchange's trading system. It offers a TCP link for the data stream, including conversational, private, and broadcast streams, of each communication mode. For the broadcast stream, the underlying communication may be added an extra UDP link. The manner (unicast or multicast) of the UDP communication is contingent on the protocol adopted by a user to log in to the broadcast stream (the phase 5 trading system supports protocol 2 and 11). If a user logs in through the protocol 11 or a higher version of such protocol, the trading system will broadcast quotation changes, among other things, through the UDP link. For details, please refer to the *ZCEAPI Reference Manual* of this Exchange.

1.6 Operating Environment for the ZCEAPI

The ZCEAPI can run on both the Windows and Linux operating systems. It offers 64-bit versions for both the platforms. However, the windows system must be of the version NT10.0 or higher (Note: the latest version requires the test of compatibility); the Linux system must be of the kernel version 3.10 or higher and the gcc version 4.8.5 or higher (support the standard C++11 or later).

On the Windows platform, the ZCEAPI is to be called by an application in the form of dynamic link library (.dll). On the Linux/Unix platform, however, it is to be called in the form of shared objects (.so). The dll or so documents will be released together with this document.

Except in the case of the standard C++ runtime library, a user does not have to make special configuration for his operating system to use the ZCEAPI. All he has to do is to obtain the dll or so documents, as well as corresponding auxiliary files, from the platform of this Exchange.

1.7 Notes for the ZCEAPI-related Documents

The ZCEAPI-related documents released include the following:

For the Windows platform:

ZCEFTDAPI.dll (a dynamic link library file for the ZCEAPI)

ZCEFTDAPI.lib (a guiding library file for the ZCEFTDAPI.dll, used by C/C++ users)

FTDAPI.h (an interface definition file for the ZCEAPI. C/C++ users can directly use it, whereas the users of other languages can, according to the definition of c-style interface functions in the file, get the declarations and categories of the interface functions in their own languages.)

For the Linux platform:

libZCEFTDAPI.so (a shared library file for the ZCEAPI)

FTDAPI.h (an interface definition file for the ZCEAPI. C/C++ users can directly use it.)

2. Instructions for the ZCEAPI

2.1 Overview

The ZCEAPI appears in the system as a dynamic library. It provides standardized C-style interface functions, as well as data categories that comply with the definitions of the C language. For the definition of these functions and categories, please see notes in Part 4 of this document.

To use the ZCEAPI, a user must download the ZCEAPI release packages from the appropriate platform of this Exchange. The packages contain the documents necessary for a user to use the API.

In using a dynamic link library, a programmer normally must make a declaration on the interface provided by the library. When using the ZCEAPI, a programmer must also make a declaration on the definition of interface functions and of basic data. A C/C++ programmer can directly use the FTDAPI.h files released by a dynamic library, for the files provide all the declarations required of the ZCEAPI. On the other hand, a user of other software development languages must make declarations for the ZCEAPI interfaces in their own languages in accordance with the interface declarations in the FTDAPI.h file or the explanations in relevant parts of this document.

Worthy of attention is that the ZCEAPI provides its own format for data packets, because it is an interface used to exchange data. For different business, a user can complete or read different data fields. For all business types and data fields, the ZCEAPI has offered corresponding codes. For the detail of such codes, please refer to the *ZCEAPI Reference Manual* of this Exchange. Given the language used, a user can also define their own constants in their programs in accordance with the definitions given in the *ZCEAPI Reference Manual* of this Exchange.

Note: For the method to use the standard dynamic library for various platforms, please refer to the technical files for relevant platforms.

To use the ZCEAPI is to use the interface provided by the former. The use of the ZCEAPI must follow some steps in succession. Firstly, a user must call the function API_Init to initialize the API, and he must succeed in doing so in order to proceed to the next step. Secondly, he must create appropriate connections as per the data stream to be accessed (**according to the instructions in version 1.5, the trading system of this Exchange allows only that a connection access a data stream**). As the ZCEAPI processes events through callback mechanisms, the user must assign a callback event handler to each connection. That done, he shall create the link to this Exchange, log in to relevant data stream, and send or receive trading data. After all transactions are finished, he shall release the connection created, and call the function API_Stop to cease the API service.

The ZCEAPI interface is non-thread safe.

For detailed instructions for the ZCEAPI, please refer to section 2.3.

2.2 The Thread-based Architecture

The ZCEAPI offers two driver models, i.e., the self-driven mode and the user-driven mode. Specifically, a user can opt to let the API initiate a thread to drive a link event (mainly data receipt and basic parsing) at the time of creating a connection, or alternatively, initiate a thread on his own to call an API event-drive function to drive a link event.

As stated above, the ZCEAPI adopts callback functions to handle events, so when a user creates a connection, he must assign a callback function to the connection. In the self-driven mode, the API

will thus voluntarily initiate a thread to help drive the event of that connection. In a user-driven mode, however, a user must initiate a thread on his own to call an event-driven function and in turn to drive a link event.

When a user chooses to receive a broadcast stream through the UDP protocol (for which the user must log in to the broadcast stream using a protocol of the version 11 or higher), the ZCEAPI will separately initiate an additional thread to drive the receipt of the UDP data and notify the user through a callback mechanism.

2.3 Basic Process

The process of using the ZCEAPI in the self-driven mode is largely as follows:

2.3.1 Initialize the ZCEAPI

A user calls the function API_Init to initialize the ZCEAPI.

A successful initialization is the basis on which to correctly use the ZCEAPI. If the initialization process fails, the API may not operate smoothly.

Example:

```
if (API_Init("./log", "./APIData") == API_TRUE)
{
	std::cout << "Succeed in Init API!!!" << std::endl;

}
else
{

	std::cout << "Fail to Init API!!!" << std::endl;
	return 1;

}
```

Note: For the use of the initialization function API_Init, please refer to Part 4 of this handbook.

2.3.2 Create an Exchange Connection

A user calls the function API_CreateExchgConnection to create a connection. To access the Exchange through the ZCEAPI, a user must create a connection in the first place. The connection handle returned by the CreateExchgConnection function is the basis of the communication with the Exchange.

