

Notice To Reader: The content of this document is only to illustrate the design and implementation of version 1.486. YD reserves the right to change the design and implementation in subsequent versions.

1 Quickstart

This chapter will demonstrate the necessary steps for investors to access and use the YD order management system (YD OMS) through a simple strategy program, and help investors establish a basic understanding of YD API.

1.1 Environment preparation

Initially, download the corresponding version of the API from <https://www.hanlinit.com/download>. Before downloading, please register an investor account on the official website and wait for approval.(YD promises that registered information will only be used for disseminating YD-related information and not for other purposes). Unzip the ydClient_1_486_XX_XX.tgz file after downloading, and use the tools and programs in the ydClient/win64 and ydAPI directories for the demonstration.Then, through using the Windows GUI client ydClient of YD, you can directly connect to the YD OMS and easily check the running results of the subsequent programs.

First, enter the ydClient/win64 directory and modify the YDConfig.ini file to the following content:

```

1  #####
2  ##### Network configurations #####
3  #####
4
5  # IP address of yd trading server
6  # TCP port of yd trading server, other ports will be delivered after logged in
7
8  TradingServerIP=118.190.175.212
9  TradingServerPort=41600
10
11 #####
12 ##### Trading configurations #####
13 #####
14
15 # Choose trading protocol, optional values are TCP, UDP, XTCP
16 TradingProtocol=TCP
17
18 # Timeout of select() when receiving order/trade notifications, in millisec. -1 indicates running without
19 # select()
20 TradingServerTimeout=10
21
22 # Affinity CPU ID for thread to send TCP trading and receive order/trade notifications, -1 indicates no
23 # need to set CPU affinity
24 TCPTradingCPUID=-1
25
26 # Affinity CPU ID for thread to send XTCP trading, -1 indicates no need to set CPU affinity
27 XCTPTradingCPUID=-1
28
29 #####
30 ##### MarketData configurations #####
31 #####
32
33 # Whether need to connect to TCP market data server
34 ConnectTCPMarketData=yes
35
36 # Timeout of select() when receiving TCP market data, in millisec. -1 indicates running without select()
37 TCPMarketDataTimeout=10
38
39 # Affinity CPU ID for thread to receive TCP market data, -1 indicate no need to set CPU affinity
40 TCPMarketDataCPUID=-1
41
42 # Whether need to receive UDP multicast market data
43 ReceiveUDPMarketData=no
44
45 #####
46 ##### Misc configurations #####
47 #####
48
49 AppID=yd_dev_1.0
50 AuthCode=ecbf8f06469eba63956b79705d95a603

```

The above configuration information points to an Internet test environment provided by YD for SHFE and INE. This environment operates around the clock, replaying market data from a specific trading day to facilitate investor debugging. For more details, please refer to <https://www.hanlinit.com/docs/dev-environments/>.

After completing the configuration, double-click YDClient.exe to run it, and use any one of the accounts 001, 002-099, 100(password same as username) to log in. Once logged in successfully, you can browse the content in various menus.

After completing the steps above, we will now focus on preparing the strategy program.

1.2 Strategy program

To compile an executable program, you need to prepare the YD sample program first. Please copy all the files from ydAPI/linux64, ydAPI/include, and ydAPI/example directories to the working folder of the Linux server, and use the following commands to compile:

```
1 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c ydExample.cpp -o ydExample.o
2 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example1.cpp -o Example1.o
3 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example2.cpp -o Example2.o
4 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example3.cpp -o Example3.o
5 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example4.cpp -o Example4.o
6 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example5.cpp -o Example5.o
7 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example6.cpp -o Example6.o
8 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example7.cpp -o Example7.o
9 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example8.cpp -o Example8.o
10 g++ -fpic -g -std=c++11 -c -O3 -pthread -Wall -c Example9.cpp -o Example9.o
11 g++ -g -std=c++11 -o ydExample -I. ./ydExample.o ./Example1.o ./Example2.o ./Example3.o ./Example4.o
    ./Example5.o ./Example6.o ./Example7.o ./Example8.o ./Example9.o -m64 -Wall -pthread -lrt -ldl -Wl,-
    rpath,. -L. -l:yd.so
```

After a successful compilation, copy the content from YDConfig.ini, as mentioned above, to the config.txt file. Then you can run the first sample program. Replace and with the credentials you use to log into the ydClient:

```
1 # ./ydExample <example name> <config file> <username> <password> <instrumentID>
2 ./ydExample Example1 config.txt <username> <password> cu2306
```

When the following information appears, it means that the strategy program has started to execute, and please refer to the content about dmidecode in the [Information Collection](#) for the warning messages:

```
1 sudo: a password is required
2 login successfully
3 Position=0
4 sell open 1 at 71580
```

Please observe the changes in orders, trades, positions, and funds in the account detail interface of ydClient after trades occur.

So far, the first YD sample program has been compiled and run successfully. Investors can continue to try other sample programs and read these programs, which helps investors to quickly grasp the basic knowledge of the API. The following is a brief excerpt of the instructions for each example in ydExample:

```
1 All sample programs use the following strategy for a single product:
2   If the buying volume in the market is 100 more than the selling volume, buy one lot at the counter
   price.
3   If the selling volume in the market is 100 more than the buying volume, sell one lot at the counter
   price.
4   The risk control restriction is that the position volume is not allowed to exceed 3.
5
6 The following sample programs demonstrate different levels of precision in position management:
7 Example1: Manages only executed positions, not order positions, and allow at most one pending order at a
   time.
8 Example2: Manages order positions based on order feedback for more precise position management.
9 Example3: Based on the previous example, the management of order positions based on self-issued orders
   has been added to achieve more accurate position management. In addition, this example also demonstrates
   how to use notifyCaughtUp to obtain information about catching up to the latest transaction records.
10 Example4: Using YDExtendedApi to achieve the same functionality can greatly simplify the program.
11 Example9: demonstrates how to use OrderGroup to implement reliable UDP order placement and cancel locally
   generated orders.
12
13 The following sample programs are used to demonstrate other functions:
14 Example5: This program demonstrates how to send instructions for executing or abandoning options, as well
   as inquiring about them.
15 Example6: Demonstrates the utilization of the YDExtendedApi in a market maker's quoting system.
16 Example7: Provides a demonstration on how to utilize YDExtendedApi for uncomplicated risk monitoring and
   alerting.
17 Example8: Demonstrates how to use YDExtendedApi to automatically establish traditional combined positions
```

By studying the sample programs, you should have mastered the basic steps of writing YD strategy programs. Next, it is recommended that investors read the remaining contents of the documentation systematically. The documentation not only provides a detailed explanation of YD's basic principles and usage methods but also offers tips and best practices for writing YD API programs.

If you encounter any problems during the reading process, feel free to contact the YD support team by email (support@hanlinit.com) or through your broker. The YD support team is happy to answer any questions you may have.

2 Basic concept

2.1 API selection

YD provides five different types of trading interfaces, including ydApi, ydExtendedApi, raw protocol, ydCTP, and Python interface, to meet various requirements of investors.

Raw protocol is the most flexible way to place orders. YD has publicly announced the uplink protocol for placing and canceling orders, as well as the downlink protocol for order notification, trade notification, error notification, etc. Investors can construct UDP packets for placing or canceling orders, or monitoring and analyzing downlink acknowledgment messages on their own. Raw protocol needs to be used in conjunction with the ydApi or ydExtendedApi. It is suitable for investors who are accustomed to using raw protocols to access OMSs. For details, refer to [Raw Protocol](#).

ydCTP is a CTP-like API that simulates core functions of the CTP such as order placement and query, allowing programs that were developed based on CTP to connect to the YD trading system without modification. This makes it easier for investors who have not yet officially used the YD OMS to evaluate and test its performance and stability. However, due to the incompatible architecture between YD and CTP, ydCTP cannot cover all CTP interfaces, nor can it achieve behavior completely identical to native interfaces. Therefore, it is not recommended to use ydCTP for trading in a production environment. For details, please refer to the relevant documentation.

Please refer to the relevant documents in the ydClient package for more detailed information about YD python version API.

2.1.1 YdApi

ydApi is a high-performance API designed with the concept of simplicity and high efficiency. Its main responsibility is to send the investor's orders to the OMSs and notify the investor through the callback function after receiving the notifications. Generally, except for [Static Data](#), the API does not retain any [Dynamic Data](#) like orders and trades. It also does not assist investors in calculating funds and positions, thus occupying minimal memory. Investors who use ydApi need to manage funds and positions based on their preday positions and the subsequent notifications such as orders and trades, and provide corresponding methods for queries. It is suitable for investors who are highly concerned about performance and can manage their own funds and positions.