Example:

```
API_BOOL Encrypt = API_TRUE;                //to encrypt? 1: Yes, 0: No
API_BOOL Commpress = API_TRUE;              //to compress? 1: Yes, 0: No

MARKET_ID Market_ID = MARKET_ZCE;   //i.e., ZCE 1

int cpunum_for_driver = 1;      //Bind the connected driver thread to a designated CPU
                                core (coded "1")


int cpunum_for_udp = - 1;       //The UDP receive thread here is not bound to the CPU
                                (In this example, the user logs in to a conversational
                                stream, which will not initiate the UDP receive thread.
                                Therefore, there is no need to assign a CPU number to
                                this parameter. Even if it is assigned, it is void.)
```

```
//Create a connection for the conversational stream
ExchgConnectionHandle DialogConn = API_CreateExchgConnection(Encrypt,
Commpress, Market_ID, cpunum_for_driver, cpunum_for_udp);

if (DialogConn == NULL)
{
    std::cout << "Can not create Dialog connection to ZCE ! " <<
    std::endl; return 3;
}
```

Notes:

1) For the use of the function API_CreateExchgConnection, please refer to Part 4 of this handbook. Creating a connection does not mean establishing a connection to the Exchange.

2) The function API_CreateExchgConnection has deleted heartbeat parameters, so that the heartbeat interval and heartbeat timeout will no longer be assigned by users themselves.

2.3.3 Assign a Callback Function to a Link

Once a link is successfully created, a user can then assign a callback function to the link. It is recommended that the callback function be assigned immediately after a link is created.

The assignment of a callback function is not a must (as in the case that a user is not concerned about what happens to a link), but if a link is not assigned such function, it may affect the fulfillment of some transactions of a user, in that the user will not be informed of some link events.

2.3.3.1 Define a Callback Function

A user must define a callback function before assigning it to a link. The callback functions in the ZCEAPI fall into two categories: those for the status of a link and those for the return of data packets.

For the definition of the categories of callback functions, please refer to the definition of callback functions in Part 3 of this handbook.

Example:

```
//Define a callback function for an error
void errorsShow(void  * CallBackArg,ExchgConnectionHandle,int error_code,const char*

error_text)
{
      cout<<" link error ！   "<<" The error is about "<<error_text<<"  The error code is：

"<<error_code<<endl;
}
```

2.3.3.2 Set a Callback Function

Once a callback function is defined, a user can call a ZCEAPI function to set the callback function.

Example:

```
//Set a callback function for a link error
API_SetErrorCallBack(conn,  errorsShow,this);     //The  third  is  a  callback
parameter

//Set a callback function for link close
API_SetCloseCallBack(conn, OnAPIClose,NULL);
```

Note: For details of the setting of callback function interfaces, please refer to Part 4.7 of this handbook.

2.3.4 Connect to the Exchange

It is recommended that before initiating a connection to the Exchange, a user call the function API_SetConnectionOpt to set the TCP heartbeat for the link. This can help to prevent the operating system from taking too much time to detect a link error when the socket is disconnected in a non-graceful manner.

A user must call the function API_Connect to initiate a connection to the Exchange. If the connection is successful, a connection channel is thus opened. (The first parameter of the API_Connect function is a connection handle. In the case of a successful connection, this connection object is then opened. Subsequently, this connection object can be used to communicate with the Exchange.)

Example:

```
//Initiate a connection to the Exchange

char IpAddr[20] = "218.29.68.231";          //The IP address of the login service of the
int port = 22677;                           Exchange
char ErrMsg[128];                           //Connect to the port of the Exchange
if (API_Connect(DialogConn, IpAddr, port, 5000, ErrMsg)  !=ERR_SUCCESS)
{//Connection failed

      cout<<" Can not connect with CZCE！ "<<endl;
      return 0;

}
else
cout<<" Connect CZCE successfully "<<endl;
```

Notes:

1) For the function API_Init, please refer to Part 4 of this handbook.

2) Once a connection is established, a user shall log in to the trading system of the Exchange as soon as possible; otherwise, the connection will become null (where no data stream is accessed) and, after some time, be automatically shut down.

3) Different business streams use separate port numbers. Please assign the numbers according to notices of the Exchange.

2.3.5 Log in to the Exchange

Once a connection is established, a user can then log in to the system of the Exchange through the connection. **Through a connection, a user can log in to only one of the conversational, private, and broadcast streams.**

The process of logging in to a stream is described as follows:

1) call the API_AllocPackage function to generate a ZCEAPI data packet;

2) call the API_SetPID function to set a PID for the data packet (the PID for login to messages is: 0x00016);

3) call an appropriate field function to complete the fields for the data packet (e.g., API_SFInt (APIpkg, FID_SequenceNo, 1));

4) call the API_Login function to log in to the Exchange;

5) judge whether the login is successful according to the value returned by the API_Login function, and obtain the information contained in the login response packet (if the return value is not void); and

6) call the API_FreePackage function to release the data packet generated in step 1.

Notes:

1) For the use of specific functions, please refer to Part 4 of this handbook. For examples, please refer to the APIDemo released together with this document.

2) Once used, a data packet need not necessarily be released. It can be used again, but must be revised or emptied given the reality.

3) Because the login timeout parameter in the API_Login function is set too small or the network connectivity is poor, a user may encounter login timeout and therefore need to adjust the wait value.

4) Even if a user opts to receive a UDP broadcast stream according to the protocol version 11, he must log in to the broadcast stream in the first place.
ProtocolVersion=2, the TCP method to be adopted for the broadcast stream
ProtocolVersion=11, the UDP method to be adopted for the broadcast stream

2.3.6 Read and Write Data Packets

Having received a valid ZCEAPI data packet handle (which may be generated by a ZCEAPI interface function or conveyed by a callback function), a user can read and write the data packet pointed by the handle. Reading and writing the data packet is a primary means for a user to obtain and send data through the ZCEAPI.

It is quite easy to read and write the ZCEAPI data packets. To do so, a user needs only to select an appropriate ZCEAPI function according to the type of a data field (for details, please refer to the *ZCEAPI Reference Manual*'s provisions governing the categories of fields for data packets) to be read and written and then do the reading and writing accordingly.

Example:

According to the *ZCEAPI Reference Manual*, we can find out that the FID_BuyPrice field is of the LN-4 data type, that is, doubles with the accuracy of 4 decimal places. Accordingly, we can read and write the field, using the functions API_GFDouble and API_SFDouble provided by the ZCEAPI to address doubles.

```
double d = API_GFDouble(APIpkg, FID_BuyPrice);          //read
API_SFDouble(APIpkg, FID_BuyPrice, 1555.552, 4);        //assign value
```

Notes:

1) In reading and writing data packets, a user must, according to the type of the data field to be read and written, select appropriate ZCEAPI functions designed to do so. For the use and applicability of specific functions, please refer to Part 4 of this handbook.

2) For use cases, please refer to relevant part of the APIDemo.

2.3.7 Traverse Data Packets

Traversal is a supplement to the reading and writing of data packets. On obtaining a valid data packet, a user can traverse it immediately. Traversal allows for no writing, but reading of the data packet only.

Before traversing a data packet, a user must call the API_FirstField function to identify the first field and then call the API_NextField function to do the traversal. To use a traversal function to read the value of a field, a user must correctly identify the data type of the field and choose an appropriate function.

Notes:

The order of fields in a ZCEAPI data packet has nothing to do with the order of fields written to the packet. Before the API_FirstField and API_NextField operations come to an end, a user can not add or delete fields in a data packet.

The traversal function is non-thread safe, so do not traverse a data packet through multiple threads at the same time.

For the functions used to traverse data packets, please refer to Part 4 of this handbook.

For use cases, please refer to relevant part of the APIDemo.

2.3.8 Send Data Packets

On logging in to a conversational stream, a user can call the API_Send function to send various data packets to the Exchange. The process of sending a data packet is largely the same as that of login:

1) call the API_AllocPackage function to generate a ZCEAPI data packet;

2) call the API_SetPID function to set a PID for the data packet;

3) call a corresponding function to complete the fields in the data packet;

4) call the API_Send function to send the completed data packet (**Note: The API_Send function for the same connection object is non-thread safe**);

5) judge whether the packet has been successfully sent according to the value returned by the API_Send function; and

6) call the API_FreePackage function to release the data packet generated in step 1.

Note: For the use of specific functions, please refer to Part 4 of this handbook. Once used, a data packet need not necessarily be released. It can be used again, but must be revised or emptied given the reality.

2.3.9 Receive Data

The ZCEAPI will automatically receive the data sent by the Exchange and convert them into data packets. A user can receive the ZCEAPI data packet by the following methods:

- **Receiving login response data packets**

  through the output of the login function. For details, please refer to the definition of the API_Login function.

- **Receiving logout response data packets**

  through the output of the logout function. For details, please refer to the definition of the API_Logout function.

- **Receiving ordinary Exchange response data packets**

  through setting callback functions to receive data packets

Note: For use cases, please refer to relevant part of the APIDemo.

2.3.10 Log out

On finishing all work, a user can log out of the Exchange's trading system. In essence, logout is to send a logout data packet to the system of the Exchange, and, when receiving a correct logout confirmation from the exchange or the link is disconnected, the exit is done.

A user can selectively log out of any stream he has logged in to, but he can only exit one stream at a time.

Logout basically follows the same steps as login:

1) call the API_AllocPackage function to generate a ZCEAPI data packet;

2) call the API_SetPID function to set a PID for the data packet (logout PID: 0x00017);

3) call an appropriate field function to complete the fields for the data packet (e.g., API_SFInt (APIpkg, FID_SequenceNo, 1));

4) call the API_Logout function to send the completed data packet;

5) judge whether the logout is successful according to the value returned by the API_Logout function, and according to the output of the function (if the output value is not void), obtain the logout response returned by the Exchange; and

6) call the API_FreePackage function to release the data packet generated in step 1.

Notes:

1) For the use of specific functions, please refer to Part 4 of this handbook.

2) Once used, a data packet need not necessarily be released. It can be used again, but must be revised or emptied given the reality.

2.3.11 Free a Connection

After logging out of the Exchange's system and before exiting an application, a user shall break his link with the Exchange first and then release his connection.

A user can directly call the API_DisConnect function to break a link with the Exchange. That done, a link with the Exchange will be severed, but **the connection** to the Exchange will remain. So, a user can either call the API_Connect function to connect to the Exchange again or call the API_FreeExchgConnection function to directly break the link and release the connection.

Notes:

Once released, a connection cannot be used again.

For the use of specific functions, please refer to Part 4 of this handbook.

2.3.12 Stop the ZCEAPI Service

It is advised that, before exiting an application that calls the ZCEAPI, a user should in the first place call the API_Stop function to cease the service of the API. Here, the service cessation means to stop the diagnostic function of the ZCEAPI.

Further, it is advised that a user not arbitrarily stop the service of the ZCEAPI; otherwise, the work of the API may be adversely affected.

3. Definition of Data

3.1 Basic Types of Data

```
typedef   unsigned short int   API_UINT16;
typedef   unsigned char        API_BYTE;
typedef   unsigned int         API_UINT32;
```

3.2 Types of Dates and Time

```
//1-Bit aligned
typedef struct tagDateTime
{
    API_UINT16    year;
    API_BYTE      month, day, hour, minute, second;
    API_UINT32    microsec;
}API_DateTime;
```

3.3 API_BOOL

```
typedef int API_BOOL;
#define API_TRUE1
#define API_FALSE    0
```

3.4 Market ID

```
typedef int MARKET_ID;
#define MARKET_ZCE 1
```

3.5 Definition of the Types of Data Fields

```
enum API_FIELDTYPE
{
    FTNULL = 0,
    FTCHAR = 1,
    FTLONG = 2,
    FTSTRING = 3,
    FTDOUBLE = 4,
    FTDATETIME = 5,
    FTINT64 = 6                    //This type is not to be used for the time being.
};
```

3.6 Flagging of Data Streams

```
typedefAPI_BYTE API_DFFLAG;
#define DFF_DIALOG          0    //conversational stream
#define DFF_PRIVATE         1    //private stream
#define DFF_BROADCAST       2    //broadcast stream
```

3.7 Status of Data Streams

```
//status of data streams
enum API_DFSTATUS
{
    DFS_CLOSED = 0,              //connection closed
    DFS_OPENED = 1,              //connection opened
    DFS_NEGOTIATEKEY = 2, //key negotiation under way, not logged in yet
    DFS_LOGIN_OK = 3             //login is OK.
};
```

3.8 Handles

The handles of connections and data packets are stated to be class pointers in the C++ language, or void pointers in the C language.

```
#ifdef __cplusplus
class MsgPackage;
class ExchangeConnection;
/*data packet handles*/
typedef MsgPackage* MsgPackageHandle;
/*exchange connection handles*/

typedef ExchangeConnection* ExchgConnectionHandle;
#else
/*data packet handles*/
typedef void* MsgPackageHandle;
/*exchange connection handles*/
typedef void* ExchgConnectionHandle;
#endif
```

3.9 Callback Functions for Return of Data Packets

```
typedef int (*ExchgPackageCallBack)(void * CallBackArg, ExchgConnectionHandle, MsgPackageHandle);
```

Note: ExchgConnectionHandle is the handle provided by the ZCEAPI for a connection.