If the [High Availability](#) feature is enabled, ydApi will occupy roughly equivalent memory as the amount of notification data for high availability recovery. Based on estimates, with 5 million orders without trades in a Linux 64-bit system, ydApi will occupy around 400M of memory. In actual production, due to the presence of trades, rejected orders, combinations, etc., it will occupy more memory in the same order volume situation.

ydApi possesses the following features:

- Thread safety: Any API method can be called at any time under any thread.
- Reentrancy: Any API method can be called in any callback function.
- Pointer stability: The pointers to static data obtained through the "get" or "find" methods in ydApi will not change throughout the entire life cycle. The data pointed to by the pointer can be read at any time through the strategy program; However, the pointers of order notifications and trade notifications obtained through the "notify" callback interface are different, and the pointers of multiple notifications with the same "OrderSysID" are also different. The exception is that the pointers of static data sent back through "notifyCombPosition" method are stable.

YD currently provides three versions of the API: Linux 64-bit, Win 32-bit, and Win 64-bit. The Linux 64-bit version of the API has been optimized extensively for the Linux platform and has significantly better performance than the two versions for Windows. It is a highly recommended version for production use. For investors who have to use Windows platforms, since the Win 32-bit program is only available for a 2G of user memory, it is suggested that investors use the 64-bit version to prevent the API from being frozen due to out-of-limit of memory.

2.1.2 YdExtendedApi

ydExtendedApi is a versatile API designed with the ideals of comprehensive functionality and ease of use. Based on inheriting all the functions and features of ydApi (ydExtendedApi is a subclass of ydApi), the API internally stores all the [Dynamic Data](#) such as orders and trades, etc. Based on these dynamic data, it helps investors manage funds and positions, and provides a variety of unique data structures and query methods at the cost of a very low downlink performance overhead and certain space occupation. The larger the transaction volume, the larger the space occupied. It is suitable for investors who are accustomed to using a full-featured API.

Since maintaining an extended data structure locally, ydExtendedApi occupies more space than ydApi. It is estimated that ydExtendedApi will occupy around 1400M of memory when the high availability feature is disabled in a Linux 64-bit system with 5 million unfilled orders. When the high availability feature is enabled, ydExtendedApi will occupy approximately 1800M of memory. In actual production, considering trades, failed orders and combinations, etc., more space can be occupied under the same order volume.

The query methods of ydExtendedApi are all executed locally rather than at the OMSs. However, considering that all queries are locked, it costs a lot. The strategy program can save the pointers returned from the API and directly use the data pointed by the pointer in the subsequent execution process. In addition to inheriting the pointer stability feature of ydApi, any pointers obtained through the "get" or "find" methods of ydExtendedApi, as well as the pointers to extended dynamic data sent back by YDExtendedListener callbacks, are also stable.

2.2 Order modes

YD supports three types of order modes: TCP, UDP, and XTCP, which are different in terms of penetration performance and reliability. Investors can choose the order mode according to their own needs.

2.2.1 TCP order

The overall penetration delay of TCP order transmission mode is relatively high. The time consumption of an API order transmission mode itself (the time from calling the API order interface to the first byte of the order being sent to the fiber) is about 1-2 μ s. The penetration to the OMS is around 8-10 μ s, but it can ensure that the order is delivered to the OMS. When investors test or connect to the OMS through the Internet, it is recommended to use the TCP order transmission mode to avoid the problem of order loss caused by the firewall and other reasons.

In the same thread of TCP order transmission, it is also responsible for sending non-trading information (such as password modification) and receiving notification information sent by the OMS. Therefore, no matter what kind of order transmission mode is used, a thread for the TCP order transmission will definitely be created.

To use TCP order transmission, please set TradingProtocol=TCP in the API configuration file. Please refer to [Configuration File](#) for details.

2.2.2 UDP order

UDP order transmission mode has a low overall penetration delay. The time consumption for an API order itself is about 200 ns, while the OMS penetration is about 2-3 μ s (depending on the different exchanges). The reference penetration value provided by YD Official is measured under the UDP order transmission mode. It is recommended that investors use UDP order transmission mode in the production environment. UDP order transmission mode is only used for submitting uplink orders, and notifications are still sent through the original TCP channel.

For some investors who are concerned about the possibility of UDP order loss, firstly, the network conditions where the YD OMSs are located are usually stable, fast and idle. Unless there is a failure in modules, fibers, etc., loss of UDP order rarely happens; Secondly, YD provides the OMS notification functions. When investors receive an OMS notification, it means that the OMS has received the order. Even if it is lost, investors can still be aware of it. Please refer to the description of the OMS notifications in [Order Notification](#) for details.

To use UDP order transmission mode, set TradingProtocol=UDP in the API configuration file. Please refer to the [Configuration File](#) for details. If the OMS has not enabled UDP service, "false" will be returned every time the order interface is called. UDP order transmission mode does not have a dedicated thread for order submission. It will directly send orders in the thread that calls the order submission and cancellation interface. Therefore, please bind the calling thread to an isolated core to ensure the order submission performance.

2.2.3 XTCP order

The overall penetration delay of XTCP order transmission mode is relatively low, and the time consumption of API order transmission itself is about 250 ns. The penetration delay of the OMS compared to UDP order transmission mode is increased by less than 100 ns. Its operation mode is the same as UDP, which is only available for submitting uplink orders, and the notification is still returned through the original TCP channel.

Compared with UDP order transmission mode, the problem of order loss for XTCP can be completely eliminated and the prolongation of penetration delay is relatively short. It is suitable for investors who do not trust the UDP protocol. However, since the maintenance cost of TCP connection protocol stack is higher than that of UDP, investors should be cautious about using XTCP connection. If an XTCP thread is used widely, it will impose significant pressure on the strategy host and the OMS.

To use XTCP order transmission mode, set TradingProtocol=XTCP in the API configuration file. Refer to the [Configuration File](#) for more details. If the XTCP service is disabled at the OMS, it will fall back to using TCP order transmission mode. Since the XTCP service is closed by default, please confirm with your broker whether they have enabled the XTCP service before using it. In most cases, XTCP will directly place orders in the thread that calls the order submission and cancellation interface. Therefore, please bind the calling thread to an isolated core to ensure order placement performance. TCP resend messages and TCP heartbeat messages are sent through a dedicated thread of XTCP. You can bind it by using the "XTCPTradingCPUID" parameter.

Starting from version 1.486, XTCP supports the network bonding of master-slave mode. As long as the master-slave mode bonding network cards are configured on the operating system, and the network cards involved in the bonding are high-performance network cards supported by YD, such as Solarflare X2522, Exanix X10, the API can automatically identify and support high-availability switching of the bonded network cards.

2.3 API thread

After the initiation of the YD API, three threads will be created: TCP notification receiving, TCP market data receiving and a timer. Orders are sent directly by calling the thread of "insertOrder", and therefore no dedicated thread is provided for handling order submission.

2.3.1 TCP notification receiving thread

The main task for the TCP notification receiving thread is to receive notifications and invoke YDListener callbacks, as well as heartbeats to OMSs. Most callbacks from YDListener or YDExtendedListener are initiated from this thread, except for "notifyMarketData". If the thread stays in a callback function for a long time (60 s) without exiting, it may cause a heartbeat timeout and lead to disconnection of the TCP connection. Therefore, it is recommended for investors to keep the processing time of each callback function as short as possible. You can set "TCPTradingCPUID" in the API configuration file to specify the CPU core to bind to.

There are two network listening modes for the TCP notification receiving thread: Busy polling and Select. If the busy polling mode is selected, the notification receiving speed will be quicker, however, the CPU utilization rate under this mode will reach 100%. If the select mode is used, the thread will wait for the select timeout to continue with the next select, resulting in almost no CPU overhead. However, the notification receiving speed is slower compared to busy polling. The timeout value for the TradingServerTimeout can be set in the API configuration file. Please refer to the [Trading Configuration](#) for details.

In order to ensure fairness in all connections receiving trading flows through OMSs, the OMSs can switch to the next connection when pushing 20 trading flows each time to a particular connection. This prevents any account from having an impact on the notification received by other existing connections when collecting all trading flows while logged in.