MsgPackageHandle is the handle provided by the ZCEAPI for a data packet.

3.10 Callback Functions for the Status of a Link

```
typedef void (*ExchgConnectionCallBack)(void * CallBackArg, ExchgConnectionHandle, int error_code, const char* error_text);
```

Note: ExchgConnectionHandle is the handle provided by the ZCEAPI for a connection.

4. Introduction to Functions

4.1 Management of the ZCEAPI

4.1.1 Get Version Numbers of the ZCEAPI

```
int API_GetVersion(char* versionbuf, unsigned int* buflen);
```

Function: To get the character string representing the version number of the ZCEAPI

Parameter specs: 1) char * versionbuf pointer for the buffer zone used to receive the version number string;

2) unsigned int buflen length of the buffer zone used to receive the version number string

Return value: If successful, return a version number string;

If the buffer zone length is not sufficient, then return "-1". In this case, the buflen returns the minimum length required of the buffer zone.

Tips: The normal format for a string is x.x.x.x (x is an integer), for example, 2.10.16.35.

In principle, the length of a string will not exceed 16 bytes.

4.1.2 Initialize the ZCEAPI

```
API_BOOL API_Init (const char* logFilePath, const char* APIDataPath);
```

Function: To initialize the ZCEAPI, and designate diagnostic log routes and diagnostic ports

Parameter specs: 1) logFilePath path to the API's log file. The path must be already in place and a user must have the right to write to it.

2) APIDataPath path to store private API data. The path must be already in place and a user must have the right to write to it.

Return value: If initialization is successful, then return API_TRUE; otherwise, return API_FALSE.

Tips: 1) **The LogFilePath must be already in place.** If not, the ZCEAPI will not automatically create a link and this will lead to a failure of the initialization.

2) The APIDataPath is not used for the time being, so the non-existence of this path will not affect the initialization of the API.

3) The ZCEAPI will be initialized only once for each app.

4) If a user chooses to log in to the Exchange's trading system through the protocol version 11, the default UDP port used to receive quotations is 22677. If a user needs to change this port, he must, before the initialization process, use the APIConfig.dat function to designate an alternative port number at the working directory of the app currently in use. For example, if "22899" is written to the directory, then the API will open the UDP port according to the given number. The port must, however, be configured according to the interfaces released by the Exchange.

5) A successful initialization of the ZCEAPI is the foundation of all subsequent work. If initialization fails, there is no guarantee for the success of subsequent operations.

4.1.3 Stop the ZCEAPI Service

```
void API_Stop();
```

Function: To stop the use of the API library

Tips: If this function is called, please do not run any other functions in the ZCEAPI library, unless the API is re-initialized (caution: please do not stop and then initialize the API time and again).
It is recommended that this function be called only when a user is ready to exit an app.

4.1.4 Get Current Time

```
int API_Now(API_DateTime*dt);
```

Function: To get the current time of a local system

Parameter specs: API_DateTime*dt pointer of the APIDateTime data used to store current time

Return value: If the time is successfully obtained, return “0”; otherwise, return “-1”.

Tips: The local time obtained from a current system through this function is stored in dt, so it is related to the setting of time zones in the local system.

4.2 Management of Data Packets

4.2.1 Create a Data Packet

```
MsgPackageHandle API_AllocPackage();
```

Function: To create a ZCEAPI data packet

Return value: If the operation is successful, return a handle for the data packet created; otherwise, return “NULL”.

Tips: Before creating a data packet, a user must ensure that the ZCEAPI is successfully initialized. Otherwise, he will fail to do so.

4.2.2 Recycle a Data Packet

```
void API_FreePackage(MsgPackageHandle pkg);
```

Function: To free an existing data packet

Parameter specs: MsgPackageHandle pkg handle created by the ZCEAPI for a data packet.

Tips: 1) The parameter completed must be a data packet handle allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.
2) Once a data packet is recycled, it cannot be used again.

4.3 Operations of Message Data

4.3.1 Get a Message ID

```
int API_GetPID(MsgPackageHandle pkg);
```

Function: To get the PID of a designated message

Parameter specs: MsgPackageHandle pkg handle of a data packet for which the PID is to be obtained

Return value: If successful, return the PID of the designated data packet; otherwise, return "0xFFFFF".

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4.3.2 Set a Message ID

```
void API_SetPID(MsgPackageHandle pkg,int pid);
```

Function: To set the PID of a designated data packet

Parameter specs: 1) MsgPackageHandle pkg data packet for which the PID is to be set

2) int pid value of the PID to be set

Tips: 1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user must fill in a meaningful PID (please refer to the *ZCEAPI Reference Manual*).

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4.3.3 Get the Data Type of a Field

```
enum API_FIELDTYPE API_GetFieldType(MsgPackageHandle pkg,int fid);
```

Function: To get the current data type of a field in a designated data packet

Parameter specs: 1) MsgPackageHandle pkg handle of the designated data packet

2)API_UINT16 fid number of the field for which the type of data is to be obtained (please refer to the ZCEAPI Reference Manual).

Return value: If successful, return the number of the type of the data field coded "fid" in the data packet pointed by pkg; otherwise, return "FTNULL". For specific types of data fields, please refer to the "**Definition of the Types of Data Fields**" part of this handbook.

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4.4 Operations of Fields

4.4.1 Get Characters from a Data Packet

```
char API_GFChar(MsgPackageHandle pkg,int fid,char def=' ');
```

Function: To get characters from a designated data field in a designated data packet

Parameter specs:
1) MsgPackageHandle pkg — handle of the designated data packet
2) API_UINT16 fid — ID of the data field from which the characters are to be obtained
3) char def — default value. In the case of a failure, the return value will be a default one. In the C++ language, the default value is a space.

Return value: If successful, return the characters in the data field coded "fid" in the data packet pointed by pkg; otherwise, return "def".

Tips:
1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "C (Character)".

3) If used to address non-C (Character) fields, this function will make adaptations accordingly, but does not guarantee the correctness of the value returned.

4) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the operation will end up in a failure.

4.4.2 Set Character-type Data Fields in a Data Packet

```
int API_SFChar(MsgPackageHandle pkg,API_UINT16 fid,char val);
```

Function: To set character value for a designated data field in a designated data packet

Parameter specs:
1) MsgPackageHandle pkg — handle of the designated data packet
2) API_UINT16 fid — ID of the data field for which the character value is to be set
3) char val — character value to be set

Return value:
0: successful assignment of value
- 1: wrong type of value for the field
-99: illegal data packet

Tips:
1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.
2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "C (Character)".
3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the assignment of value will end up in a failure.

4.4.3 Get Integers from a Data Packet

```
int API_GFInt(MsgPackageHandle pkg,API_UINT16 fid,int def=0);
```

Function: To get integers by integer type from a designated data field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg	handle of the designated data packet
2) API_UINT16 fid	ID of the data field from which the characters are to be obtained
3) int def	default value. In the case of a failure, the return value will be "0" by default.

Return value: If successful, return the integers in the data field coded "fid" in the data packet pointed by pkg; otherwise, return "def".

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields whose fid are SI (short integers), N (integers) or UN (unsigned integers).

3) If a data field is of the UN type, a user must convert the return value to unsigned integers.

4) If used to address non-matched fields, this function will make adaptations accordingly, but does not guarantee the correctness of the value returned.

5) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the operation will end up in a failure.

4.4.4 Set Integer-type Data Fields in a Data Packet

```
int API_SFInt(MsgPackageHandle pkg,API_UINT16 fid,int val);
```

Function: To set integer value for a designated data field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg	handle of the designated data packet
2) API_UINT16 fid	ID of the field for which the value is to be set
3) char val	integer value to be set

Return value:

0:	successful assignment of value
- 1:	wrong type of value assigned for the field
-99:	illegal data packet

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields whose fid are SI (short integers), N (integers), or UN (unsigned integers).

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the assignment of value will end up in a failure.

4.4.5 Get Doubles from a Data Packet

```
double API_GFDouble(MsgPackageHandle pkg,API_UINT16 fid,double def=0.0);
```

Function: To get a value in the format of doubles from a designated data field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg	handle of the designated data packet
2) API_UINT16 fid	ID of the data field from which the characters are to be obtained
3) double def=0.0	In the case of a failure, the return value will be "0" by default.

Return value: If successful, return the doubles in the data field coded "fid" in the data packet pointed by pkg; otherwise, return "def".

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "LN (double)".

3) If used to address non-matched fields, this function will make adaptations accordingly, but does not guarantee the correctness of the value returned.

4) In the case that a field to be retrieved is completed by a user himself (using the API_SFDouble function to be introduced below), if, after completion, the pkg is not transmitted to the backend system or not copied, then whatever completed by the user will be retrieved through this function. That's to say, the level of accuracy completed will not work in this case.

5) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the operation will end up in a failure.

4.4.6 Set a Double-type Data Field in a Data Packet

```
int API_SFDouble(MsgPackageHandle pkg,API_UINT16 fid,double val,unsigned int precision=4);
```

Function: To set double value for a designated data field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg	handle of the designated data packet
2) API_UINT16 fid	ID of the data field for which the value is to be set
3) double val	Double value to be set
4) unsigned int precision	The level of accuracy of the double to be set, "4" binary points by default, "1" binary point on the minimum level

Return value:

0: successful assignment of value

- 1: wrong type of value for the field

-99: illegal data packet

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "LN (double)".

3) Precision means the number of binary points. If the number of a value's binary points is greater than is indicated by the precision, then the value will retain only the binary points indicated by the level of precision when sent to the Exchange. If the value is not transmitted or copied, there will be no loss in the level of precision for the same value gotten through the API_GFDouble function.

4) When setting the precision of doubles, a user must refer to the level of precision provided for data fields in the *ZCEAPI Reference Manual*. Should the precision set be not consistent with that provided in the *Reference Manual*, the value set will be processed at the level of precision specified in the *Reference Manual* when it is sent to the Exchange.

5) Zhengzhou Commodity Exchange's phase 5 system supports 12-bit integers with 2-bit precision or 8-bit integers with 4-bit precision at the maximum (for details, please refer to the *Reference Manual*). If there are more floating points than allowed, the value will not be effectively conveyed.

6) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the assignment of value will end up in a failure.

4.4.7 Get Character Strings from a Data Packet

```
int API_GFString(MsgPackageHandle pkg,API_UINT16 fid, char* buf,unsigned int bufsize);
```

Function: To get a value in the format of character strings from a designated data field in a designated data packet

Parameter specs:
1) MsgPackageHandle pkg handle of the designated data packet
2) API_UINT16 fid ID of the data field from which the characters are to be obtained
3) char * versionbuf pointer for the buffer zone used to receive the character string
4) unsigned int buflen length of the buffer zone used to receive the character string

Return value: Return value>0: if the return value is equal to or smaller than the bufsize, then return the full length of the character string obtained; if the return value is greater than the bufsize, then return the length of the buffer zone required to get the entire data, because it means that the buffer zone is too small and, subject to the size of the buf, only part of the target data is copied to it.

Return value=0: it means a failure to get the value from a field, or it simply means that the parameters given cannot get the value at all (as in the case that the buf is empty and the bufsize is 0).

Tips:
1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.
2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "S (character string)".
3) If used to address non-matched fields, this function will make adaptations accordingly, but does not guarantee the correctness of the value returned.
4) The bufsize given must be large enough; otherwise, subject to the size of the buf given, only part of the characters will be copied to the buffer zone.
5) However, the value of the bufsize shall not be greater than the actual size of the buf; otherwise, memory errors may occur.
6) The character string in this function is **not necessarily visible**. In the case of a successful operation, the character string returned is full-length. Note that the character string received in the buffer zone has no EOT. Therefore, if a user wants to process the buf as character strings, he must add EOT on his own.

7) If this function is used to address the DT-type of data fields, five circumstances may arise:

① If, in a DT object, none of the year, month, date, hour, minute, second, and microsecond values is 0, then the format of the character string gotten shall be: YYYY-MM-DD HH:mm:ss.iiiiii

② If, in a DT object, none of the year, month, date, hour, minute, and second values is 0, but the microsecond value is 0, then the format of the character string gotten shall be: YYYY-MM-DD HH:mm:ss

③ If, in a DT object, none of the year, month, and date values is 0, but the hour, minute, second, and microsecond values are all 0, then the format of the character string gotten shall be: YYYY-MM-DD

④ If, in a DT object, the hour, minute, and second values are not 0, but the date value is 0, then the format of the character string gotten shall be: HH:mm:ss.iiiiii

⑤ If, in a DT object, none of the hour, minute, second values is 0, but the date and microsecond values are 0 (when only time is available), then the format of the character string gotten shall be: HH:mm:ss

Note: The signs mentioned above have the following meaning: Y: year; M: month; D: date; H: hour; m: minute; s: second; i: microsecond (e.g., YYYY means a year in four digits, iiiiii means a microsecond in six digits)

8) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the operation will end up in a failure.