2.3.2 TCP market data receiving thread

The main task for the TCP market data receiving thread is to receive market data pushed by the gateway of the exchange, call back YDListener and send heartbeat messages to OMSs. The callback of notifyMarketData in YDListener is initiated from this thread, which is similar to the TCP notification thread. The processing of notifyMarketData also needs to be fast, otherwise it may cause the TCP connection of this thread to be disconnected. The CPU core to be bound can be specified by setting "TCPMarketDataCPUID" in the API configuration file.

There are two network listening modes for the TCP market data receiving thread: Busy polling and Select. If the busy polling mode is selected, the market data receiving speed will be quicker, but the CPU utilization rate will reach 100%. If the Select mode is selected, the select function will wait for a specified time before timing out and continuing to the next select, which has almost no CPU overhead. However, the speed of receiving market data is slightly slower compared to the busy polling mode, with a delay of a few microseconds. The timeout for "TCPMarketDataTimeout" can be set in the API configuration file. Please refer to the [Market Data Configuration](#) for more details.

2.3.3 Timer thread

The timer thread is used for uniformly executing all timed tasks in the API. This thread wakes up regularly every other 1 millisecond. Each time the timer triggers, it asks each timed task whether to execute based on their own characteristics. At present, the timer thread is only used for notifying investors of the appropriate refresh time through [Auto Mode](#) or [Fund Refresh Mechanism](#).

The timer thread cannot be turned off and can be bound to a specific CPU by setting the TimerCPUID in the API configuration file.

2.3.4 User-defined field

YD Api supports two types of user-defined fields: local and remote. Local user-defined fields are only saved in the local memory of the API and will not be sent to the YD OMS. Remote user-defined fields will be sent to the counter through orders and quotes, and will be brought back in order notifications, quote notifications and trade notifications.

2.3.5 Local User-defined field

To facilitate investors in storing user data in the structures of the YD, based on the stable characteristics of YD's pointers, YD provides four fields: pUser, UserFloat, UserInt1 and UserInt2, in YDExchange, YDProduct, YDInstrument, YDMarketData, YDCombPositionDef, YDAccount, YDAccountExchangeInfo, YDAccountProductInfo and YDAccountInstrumentInfo.

The API will not modify the data set in these fields, including the following special scenarios **forever**:

- When calling the function of "startDestroy()" to destroy the API, the space stored in pUser will not be released automatically.
- When a main-standby switchover occurs, as the above structure does not change, the data stored in the user-defined field will not be affected.

If the above fields are not sufficient to store user data, a structure can be defined and the pointer to that structure can be stored in pUser. Essentially, the API provides a continuous user-defined space of 24 bytes, where any value can be stored, breaking the API's pre-defined field boundaries for cross-field data storage. Investors can modify the field names, types, and numbers in the header file to meet actual needs, but ensure that the total length does not exceed 24 bytes.

2.3.6 Remote User-defined field

Starting from version 1.486, a 32-byte UserRef field is added to orders and quotes. The UserRef will be returned through order notifications, quotation notifications and trade notifications. The API and the YD server will not use or modify the value of UserRef.

Although OrderRef and UserRef both remote user-defined fields that investors can fill in and bring back from various notifications, the difference is that OrderRef stores sequential data and will participate in the order and quotation reference number increment check. Therefore, it is not suitable to store categorical information in OrderRef through encoding, etc. UserRef is suitable for storing categorized and identifying information, such as strategy number, server number, etc. The 32-byte field length provides sufficient space for segmented storage of multiple information.

2.4 Version rule

YD adopts a four-segment version number encoding rule with the format of a.b.c.d. Each release version will only modify one segment of the version number. YD makes the following commitments:

- a: Protocol Version Number - Changing the version number in this section will inevitably modify the protocol and render all previously released API versions incompatible. It may also include modifications to sections b and c. Only when the segment version number of the API is the same as that of OMSs, and the API version number a.b.c is lower than the OMSs' version number a.b.c, can the API function work properly on that version of OMSs. A higher version number in this section indicates a higher version, and if they are the same, the comparison continues with section b.
- b: Function version number - Adding this segment version number certainly means adding new functions, which also include the modified content of Segment c. A higher version number in this section indicates a higher version, and if they are the same, the comparison continues with section c.
- c: Patch version number - Increasing the version number in this segment can only contain bug-fixing patches and must be for an unreleased version. A larger version number in this segment indicates a higher version. If they are the same, continue comparing the d segment.
- d: Emergency patch version number - A version with this segment version number added, **in principle**, means only the patch content for fixing an emergency Bug can be included, which must refer to an officially released production version. However, in order to cope with rapidly changing business needs, YD may have to add new functions to the emergency patch while ensuring the version compatibility. A larger version number in this segment indicates a higher version.

YD promises that all investors use the same version of YD, and no so-called special versions will be provided. However, considering the upgrade strategy and specifications of YD, it should be understood that during the trial operation and upgrade process, some investors may use a higher version of the OMS that have not been officially released to the whole market. Wish investors understand:

- After completing the internal testing phase of a version, YD will invite selected brokers who are willing and capable of taking risks, and possess strong operational and emergency handling capabilities in both business and technology, to conduct trial runs of the new version. The duration of the trial run is not fixed. The more content released in this version, the longer the trial run will usually last. Moreover, if new issues are fixed or new features are introduced during the trial run stage, the duration of the trial run will be further extended.
- After the trial operation, YD will release the official version to the entire market. At this time, all brokers can gradually upgrade to the latest version in batches. To ensure that YD has sufficient service manpower to handle various problems that appeared during the upgrade process, the total number of upgrades per week will be limited during the first 1-2 weeks. Considering the current market share of the YD OMSs and the relatively conservative upgrading strategies adopted by some brokers, it will take 2-3 months for the entire market to complete the upgrade.

2.4.1 API version compatibility

Only when the segment version number of the API is the same as that of OMSs, and the segment version numbers a, b and c of the API are set between MinApiVersion and MaxApiVersion can the API connect and log in to OMSs normally, otherwise a log-in error message "YD_ERROR_TooHighApiVersion=62" or "YD_ERROR_TooLowApiVersion=58" will be received.

- If brokers do not make special settings, the default MaxApiVersion of OMSs is equal to the version of the OMSs. Under special circumstances, the highest API version supported by OMSs can be modified by configuring the "MaxApiVersion" parameter of OMS. The MaxApiVersion can only be set using the a.b.c format for version numbers. Comparing only the a.b.c segments of the version number allow urgent patches (modifying the fourth segment of the version number) for the API to be compatible with older versions of the OMS.
- If brokers do not make special settings, the default MinApiVersion of OMSs will be 1.0.0.

To obtain the current version, MaxApiVersion, and MinApiVersion of OMSs, please refer to [System Parameters](#).

The version of API can be obtained by the following two different methods, or you can check the API version in the log file. For more details, please refer to [Logging](#).

```

1  class YDApi
2  {
3      virtual const char *getVersion(void);
4  }
5
6  /// Same as getVersion inside YDApi, put here to get version without make api
7  YD_API_EXPORT const char *getYDVersion(void);

```

3 Life cycle

3.1 Creation

To create an API instance, you first need to determine whether to use `ydApi` or `ydExtendedApi`. The differences between these two APIs can be found in the [API Selection](#). For more detailed distinctions, please refer to the relevant content in this document.

The process of creating an API is not allowed to call any system calls that create child processes, such as `fork()`, `system()`, `exec()`, `vfork()`, `clone()`, `posix_spawn()`, and so on, in order to avoid creating unexpected problems.

3.1.1 Create ydApi

There are two ways to create `ydApi`. The argument to `makeYDApi` needs to be filled in with the path to the configuration file, and the API will read the configuration content from the file pointed to by the path; the argument to `makeYDApiFromConfig` is directly filled in with the content of the configuration file itself, which should be in the same format as the configuration file pointed to by the previous interface, i.e., the parameter name and parameter value consisting of a configuration pairs, separated by `\n`.

```
1 | YDApi *makeYDApi(const char *configFilename)
2 | YDApi *makeYDApiFromConfig(const char *configDesc)
```

If `ydApi` is selected, the creation process is approximately as follows:

```
1 | // Create YDApi
2 | YDApi *pApi=makeYDApi(configFilename);
3 | if (pApi==NULL)
4 | {
5 |     printf("can not create API\n");
6 |     exit(1);
7 | }
8 |
9 | // Create listener of Api
10 | YDExampleListener *pListener=new YDExampleListener(...);
11 | /// Start Api
12 | if (!pApi->start(pListener))
13 | {
14 |     printf("can not start API\n");
15 |     exit(1);
16 | }
```

The `makeYDApi` function requires a configuration file as input. Please refer to the [Configuration File](#) for instructions and an example of how to configure it.

The `YDExampleListener` mentioned above is a subclass of the `YDListener`, which is implemented by the investors themselves. All notification information is sent to the strategy program through the callback function of this subclass. Since the instance of `YDListener` is created by the strategy program, its life cycle should be maintained by the strategy program, which can be destroyed when necessary. The callback function of `YDListener` will be introduced in each functional section of this document.

The API can be started by calling `ydApi.start` and the parameter "pListener" cannot be empty. The notification will be made immediately without blocking after the function call. To prevent the program from exiting directly, the strategy program needs to add blocking code after a successful call.

```
1 | virtual bool start(YDListener *pListener);
```

3.1.2 Create ydExtendedApi

There are two ways to create `ydExtendedApi`. The argument to `makeExtendedYDApi` needs to be filled in with the path to the configuration file, and the API will read the configuration content from the file pointed to by the path; the argument to `makeExtendedYDApiFromConfig` is directly filled in with the content of the configuration file itself, which should be in the same format as the configuration file pointed to by the previous interface, i.e., the parameter name and parameter value consisting of a configuration pairs, separated by `\n`.

If `ydExtendedApi` is selected, the creation process is roughly as follows:

```
1 | // Create YDApi
2 | YDExtendedApi *pApi=makeYDExtendedApi(configFilename);
3 | if (pApi==NULL)
4 | {
5 |     printf("can not create API\n");
6 |     exit(1);
7 | }
8 | // Create listener of Api
9 | YDExampleListener *pListener=new YDExampleListener(...);
10 | /// Start Api
11 | if (!pApi->start(pListener))
12 | {
13 |     printf("can not start API\n");
14 |     exit(1);
15 | }
```

It can be seen that except for calling `makeYDExtendedApi` to create an API instance, the other steps are the same as those for `ydApi`.

In order to facilitate users of the ydExtendedApi in receiving notifications about changes in extended messages, "startExtended" can be used to receive callback notifications from YDListener and YDExtendedListener at the same time when starting API.

```

1 // YDExample7Listener implements both YDListener and YDExtendedListener interfaces.
2 class YDExample7Listener: public YDListener, public YDExtendedListener {}
3
4 // Create YDApi
5 YDExtendedApi *pApi=makeYDExtendedApi(configFilename);
6 if (pApi==NULL)
7 {
8     printf("can not create API\n");
9     exit(1);
10 }
11 // Create listener of Api
12 YDExample7Listener *pListener=new YDExample7Listener(...);
13 /// Start Api, YDExtendedListener is used here
14 if (!pApi->startExtended(pListener,pListener))
15 {
16     printf("can not start API\n");
17     exit(1);
18 }

```

The function signature of "startExtended" is as follows: both parameters "pListener" and "pExtendedListener" shall not be empty. Except for the addition of the callback notification of "YDExtendedListener", this function is identical to "start":

```

1 virtual bool startExtended(YDListener *pListener,YDExtendedListener *pExtendedListener);

```

The definition of "YDExtendedListener" is as follows. Compared to the structure sent back by "YDListener", the extended structure sent back by YDExtendedListener contains more information, which can facilitate investors in writing code. Please refer to ydDataStruct.h for the specific content of each extended structure.

```

1 class YDExtendedListener
2 {
3 public:
4     virtual ~YDExtendedListener(void)
5     {
6     }
7
8     // all address of parameters in following methods are fixed
9     virtual void notifyExtendedOrder(const YDExtendedOrder *pOrder)
10    {
11    }
12    virtual void notifyExtendedTrade(const YDExtendedTrade *pTrade)
13    {
14    }
15    virtual void notifyExtendedQuote(const YDExtendedQuote *pQuote)
16    {
17    }
18    virtual void notifyExtendedPosition(const YDExtendedPosition *pPosition)
19    {
20    }
21    virtual void notifyExtendedAccount(const YDExtendedAccount *pAccount)
22    {
23    }
24    // notifyExchangeCombPositionDetail and notifyExtendedSpotPosition will only be used when trading
25    // SSE/SZSE
26    virtual void notifyExchangeCombPositionDetail(const YDExtendedCombPositionDetail
27    *pCombPositionDetail)
28    {
29    }
30    virtual void notifyExtendedSpotPosition(const YDExtendedSpotPosition *pSpotPosition)
31    {
32    }
33 };

```

3.1.3 Configuration file

When calling the makeYDApi method, the configuration file used by the client should be specified. The configuration file contains parameters mainly classified into three categories:

- Network configuration, including the IP address and port of ydServer. Please always fill in the port blank with the TCP port information provided by brokers. Other ports, including the UDP order port and TCP market data port, will be automatically provided after logging in. The option to enable UDP order transmission mode is controlled by the UDPTrading parameter.
- Trading configuration, including whether to use UDP for order transmission mode and selecting the working mode of the thread receiving the notification.
- Market data configuration, including receiving the TCP or UDP market data from ydServer or not.

The following shows the best practice template of the client configuration file provided by ydApi for the production environment. The final effect of this template is that:

- UDP order transmission mode can be used;

- TCP notifications can be received under the busy query mode, quickening the acquisition of notifications and binding the receiving thread to CPU 3.
- The RecalcMode feature for fund recalculation has been enabled. The setting for "ConnectTCPMarketData=no" will be overwritten as "yes", meaning market data will be received.

```

1 #####
2 ##### Network configurations #####
3 #####
4
5 # Count of recovery site. Used to achieve high availability at the expense of a little performance of
6 # 0 for no recovery, 1 for recovery always use primary site, 2 for recovery use primary and secondary
7 # sites
8 RecoverySiteCount=0
9
10 # IP address of primary trading server
11 TradingServerIP=127.0.0.1
12
13 # TCP port of primary trading server, other ports will be delivered after logged in
14 TradingServerPort=51000
15
16 # IP address of secondary trading server.
17 # Valid only when RecoverySiteCount equals to 2.
18 TradingServerIP2=
19
20 # TCP port of secondary trading server, other ports will be delivered after logged in.
21 # Valid only when RecoverySiteCount equals to 2.
22 TradingServerPort2=
23
24 #####
25 ##### Trading configurations #####
26 #####
27
28 # Choose trading protocol, optional values are TCP, UDP, XTCP
29 TradingProtocol=UDP
30
31 # Affinity CPU ID for thread to receive order/trade notifications, -1 indicate no need to set CPU
32 # affinity
33 TCPTradingCPUID=-1
34
35 # Affinity CPU ID for thread to send XTCP trading, -1 indicate no need to set CPU affinity
36 XTCPTradingCPUID=-1
37
38 # Timeout of select() when receiving order/trade notifications, in millise. -1 indicates running without
39 # select()
40 TradingServerTimeout=-1
41
42 # Work mode for recalculation of margin and position profit. Valid when using ydExtendedApi.
43 # auto(default): subscribe market data and automatically recalculate in proper time.
44 # subscribeOnly: subscribe market data and recalMarginAndPositionProfit should be called explicitly
45 # off: never do recalculation
46 RecalcMode=auto
47
48 # Gap between recalculations, in milliseconds. Valid when RecalcMode is set to auto.
49 # It will be adjusted to 1000 if less than 1000
50 RecalcMarginPositionProfitGap=1000
51
52 # Delay of recalculation after market data arrives to avoid collision with input order, in milliseconds.
53 # Valid when RecalcMode is set to auto. Should be between 0 and 100.
54 RecalcFreeGap=100
55
56 #####
57 ##### MarketData configurations #####
58 #####
59
60 # Whether need to connect to TCP market data server
61 ConnectTCPMarketData=no
62
63 # Timeout of select() when receiving TCP market data, in millise. -1 indicates running without select()
64 TCPMarketDataTimeout=10
65
66 # Affinity CPU ID for thread to receive TCP market data, -1 indicate no need to set CPU affinity
67 TCPMarketDataCPUID=-1
68
69 # Whether need to receive UDP multicast market data
70 ReceiveUDPMarketData=no
71
72 #####
73 ##### Other configurations #####
74 #####
75
76 AppID=yd_dev_1.0
77 AuthCode=ecbf8f06469eba63956b79705d95a603

```

3.1.3.1 Network configuration

The network configuration includes the IP address and port of the YD OMS. When trading in the production environment, please consult the technical department of the related broker for specific configuration details.