4.4.8 Set Character-string-type Data Fields in a Data Packet

```
int API_SFString(MsgPackageHandle pkg,API_UINT16 fid,const char* buf,unsigned int bufsize);
```

Function: To set a value in the format of character strings for a designated data field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg handle of the designated data packet

2) API_UINT16 fid ID of the data field for which the character value is to be set

3) char* buf head pointer of a character string

4) unsigned int bufsize length of a character string

Return value:

0: successful assignment of value

-1: wrong type of value for the field

-2: bufsize 0, and the fid not in existence

-99: illegal data packet

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid “S (character string)”.

3) If bufsize is the length of a character string, then the former must not be smaller than the latter; otherwise, only characters in the size of the buf will be completed.

4) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the assignment of value will end up in a failure.

4.4.9 Get Date and Time from a Data Packet

```
API_BOOL API_GFDateTime(MsgPackageHandle pkg,API_UINT16 fid,API_DateTime*val);
```

Function: To get a value in the format of date and time from a designated field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg handle of the designated data packet

2) API_UINT16 fid ID of the data field from which the characters are to be obtained

3) API_DateTime*val pointer of date in the ZCEAPI, pointing to variables that store the date gotten (parsed per Beijing Time (UTC+8)). For the types of date and time in the ZCEAPI, please refer to the part "Types of Date and Time" of this handbook.

Return value: If successful, then return API_TRUE; otherwise, return API_FALSE.

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "D (Date)" or "DT (Date and Time)".

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the operation will end up in a failure.

4) The API time is parsed per Beijing Time (UTC+8). That's to say, a time value gotten is to be explained according to Beijing Time. For instance, API_DateTime (2019,11,23,9,45,08,234208) means 09:45:08:234208 on November 23, 2019 Beijing Time.

4.4.10 Set DateTime-type Data Fields in a Data Packet

```
int API_SFDateTime(MsgPackageHandle pkg,API_UINT16 fid,API_DateTime*val);
```

Function: To set a value in the format of date and time for a designated field in a designated data packet

Parameter specs:

1) MsgPackageHandle pkg handle of the designated data packet

2) API_UINT16 fid ID of the data field for which the value is to be set

3) API_DateTime*val pointer of date in the ZCEAPI, pointing to variables that store the date set (parsed per Beijing Time (UTC+8))

For the types of date and time in the ZCEAPI, please refer to the part "Types of Date and Time" of this handbook.

Return value:

0: successful assignment of value

-1: wrong type of value for the field

-2: non-transmittable time (locally stored, non-transmittable and non-copyable)

-99: illegal data packet

Tips:

1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) A user is required to refer to the data fields and data field types defined in the *ZCEAPI Reference Manual*. This function is applicable to data fields represented by the fid "D (Date)" or "DT (Date and Time)".

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized; otherwise, the assignment of value will end up in a failure.

4) The API time is parsed per Beijing Time (UTC+8). That's to say, a time value set is to be explained according to Beijing Time. For instance, API_DateTime (2019,11,23,9,45,08,234208) means 09:45:08:234208 on November 23, 2019 Beijing Time.

5) The time set shall fall in the range of 08:00:00, January 1, 1970 to 23:59:59, December 31, 3000 UTC Time. A user can judge whether a time falls outside of the range according to a return value.

4.4.11 Judge Whether a Field Is Null

```
API_BOOL API_FieldIsNull(MsgPackageHandle pkg,API_UINT16 fid);
```

Function:	To judge whether a designated field in a designated data packet is null
Parameter specs:	1) MsgPackageHandle pkg handle of the designated data packet 2) API_UINT16 fid ID of the data field for which a judgment is to be made
Return value:	If a designated field is null, then return API_TRUE; otherwise, return API_FALSE.
Tips:	1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise. 2) This function is equivalent to API_GetFieldType() == API_FIELDTYPE::FTNULL. 3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4.4.12 Remove a Particular Field

```
void API_ClearField(MsgPackageHandle pkg,int fid);
```

Function:	To remove a designated field from a designated data packet
Parameter specs:	1) MsgPackageHandle pkg handle of the designated data packet 2) API_UINT16 fid ID of the data field to be removed
Tips:	1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise. 2) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4.4.13 Remove All Fields from a Data Packet

```
void API_ClearAll(MsgPackageHandle pkg);
```

Function:	To remove all fields from a data packet
Parameter specs:	MsgPackageHandle pkg handle of the designated data packet

Tips: 1) The parameter completed must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.
2) All fields, excluding PID, are to be cleared.

4.4.14 Copy a Data Packet

```
int API_Copy(MsgPackageHandle pkg,MsgPackageHandle source);
```

Function: To copy the data designated by a source to a data packet identified by a pkg

Parameter specs: 1) MsgPackageHandle pkg handle of the designated target data packet
2) MsgPackageHandle source handle of the designated source data packet

Return value: If the copy is successful, then return "0"; otherwise, return an error code.

Tips:
1) The parameters pkg and source must be data packets allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) The PID of a data packet is copied as well.

3) Due to the operation of this function, a double set by a user may retain the set level of precision only.

4.5 Traversal of Data

4.5.1 Move a Data Field Pointer to the First Field

```
int API_FirstField(MsgPackageHandle pkg);
```

Function: To find out the first data field in a data packet to be traversed

Parameter specs: MsgPackageHandle pkg handle of the target data packet

Return value:
0: fid of the first field traversed
-1: null data packet
-99: illegal data packet

Tips:
1) The parameter pkg must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) The order of data fields is different from that of the assignment of value to a data packet.

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4) This function must be called before traversal.

5) No data fields can be added or deleted before the end of traversal.

6) This function is non-thread safe, so do not traverse a data packet through multiple threads at the same time.

4.5.2 Identify the Next Data Field

```
int API_NextField(MsgPackageHandle pkg);
```

Function: To get the fid of a next data field in a data packet traversed

Parameter specs: MsgPackageHandle pkg handle of the target data packet

Return value:
>0: fid of the next field traversed
-1: traversal of a data packet over
-99: illegal data packet

Tips:
1) The parameter pkg must be a data packet allocated by the API_AllocPackage function but not yet freed; otherwise, anomaly will arise.