RecoverySiteCount: Count of high availability sites. When it is set to 0, the high availability feature for the API is not enabled. When set to 1, the high availability of the single OMS is enabled. When the OMS is restarted, the API will reconnect to the OMS and recover to the latest state as much as possible, preventing direct termination of the API. When set to 2, the primary-secondary dual OMS high availability feature is enabled. If the primary OMS fails, the backup OMS(specified in TradingServerIP2 and TradingServerPort2) will start, and the API will reconnect to the OMS and recover to the latest state. The use of high availability will result in little loss of notification performance.

TradingServerIP: IP address of the YD OMS.

TradingServerPort: The port of YD OMS. YD OMS usually opens three ports, which are TCP trading, TCP market data and UDP trading. The XTCP trading and UDP market data ports are closed by default. The TradingServerPort should be filled in with the TCP trading port. Other ports will be automatically provided to the client by the OMS after a successful login. Do not fill in the TradingServerPort with the UDP trading port, otherwise the YD OMS will not be connected. If orders are to be sent through UDP or XTCP, the parameters TradingProtocol=UDP or TradingProtocol=XTCP should be used for control.

UDPTradingServerPort: The port for UDP transmission mode. It is automatically provided by the OMS and is usually unnecessary to be filled in. However, when accessing the OMS from an external network through NAT, the port for the UDP transmission mode may change. In such cases, the assigned port will not be able to receive UDP order data correctly. Therefore, it is necessary to configure this parameter according to the actual port provided by the broker.

XTCPTradingServerPort: The port for XTCP transmission mode. It is automatically provided by the OMS and is usually unnecessary to be filled in. However, when accessing the OMS from an external network through NAT, the port for the XTCP transmission mode may change. In such cases, the assigned port will not be able to receive XTCP order data correctly. Therefore, it is necessary to configure this parameter according to the actual port provided by the broker.

TCPMarketDataServerPort: The port for TCP market data. It is automatically provided by the OMS and is usually unnecessary to be filled in. However, when accessing the OMS from an external network through NAT, the external port for the market data may change. In such cases, the assigned port will not be able to receive market data correctly. Therefore, it is necessary to configure this parameter according to the actual port provided by the broker.

TradingServerIP2: The IP address of YD backup OMS. This is effective only when the RecoverySiteCount is set to 2.

TradingServerPort2: The port number of YD backup OMS. The setting method is the same as TradingServerPort. It will be enabled when RecoverySiteCount is set to 2.

UDPTradingServerPort2: The UDP trading port of YD standby OMS. Its operation method is the same as that for UDPTradingServerPort. It will be enabled when RecoverySiteCount is set to 2.

XTCPTradingServerPort2: The XTCP trading port of YD standby OMS. Its operation method is the same as that for XTCPTradingServerPort. It will be enabled when RecoverySiteCount is set to 2.

TCPMarketDataServerPort2: The TCP market data port of YD standby OMS, the operation method is the same as that for TCPMarketDataServerPort. It can be enabled when RecoverySiteCount is set to 2.

3.1.3.2 Trading configuration

TradingProtocol: Order transmission mode. The optional order transmission modes are TCP, UDP and XTCP. Since the overall penetration performances of UDP and XTCP are much better than that of TCP, it is suggested to use UDP or XTCP order transmission mode in the production environment. In order to ensure compatibility, if TradingProtocol is not set, the original configuration UDPTrading will be used.

UDPTradingType The implementation options for the UDP transaction sender are automatic, sf, exa, and socket. If this parameter is not specified, the default value is automatic. "socket" represents using the standard socket protocol stack, "sf" represents using Solafire's ef_vi, "exa" represents using Exanic, and "automatic" represents automatically selecting an appropriate sender implementation, which can be one of sf, exa, or socket. Due to the complexity of the production environment, there are cases where the sender implementation of other applications and the YD API are incompatible. This can result in automatic degradation of the API to socket sending, significantly reducing the API's sending performance. Therefore, it is recommended that investors using UDP trading enforce the corresponding sender implementation method. In case of conflicts, there will be significant error prompts or the YD API application may fail to start, making the problem easier to detect and avoiding significant performance degradation for investors who are unaware of the situation.

TCPTradingCPUID: For setting the affinity of the TCP sending order and receiving notification threads. - 1 means that no affinity needs to be set. Otherwise, it is the number of the CPU/Core.

XTCPTradingCPUID: For setting the affinity of the XTCP order sending thread. - 1 means that no affinity needs to be set. Otherwise, it is the number of the CPU/Core.

TradingServerTimeout: This parameter specifies how the TCP trading and receiving threads of YDListener use the listening network communication. When set to a positive integer value, it means that the thread can be used to listen the arrival of network data through the POSIX Select Mechanism. The listening gap is a specified value in milliseconds. When set to -1, the receiving thread will read messages in a non-blocking manner, and its CPU utilization rate will reach 100%. If the client does not allocate CPU/Core properly based on its own machine configuration and set the affinity between the thread and CPU/Core, it will severely impact the performance and significantly increase the latency of the client program.

RecalcMode: Recalculation mode of margin and position profit/loss. The default value under this mode is "auto".

- auto: The margin and position profit/loss can be recalculated by API. API will automatically subscribe to the market data required for calculations. The refresh interval and delay time from the market can be controlled through the parameters "RecalcMarginPositionProfitGap" and "RecalcFreeGap". The TCP market data will be received forcefully.

- **subscribeOnly:** Only helps to automatically subscribe to the required market data, but investors need to call the "recalcMarginAndPositionProfit" method to recalculate. The TCP market data will be received forcefully.
- **off:** The margin and position profit/loss will not be recalculated. Investors should generally avoid using this mode.

RecalcMarginPositionProfitGap: The interval between two recalculations, measured in milliseconds. The default value is 1000ms. The minimum value is 1000ms, and if it is set below 1000ms, the system will automatically adjust it to 1000ms. This parameter is only effective when RecalcMode is set to "auto".

RecalcFreeGap: The time interval for delayed market data. In order to avoid clashes with the order submission time as much as possible, the recalculation should be made before or after the arrival of market data if possible. Investors can adjust the parameter based on different exchanges so that the market data can be recalculated during periods of low order activity. The unit is milliseconds, with a default value of 100 milliseconds. The delay must be kept within 0 - 100ms. When greater than 100ms, this value will be adjusted to 100ms. When less than 0, this value will be adjusted to 0. This parameter is only effective when RecalcMode is set to "auto".

OperWay: Specify the operation mode in spot trading. If the operation mode entered is not within the range of operation modes allowed for this investor, login will not be allowed (only works in the spot system). Most of the securities companies only support single character operation mode, individual securities companies need to use double characters or more operation mode, according to the requirements of the securities company to fill in.

3.1.3.3 Market data configuration

Please note that both TCP market data and UDP multicast market data from YD are not high-speed market data. At present, they can only be used by YD's server and client for calculating the margin and position profit/loss. If market data is needed for in production trading, please contact the related broker to receive the multicast market data.

ConnectTCPMarketData: Connect to the TCP market data service of ydServer or not. "Yes" means "Receiving allowed", and "No" means "Receiving not allowed".

TCPMarketDataTimeout: This parameter specifies how the TCP market data receiving thread of YDListener use the listening network communication. When set to a positive integer value, it means that the thread can be used to listen the arrival of network data through the POSIX Select Mechanism. The listening gap is a specified value in milliseconds. When set to -1, the receiving thread will read messages in a non-blocking manner, and its CPU utilization rate will reach 100%. If the client does not allocate CPU/Core properly based on its own machine configuration and set the affinity between the thread and CPU/Core, it will severely impact the performance and significantly increase latency of the client program.

TCPMarketDataCPUID: For setting the affinity of the TCP market data receiving thread. - 1 means that no affinity needs to be set. Otherwise, it is the number of the CPU/Core.

ReceiveUDPMarketData: Receive UDP multicast market data sent by ydServer or not. At present, the UDP multicast market data function of all YD OMSs is turned off. Please always keep this parameter as "No".

3.1.3.4 Other configurations

TimerCPUID: For setting CPUID of the timer thread for API.

AppID and AuthCode: When investors set AppID and AuthCode in the configuration file. When calling the function "login", they can leave the appID and authCode parameters as "NULL", and the API will then use the values configured in the configuration file.

LogDir: For specifying the log directory generated by ydApi ("log" by default). The directory will not be created automatically and needs to be manually created. If the corresponding directory does not exist, no log files will be generated.

3.1.3.5 User-defined configuration

YD supports reading configuration content from configuration files. Investors can not only read the configuration information of YD API, but also put the user-defined configuration in YD's configuration file, which can be then accessed through the YD API.

When investors set user-defined configuration items, they should name them in the format of "Application name. parameter name", for example, MyApp.Username. Though a parameter without an application name can still be entered directly and used, but the API will report a warning message stating that the parameter is not being used when it starts. To avoid confusion, it is not recommended to use parameters without an application name.

YD's user-defined configuration supports the listed parameters. Multiple parameters with the same name can be set to achieve this effect. The following shows the examples:

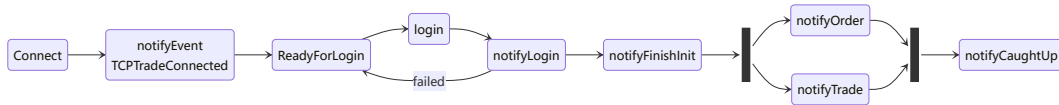
```
1 | MyApp.TradeExchange=SHFE
2 | MyApp.TradeExchange=INE
3 | MyApp.TradeExchange=CFFEX
4 | MyApp.TradeExchange=DCE
5 | MyApp.TradeExchange=CZCE
```

The YD API provides two methods for obtaining customized parameters. The first method obtains a single customized parameter, and when it is a listed parameter, it returns the first set value in that list. The second method can obtain all parameter values of a list parameter. The notified result set needs to be manually destroyed by the investor after being used up through calling "destroy()". When used for a single custom parameter, it will return a YDQueryResult collection containing a single element, which also needs to be destroyed using the "destroy()" function provided by the API to free up allocated memory.

```
1 | /// get first config using this name in config file (parameter of makeYDApi or makeYDExtendedApi), NULL if
   | not found
2 | virtual const char *getConfig(const char *name);
3 | /// get all configs using this name, user should call destroy method of return object to free memory
   | after using
4 | virtual YDQueryResult<char> *getConfigs(const char *name);
```

3.2 Start

After creating an API instance and calling "start" or "startExtend" to start the API, the API will complete the startup initialization process according to the following steps.



3.2.1 Connect

After the instance is started, it will continuously connect to the OMS address and TCP port specified by TradingServerIP and TradingServerPort respectively until the connection is successful. Once connected, After the successful connection, a callback message showing "Connected" will be sent through "notifyEvent"(YD_AE_TCPTradeConnected).

If the API is configured with dual-host high availability mode, it will cycle through the primary OMS address and TCP port specified by TradingServerIP and TradingServerPort, as well as the backup OMS address and TCP port specified by TradingServerIP2 and TradingServerPort2, until a successful connection is established. After the successful connection, a callback message showing "Connected" will be sent through "notifyEvent"(YD_AE_TCPTradeConnected).

3.2.2 Login

After the successful connection, the strategy program will be notified immediately through the callback function "notifyReadyForLogin" that it is ready to attempt a login. The parameter "hasLoginFailed" in this callback function refers to whether the initiation of the "notifyReadyForLogin" callback is caused by the last login failure. If yes, it will be unnecessary to log in again with exactly the same information since the failure is inevitable.

```
1 | virtual void notifyReadyForLogin(bool hasLoginFailed)
```

After receiving the "notifyReadyForLogin" callback notification, the strategy program should typically call the "login" function immediately to initiate the "login". The login function signature is shown below:

- If applID and authCode are set to NULL, the API will use the AppID and AuthCode specified in the configuration file. If these two parameters are not configured in the configuration file, an error will be caused;
- If applID and authCode are set to specific values, the API will only use the parameters specified in the interface and will not use the values specified in the configuration file.

```
1 | virtual bool login(const char *username, const char *password, const char *appID, const char *authCode);
```

username should be the account number, and password should be the password corresponding to the account number. For more information about password, please refer to [password](#). For more information about AppID and AuthCode, please refer to [AppID and authCode](#).

The following shows a code example of login. (In this example, AppID and AuthCode are defined in the configuration file, so those two parameters are filled with NULL):

```

1 | virtual void notifyReadyForLogin(bool hasLoginFailed)
2 | {
3 |     if (!m_pApi->login(m_username, m_password, NULL, NULL))
4 |     {
5 |         printf("can not login\n");
6 |         exit(1);
7 |     }
8 | }

```

The login result will be sent back through the callback of the function "notifyLogin" callback:

```
1 | virtual void notifyLogin(int errorNo, int maxOrderRef, bool isMonitor)
```

If the errorNo of notifyLogin is "0", it means that the "login" is successful. If "isMonitor" is true, it indicates that the currently logged-in user is the OMS administrator. If the logged-in user is an ordinary investor, it must be "false". maxOrderRef represents the maximum OrderRef received by the OrderGroupID 0 of the OMS at the time of login. For the maximum OrderRef of all order groups, please refer to [Order Group](#). Except for the following three cases, no matter whether an order or quote order is successful or not, as long as the OrderRef of the new order or quote order is greater than the current MaxOrderRef of the order group, the MaxOrderRef of the current account will be modified to be the OrderRef of the order:

- Failed orders and quotes caused by the [monotonic increase check of the order number](#) will not be included in MaxOrderRef;
- Failed orders and quotes with order account identification failure due to the error of the network data packet received by OMSs will not be counted in MaxOrderRef. This error should not occur in general and It often occurs in cases where investors try to break our company's protocol to place orders or quote orders.
- Failed orders caused by submitting the quote at an OMS that does not support the quote instruction (license file does not have market maker flag) will not be counted in MaxOrderRef.

After a successful login, the TCP market data port and UDP trading port issued by the OMS can be received by API. Therefore, it unnecessary for YD to specify the market data port, UDP trading port and XTCP trading port in the configuration file. However, in certain scenarios, such as accessing the OMS from the Internet, the mapped Internet port is usually different from the port issued by the OMS. If not processed, it will cause TCP market data connection failure or UDP trading being unable to send to the OMS. In this case, the TCP market data port or UDP trading port can be set manually to override the automatically issued ports. The "TCPMarketDataServerPort" can be used to override the market data port, "UDPTradingServerPort" to override the UDP trading port, and "XTCPTradingServerPort" to override the XTCP trading port. If the dual-host high availability mode is configured, the "TCPMarketDataServerPort2" can be used to override the market data port of the standby OMS, the "UDPTradingServerPort2" to override the UDP trading port of the standby OMS, and the "XTCPTradingServerPort2" to override the XTCP trading port of the standby OMS.

If the "errorNo" of "notifyLogin" is not 0, it means that the login fails. The errorNo will inform the specific reason for the failure and it is unnecessary to use maxOrderRef and isMonitor since they are meaningless at this time. The API will wait for 3 s and then continue to callback "notifyReadyForLogin" to wait for the user to send a login request. At this time, the "hasLoginFailed" of "notifyReadyForLogin" will be set to "true". The reasons for login failure may include:

Error No.	Error definition	Description
12	YD_ERROR_InvalidClientAPP	AppID or AppAuthCode error.
18	YD_ERROR_AlreadyLoggedIn	Current API login completed. Re-login is not allowed.
19	YD_ERROR_PasswordError	Wrong login password.
20	YD_ERROR_TooManyRequests	The number of login requests is beyond the limit.
21	YD_ERROR_InvalidUsername	Invalid login user.
27	YD_ERROR_InvalidAddress	The IP address of the client is not in the address range allowed for logging in. It is only valid for the administrator.
35	YD_ERROR_ClientReportError	Failure in collecting the look-through regulation information. Please refer to Look-through Regulation for more details.
50	YD_ERROR_TooManyLogines	The license file of OMS restricts the number of accounts that can log in at the same time. When the login limit is reached, the subsequent accounts will not be able to log in unless the existing ones log out. No matter how many connections an account has, it is counted as one login account. In other words, only when all connections of the account are logged out and YDAccount.LoginCount is 0, the account can be regarded as logged out.
58	YD_ERROR_TooLowApiVersion	The API version is lower than the lowest version required by the OMS. Please refer to API Version Compatibility for details.
59	YD_ERROR_PasswordNotSet	The password is not set.
62	YD_ERROR_TooHighApiVersion	The API version is higher than the highest version supported by the OMS. Please refer to API Version Compatibility for details.

3.2.2.1 Password

YD OMS support both local password authentication and unified authentication. Futures companies and most securities companies generally use the local password authentication where passwords are saved at the server. While some securities companies have established a unified authentication system and require YD OMS to access and use the system to authenticate the password entered by the investor when logging in.