2) The order of data fields is different from that of the assignment of value to a data packet.

3) Before using this function, a user must ensure that the ZCEAPI is successfully initialized.

4) No data fields can be added or deleted before the end of traversal.

5) This function is non-thread safe, so do not traverse a data packet through multiple threads at the same time.

4.6 Management of Connection

4.6.1 Create Exchange Connection

```
ExchgConnectionHandle API_CreateExchgConnection(API_BOOL Encrypt, API_BOOL Commpress, MARKET_ID Market ,int thread_bind_cpu, int thread_bind_cpu_udp, API_BOOL drive_auto = API_TRUE);
```

Function: To create an exchange connection

Parameter specs:

1) API_BOOL Encrypt	encrypt? API_TRUE Yes, API_FALSE No Only Yes is supported at present.
2) API_BOOL Commpress	compress? API_TRUE Yes, API_FALSE No
3) MARKET_ID Market	ID of the exchange connected. Please refer to the "**market ID**" in this handbook.
4) thread_bind_cpu	number of the CPU to which a driver thread is bound
5) thread_bind_cpu_udp	number of the CPU to which a driver thread is bound and where a broadcast stream is received through the UDP protocol
6) drive_auto	Whether to automatically initiate a business-driven thread

Return value: If successful, return the handle of an exchange connection; otherwise, return a null pointer.

Tips:
1) Before creating an exchange connection, a user must ensure that the ZCEAPI is successfully initialized. Otherwise, he will fail to do so.

2) If the "number of the CPU (starting from 0)" is negative or greater than the number of CPUs of a computer, then a driver thread is not to be bound to the CPU. If bound, the thread will occupy the CPU core all the time.

3) If a broadcast stream is not received through the UDP protocol, the parameter thread_bind_cpu_udp will not be effective, because the driver thread concerned will not be initiated.

4) If the drive_auto parameter is set API_TRUE or left unset, then a connection will automatically initiate a business-driven thread. If, however, the parameter is set API_FALSE, then the API will not initiate a driver thread for the connection. In this case, a user must initiate the thread on his own and call the API_RunEvent function for the connection.

4.6.2 Set Connection Properties

```
int API_SetConnectionOpt(ExchgConnectionHandle conn,int keepIdle,int keepInterval, int keepCount);
```

Function:	To set the properties of a connection	
Parameter specs:	1) ExchgConnectionHandle conn	Handle of an exchange connection
	2) int keepIdle	Idle time of a connection. The unit is second for Linux platforms, or microsecond for Windows platforms.
	3) int keepInterval	Interval for the detection of a connection The unit is second for Linux platforms, or microsecond for Windows platforms.
	4) int keepCount	Times of detection
Return value:	If successful, return "0"; otherwise, return an error code.	
Tip:	This function must be called before a connection is initiated; otherwise, the setting will be void.	

4.6.3 Run an Event

```
int FTDAPI_CALL API_RunEvent(ExchgConnectionHandle conn);
```

Function:	This is an event-driven function to be called by a user's own thread when the user calls the API_CreateExchgConnection function to create a connection, assigning the value API_FALSE to the drive_auto parameter, and using his own thread to replace an automatic thread to drive a connection.	
Parameter specs:	ExchgConnectionHandle conn	handle of an exchange connection
Return value:	0: event successfully processed	
	3000: wrong parameter	
	3029: illegal call	
	Other values: an error encountered, link disconnected, and event processing over	
Tips:	1) If, when creating a connection, a user chooses to automatically initiate a business-driven thread (drive_auto = API_TRUE), he is not advised to call this function.	

2) This function is non-thread safe, so do not call it through multiple threads at same time.

4.6.4 Initiate a Connection to the Exchange

```
int API_Connect(ExchgConnectionHandle conn, const char* IP, unsigned short port, int wait, char* errMsg);
```

Function:	To establish a connection to the exchange.	
Parameter specs:	1) ExchgConnectionHandle conn	Handle of an exchange connection
	2) const char* IP	IP address of trade front
	3) unsigned short port	Trade front port
	4) int wait	Waiting time in the case of timeout, unit-microsecond
	5) char * errMsg	Output. The memory used to store an error message returned shall be allocated by a user himself and be in a size of 64 bytes at least.
Return value:	0: connected	
	3000: wrong parameter	
	3027: callback failure	
	3035: connection to the trade front failed	
Tips:	1) The parameter conn must be the handle of a connection created by the ZCEAPI. 2) Once an exchange connection is established, a user shall log in to the exchange's trading system as soon as possible; otherwise, the connection will be automatically shut down.	

4.6.5 Judge Whether a Connection Has Been Established

```
API_BOOL API_Connected(ExchgConnectionHandle conn);
```

Function:	To judge whether a connection to the exchange has been successfully established	
Parameter specs:	ExchgConnectionHandle conn	handle of an exchange connection
Return value:	In the case of a successful connection, return API_TRUE; otherwise, return API_FALSE.	

4.6.6 Log in to the Exchange

```
int API_Login(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait , MsgPackageHandle* rspPkg);
```

Function:	To log in to any of the three types of data streams to connect to the exchange's trading system	
Parameter specs:	1) ExchgConnectionHandle conn	Handle of an exchange connection
	2) MsgPackageHandle reqPkg	Handle of the ZCEAPI packet containing login information;
	3) double Wait	waiting time in the case of login timeout, unit: second
	4) MsgPackageHandle * rspPkg	Output, pointer of the handle of a login response data packet. In the absence of response, the pointer is null.

Return value:

0: Login is OK.

3000: Wrong parameter

3001: Connection yet to be established

3002: Error in sending link negotiation data

3003: Link negotiation timeout

3004: Error in sending login data

3005: Login timeout

4001: Error in login protocol version number

3018: Error in flagging data stream

3019: Duplicate login not allowed

3020: You are logging in to an undesignated seat.

3022: Key agreement verification failed

3027: Driver thread cannot be called.

3051: Generation error of key

Backend error codes

2007: incorrect login mode of data flow

610: illegal trader code

262: trader has been suspended.

673: illegal login request

674: illegal password

675: The system cannot be logged in from this workstation.

2005: login protocol version error

40: unable to log in the system from this front, too many private stream data requests

7001: duplicate login

Tips: 1) The parameter conn must be the handle of an established connection to the exchange.