When passwords are managed by the YD OMS locally, you can set the complexity and validity period of the password. When password are managed by the unified authentication system, you can not set the complexity and validity period of the password or use API to change the password. The following descriptions of password complexity and validity period are limited to local password authentication.

The password complexity includes both the minimum password length and the number of character sets:

- The minimum password length refers to the minimum length that a new password must meet when changed. However, because the maximum password length for YD OMS is 64 bytes, the password length should be between the minimum password length and 64 bytes.
- The number of character sets in a password refers to the minimum number of character types that must be included when changing the password. Character types include digits, lowercase letters, uppercase letters, and symbols, with the specified symbols being limited to (+, *, /, `", @, #, \$, %, &, |, ~, ^, (), [], {}, ., :, !, ?, <, >, =). The number of character sets can be a number between 0 and 4: 0 indicates no restriction (allowing the use of any characters beyond the four specified sets), while 1-4 mandates the presence of at least n types of character sets in the password.

3.2.2.1.1 Password Validity Period

By default, the passwords of the investor and the administrator are permanently valid. However, brokers can set the validity period of the password for investors and administrators respectively. When the password expires, you can still log in to the YD OMS, but at this time, you can only modify the password, but cannot do any trading. After the password is successfully changed, you can use the current API instance to start trading directly without reconnecting and logging in.

YD OMS uses the following callback method to notify investors and administrators to change the password when it expires, as well as to remind investors and administrators of the remaining validity of the password. This method is called upon successful login.

```
1 | virtual void notifyServerNotice(const YDServerNotice *pServerNotice)
```

When the password expires, YD OMS will push a message to the API that the password has expired, and the ServerNoticeType of pServerNoticed in the above callback method is set to YD_SENT_PasswordExpired, at which time the investor or the administrator must change the password in order to continue trading.

When the password expiration date is less than 15 days, YD OMS pushes a message to the API that the password expiration date is approaching. pServerNoticed's ServerNoticeType in the above callback method is set to YD_SENT_PasswordNearExpiration. PasswordNearExpirationInfo.RemainDayCount is set to the number of days remaining.

Both of these types of messages are written to the API's log at the same time.

3.2.2.2 Look-through regulation

According to regulatory requirements, the look-through regulation is mainly consisted of two parts. The first part is to specify the AppID and AuthCode of the client system during login, and the second part is to report information about the login system's server. The following will introduce these two parts separately.

3.2.2.2.1 AppID and AuthCode

AppID and AuthCode are essentially the username and password of the client system, such as the strategy system, that uses API to access the OMS. The client system must first meet brokers' access testing requirements before connecting to the OMS. After passing the test, the broker will add the AppID and AuthCode representing the system to the OMS. Then, any account can connect to the OMS through this client system. It is obviously that the AppID and AuthCode of the client system are not bound to the username and password of an investor account.

The ydClient client provided by YD has its own fixed AppID and AuthCode embedded within the client. The AppID and AuthCode of this client are also added by default to the list of allowed systems on the OMS. Therefore, any investor can log in to the OMS through ydClient.

YD does not restrict the naming format of AppID, but there are regulatory requirements for AppID. It is suggested to name your client system according to regulatory requirements. Specific requirements for AppID are as follows:

- AppID consists of three parts, namely manufacturer name, software name and version number. The maximum length of each part is 10, 10 and 8 characters respectively.
- AppID should follow the following format: manufacturer name_software name_version number, e.g.: yd_client_1.0;
- For terminal software developed by individuals, the manufacturer name should be fixed as "client". Since most investors using the YD system have their own self-developed systems, the manufacturer name should be fixed as "client".

Futures company are supervised by the Margin Monitoring Center. Therefore, in futures companies' YD OMSs, you need to enter a valid AppID and AuthCode, otherwise you cannot log in. Securities companies are not supervised by the Margin Monitoring Center. Therefore, in securities' YD OMSs, AppID and AuthCode can be set to NULL, and the YD OMS will not verify these two parameters.

By default, the AppID of the client system is not bound to the investor account, that is, the client system can be shared by all investors. However, in fact, most of the client systems connected to YD OMS are exclusive strategy systems developed by investors themselves. In response to the compliance requirements of brokers, YD OMS have provided the function of binding account with AppID since version 1.486. When this function is enabled, only the bound account can use the AppID. If you use an AppID that is not bound to any account when logging in, you will receive the error YD_ERROR_InvalidClientAPP.

3.2.2.2.2 Information collection

The information that needs to be collected for regulation includes two parts, one of which must be collected from the client server and the other can be collected from the OMS. YD API will automatically collect the client server information for regulation from the client server. The information collected varies depending on the type of terminal operating system. The following only shows the information collected from the client server:

```
1 | Windows: Terminal Type (fixed to 1) @ Information Collection Time @ Private Network IP1 @ Private Network
   | IP2 @ Network Card MAC Address 1 @ Network Card MAC Address 2 @ Device Name @ Operating System Version @
   | Hard Disk SN (serial number) @ CPU SN @ BIOS SN @ System Disk Partition Information.
2 | Linux: Terminal Type (fixed to 2) @ Information Collection Time @ Private IP1 @ Private IP2 @ Network Card
   | MAC Address 1 @ Network Card MAC Address 2 @ Device Name @ Operating System Version @ Hard Disk SN @ CPU
   | SN @ BIOS SN.
```

By default, the API will collect penetrating supervision information when the dynamic link library is loaded. You can delay the collection of penetrating supervision information to when the API is started by adding the environment variable YD_DCINFO. YD_DCINFO can be any value.

During the information collection process in Linux system, BIOS SNs can be collected by YD API through the dmidecode program. Generally, if no special settings are made, normal users do not have the execution permission of this program. Users running the client system must add the execution permission of this program to ensure a complete information collection. YD will first check whether dmidecode has been given suid. If it has, it will be executed directly. If not, it will be executed through sudo dmidecode. Investors can add suid to this program through the command "chmod +x /usr/sbin/dmidecode".

When the OMS parameter "ClientReportCheck" is set to "enforce", if the OMS finds incomplete collection information during the information collection check, such as the hard disk SN of the second Linux sample mentioned above is not collected, the client login will not be allowed, and a login error of "YD_ERROR_ClientReportError" will be generated. The following shows the operating system commands used by YD API for information collection. If the collected information is found incomplete, please manually execute the corresponding collection method to confirm whether the command execution results are normal. Please execute the command with the same operating system user as the client system to ensure that problems caused by insufficient permissions (the majority of the reasons for failing to obtain data) are discovered.

- Windows Hard disk SN: Get-WmiObject -Query "SELECT SerialNumber FROM Win32_PhysicalMedia";
- Windows CPU SN: Get-WmiObject -Query "SELECT ProcessorID FROM Win32_Processor";

- Windows BIOS SN: Get-WmiObject -Query "SELECT SerialNumber FROM Win32_BIOS";
- Linux Hard Disk SN: First, find the device NAME whose TYPE is "disk" through "/bin/lslblk -dn -o TYPE,NAME", and then call "/sbin/udevadm info --query=all --name=/dev/{NAME}" to get the serial number of this device;
- Linux BIOS SN: /usr/sbin/dmidecode -s system-serial-number.

3.2.2.3 Trading day

The Trading day can be sent back to the API together with the login notifications starting from the "notifyLogin" callback, and investors can use getTradingDay to get the trading day.

```
1 | virtual int getTradingDay(void)
```

Trading days of YD OMSs are calculated based on the system time (For day trading hours, a trading day is set as the date of that natural day. The trading day before 24:00:00 (in night trading hours) is considered as the date of the next non-holiday, and after 24:00:00 (in night trading hours), if that natural day is a trading day, will be considered as the date of that natural day. If that natural day is a holiday, the trading day will be considered as the date of the next non-holiday). The Trading day calculated based on the production environment is usually accurate, but in the testing environment, it may be inconsistent with "YDMarketData.TradingDay", which is probably because that a past preday data has been used, the trading day of the exchange's test environment is not switched, or a man-made trading day is used during a weekend test.

3.2.3 Receive static data

In order to reduce the impact of the OMS queries on the OMS, all query services of YD are executed locally, which requires that the client and server have exactly the same data. This principle applies to both ydApi and ydExtendedApi. The difference is that ydApi only involves static data, while ydExtendedApi like the OMS, in addition to having static data, also manages dynamic data such as funds, positions, orders, trades and so on. Therefore, query services for funds, positions, orders and trades on ydExtendedApi are also provided locally.