2) A user must correctly complete such information as the ID of a trader or data stream, password and more in the data packet pointed by reqPkg. For details, please refer to the *ZCEAPI Reference Manual*.

3) This is a blocking function. If the value of the parameter wait is not large enough, the function will return API_FALSE for timeout. For that reason, the value of wait shall not be too small, let alone zero.

4) The MsgPackage pointed by rspPkg is read-only.

5) This is a synchronous function, so please do not call it from the callback functions of the API.

4.6.7 Log out

```
int API_Logout(ExchgConnectionHandle conn,MsgPackageHandle reqPkg, double Wait, MsgPackageHandle* rspPkg);
```

Function: To log out of any of the three types of data streams

Parameter specs:
1) ExchgConnectionHandle conn handle of an exchange connection
2) MsgPackageHandle reqPkg handle of the data packet that contains logout information
3) double Wait waiting time in the case of login timeout unit: second
4) MsgPackageHandle *rspPkg output, pointer of the handle of a logout response data packet. In the absence of response, the pointer is null.

Return value:
0: Logout is OK.
3000: wrong parameter
3006: unable to log out when not logged in
3018: data stream flag error

Tips:
1) The parameter conn must be the handle of an established connection to the exchange.
2) A user must correctly complete such logout information as the flag of a stream in the data packet pointed by reqPkg. For details, please refer to the *ZCEAPI Reference Manual*.
3) If the value of the parameter wait is not large enough, the function will return API_FALSE for timeout. In this case, the return value cannot accurately reflect whether logout is successful or not.
4) The MsgPackage pointed by rspPkg is read-only.
5) This is a synchronous function, so please do not call it from the callback functions of the API.
6) If a link is disconnected, the login status becomes void immediately. This is equivalent to automatic logout.

4.6.8 Send a Data Packet

```
int API_Send(ExchgConnectionHandle conn,MsgPackageHandle pkg);
```

Function: To send a ZCEAPI data packet

Parameter specs:
1) ExchgConnectionHandle conn Handle of an exchange connection
2) MsgPackageHandle reqPkg Handle of the data packet to be sent

Return value:
0: Sent successfully
3000: Wrong parameter
3009: Unable to send for not logged in
3010: Data packet format error
3015: Unable to identify the data packet, maybe the API version is low
3016: Error in sending login data
3018: Error in flagging data stream
3024: Data packet PID error
3025: Null data packet
3028: Too frequent query

Tips: 1) The parameter conn must be the handle of an established connection to the exchange.

2) A user must correctly complete all necessary information in the data packet pointed by pkg. For details, please refer to the *ZCEAPI Reference Manual*.

3) This function cannot send login and logout data packets.

4) This function is non-thread safe, so please do not use the same connection to send data through multiple threads at the same time.

5) The API will limit the number of queries to be initiated at the same time. If the limit is exceeded, the request will fail, and the error code "3028 (too frequent query)" will be returned.

4.6.9 Close a Connection to the Exchange

```
void API_DisConnect(ExchgConnectionHandle conn);
```

Function:	To break the connection to the exchange	
Parameter specs:	ExchgConnectionHandle conn	handle of an exchange connection
Tip:	When the execution of this function is over, a connection is not yet freed, but the link to the exchange is closed.	

4.6.10 Close and Free an Exchange Connection

```
void API_FreeExchgConnection(ExchgConnectionHandle conn);
```

Function:	To close and free a connection to the exchange	
Parameter specs:	ExchgConnectionHandle conn	handle of an exchange connection
Tip:	When the execution of this function is over, a connection to the exchange is closed first and then freed. Once this function is executed, the handle conn of the connection cannot be used again.	

4.6.11 Get the Status of a Data Stream

```
enum API_DFSTATUS API_GetDataFlowStatus(ExchgConnectionHandle conn ,int DataFlowFlag);
```

Function:	To get the status of a data stream accessed through a designated connection to the exchange	
Parameter specs:	1) ExchgConnectionHandle conn	Handle of an exchange connection
	2) int DataFlowFlag	Flag of a data stream
Return value:	If successful, return the status of a data stream identified by a DataFlowFlag and accessed through the exchange connection conn. Please refer to the "status of data streams" in this handbook.	

4.7 Set a Callback Function

4.7.1 Set a Notice for Disconnection

```
void API_SetCloseCallBack(ExchgConnectionHandle conn, ExchgConnectionCallBack callback,void * CallBackArg);
```

Function: To set the callback function to be invoked when a designated exchange connection is closed

Parameter specs:

1) ExchgConnectionHandle conn	Handle of an exchange connection
2) ExchgConnectionCallBack callback	Callback function to be set for the status of a link
3) void * CallBackArg	Parameter of the callback function to be set for the status of a link, ultimately fed to the callback function. Please refer to the "**Callback Functions for the Status of a Link**" in Part 3 of this handbook.

4.7.2 Set a Notice for Errors

```
void API_SetErrorCallBack(ExchgConnectionHandle conn, ExchgConnectionCallBack callback,void * CallBackArg);
```

Function: To set the callback function to be invoked when a designated exchange connection encounters an error

Parameter specs:

1) ExchgConnectionHandle conn	Handle of an exchange connection
2) ExchgConnectionCallBack callback	Callback function to be set for the status of a link
3) void * CallBackArg	Parameter of the callback function to be set for the status of a link, ultimately fed to the callback function. Please refer to the "**Callback Functions for the Status of a Link**" in Part 3 of this handbook.

Tip: For a link that receives broadcast streams through the UDP protocol, there is the possibility that two threads simultaneously invoke a callback function, so there is the need to synchronize the two threads.

4.7.3 Set a Notice for Data Receipt

```
void API_SetRecvCallBack(ExchgConnectionHandle conn, ExchgPackageCallBack callback,void * CallBackArg);
```

Function: To set the callback function for data processing

Parameter specs:

1) ExchgConnectionHandle conn	Handle of an exchange connection
2) ExchgPackageCallBack callback	Callback function to be set for the return of data packets
3) void * CallBackArg	Parameter of the callback function to be set for the return of data packets, ultimately fed to the callback function. Please refer to the "**Callback Functions for the Return of a Data Packet**" in Part 3 of this handbook.

Tips: 1) The login/logout response data packets are to be returned to a user through the output of the login/logout function.

2) A user cannot call the API_FreePackage () function to release the data packet called back.

3) For a link that receives broadcast streams through the UDP protocol, there is the possibility that two threads simultaneously invoke a callback function set, so there is the need for the callback function to synchronize the two threads.