Static data refers to the data that will not change within a trading day, such as the instrument, preday margin rate, preday commission rate, and account settings, etc. Each type of preday data has its corresponding structure for data storage, for example, the instrument data and account data correspond to "YDInstrument" and "YDAccount", respectively. Because some structures contain both static data and dynamic data at the same time, for example, the presettlement price (PreSettlementPrice) in the market data structure "YDMarketData" is static data, however the last price (LastPrice) is dynamic data. During the static data collection phase, a YD OMS only sends static data and does not guarantee the correctness of the dynamic data part of the structure. Therefore, it is necessary to only use the static data in each structure during the static data collection phase, and do not use dynamic data, otherwise unknown errors may be caused. The static and dynamic parts of each data structure will be indicated separately in the following text. Unless otherwise specified, all fields in the whole structure corresponding to these static data are static.

As the volume of preday data significantly increases, especially the traditional portfolio positions defined by DCE, the time required to receive static data has increased significantly. In order to accelerate the transmission speed, in version 1.386, when the API of version 1.386 is integrated with the trading system of the version 1.386, the trading system will send compressed preday data, thereby greatly reducing the transmission time. Therefore, if you want to speed up the reception of static data, please upgrade the API to the version 1.386.

After a successful login, the OMS will start to push static data. After the data is received by API, the market data connection and XTCP (if enabled) connection will be established. The "notifyFinishInit" will be called back to notify the strategy program that all static data has been loaded regardless of whether the connection is successful or not, which provides investors a time point to prepare for the subsequent receipt of order and trade notifications, including obtaining information such as preday funds, positions, margin rates, and commission rates, etc.

```
1 | virtual void notifyFinishInit(void)
```

In addition to knowing the completion of the static data push through the callback function, YD API also provides a synchronous query function to check whether the static data push is completed. If you do not want to receive callback messages during the static function receiving phase, this function can be used to determine whether the current initialization data is complete or not.

```
1 | virtual bool hasFinishedInit(void)
```

The static data of YD will be introduced one by one below. The functions involved in the following content can be used normally in "notifyFinishedInit" callback and afterwards.

3.2.3.1 Exchanges

YD currently supports a total of 8 exchanges, including China Financial Futures Exchange (CFFEX), Shanghai Futures Exchange (SHFE), Shanghai International Energy Exchange (INE), Dalian Commodity Exchange (DCE), Guangzhou Futures Exchange (GFEX), China Zhengzhou Commodity Exchange (CZCE), Shanghai Stock Exchange (SSE, option) and Shenzhen Stock Exchange (SZSE, option). It also supports configuring connections to all exchanges on one OMS. SHFE and INE are usually configured on one YD OMS, and therefore, YD provides two sets of methods for traversing and searching for exchanges, as shown below:

```
1 | virtual int getExchangeCount(void)
2 | virtual const YDExchange *getExchange(int pos)
3 | virtual const YDExchange *getExchangeByID(const char *exchangeID)
```

The first two methods can be used for traversing all exchanges, the specific methods are shown in the code below:


```

1  for(int i = 0; i < pApi->getExchangeCount(); i++) {
2      YDExchange *exchange = pApi->getExchange(i);
3  }

```

The last method can be used for specifying and searching for a single exchange, and the invocation for searching each exchange is shown below:

```

1  pApi->getExchangeByID("CFFEX") //CFFEX
2  pApi->getExchangeByID("SHFE")  //SHFE
3  pApi->getExchangeByID("INE")   //INE
4  pApi->getExchangeByID("DCE")   //DCE
5  pApi->getExchangeByID("GFEX")  //GFEX
6  pApi->getExchangeByID("CZCE")  //CZCE
7  pApi->getExchangeByID("SSE")   //SSE
8  pApi->getExchangeByID("SZSE")  //SZSE

```

All fields in the YDExchange structure are static data, and most of them are used to indicate whether a certain functionality is supported. Therefore, if you want to know the meaning and usage of a specific field in detail, please search for that field in this document to obtain a complete introduction related to that functionality.

3.2.3.2 Products

YD provides two sets of methods for traversing and searching products.

```

1  virtual int getProductCount(void);
2  virtual const YDProduct *getProduct(int pos);
3  virtual const YDProduct *getProductByID(const char *productID);

```

The first two methods can be used for traversing all products of all exchanges, if multiple exchanges are configured on a YD OMS at this time. The specific methods are shown in the following code:

```

1  for(int i = 0; i < pApi->getProductCount(); i++) {
2      YDProduct *product = pApi->getProduct(i);
3  }

```

The last method can be used to specify and search for a product. The following shows an invocation example for searching for a product. If you want to know the expressions of all YD products, please use the above method to traverse and view "YDProduct.ProductID" one by one:

```

1  pApi->getProductByID("cu")
2  pApi->getProductByID("cu_o")
3  pApi->getProductByID("IC")

```

All fields in the YDProduct structure are static data, most of which are consistent with those of YDInstrument. They also include the attributes that should belong to an instrument, such as "Multiple", "Tick", "UnderlyingMultiple", "MaxMarketOrderVolume", "MinMarketOrderVolume", "MaxLimitOrderVolume" and "MinLimitOrderVolume". The original meaning of them is that when the corresponding fields on the instrument are not assigned, they can be used as their default values. However, at present, all instruments have their attribute values properly set, and there will be no null values. Therefore, the instrument attributes on "YDProduct" have little significance. For the meanings of the fields, please refer to the relevant content of [Instrument](#).

m_PMarginProduct is used for calculating large-side margins and cross-product large-side margins. Please refer to [Margin Deduction](#) for details.

3.2.3.3 Instrument

YD provides two sets of methods for traversing and searching for instruments.

```

1  virtual int getInstrumentCount(void)
2  virtual const YDInstrument *getInstrument(int pos)
3  virtual const YDInstrument *getInstrumentByID(const char *instrumentID)

```

The first two methods can be used for traversing all instruments. If multiple exchanges are configured on a YD OMS at this time, the instruments of all exchanges will be traversed. The specific methods are shown in the following code:

```

1  for(int i = 0; i < pApi->getInstrumentCount(); i++) {
2      YDInstrument *instrument = pApi->getInstrument(i);
3  }

```

The last method can be used to specify and search for an instrument. The following shows an invocation example for searching for an instrument. If you want to know the expressions of all YD instruments, please use the above method to traverse and view "YDInstrument.InstrumentID" one by one:

```

1  pApi->getInstrumentByID("cu2208")

```

"YDInstrument" is the core data structure of the YD system. Except for the "AutoSubscribed" and "UserSubscribed" fields, all other data are static. The following will introduce the meanings of key fields:

Field	Description
InstrumentID	Instrument code, e.g.: cu2208, SP c2211&c2301
InstrumentHint	Hinted instrument information, applicable to the ETF options of SSE/SZSE, for example: 510050C2009M02350
ProductClass	Product class of instrument YD_PC_Futures=1: futures YD_PC_Options=2: Options YD_PC_Combination=3: combination, e.g.: SP c2211&c2301. At present, arbitrage instruments for DCE and CZCE are supported YD_PC_Index=4: index, e.g.: CSI 300 index YD_PC_Cash=5: cash, e.g.: CSI 300ETF
DeliveryYear	Delivery year, e.g.: 2022
DeliveryMonth	Delivery month, e.g.: "1" represents January, and "12" represents December.
ExpireDate	Expiration date of instrument, e.g.: 20220808
ExpireTradingDayCount	The number of trading days (excluding weekends and holidays) between the current trading day and the expiration date of the instrument. Since the calculation of the remaining days depends on the trading calendar, for instruments that expire in the next year, the calculation of the remaining days is only accurate after the exchange officially releases the trading calendar for the next year at the end of the current year. Otherwise, the trading calendar required for the remaining day calculation is estimated by YD, which may differ from the remaining days calculated based on the official trading calendar. If it is the last trading day, the value will be "0".
Multiple	Instrument multiplier, the multiple relative to the underlying asset.
Tick	Minimum price change.
MaxMarketOrderVolume	Maximum volume of market orders.
MinMarketOrderVolume	Minimum volume of market orders, the order volume must be a multiple of it.
MaxLimitOrderVolume	Maximum volume of price-limited orders.
MinLimitOrderVolume	Minimum volume of price-limited orders, the order volume must be a multiple of it.
OptionsType	Option type YD_OT_NotOption=0: non-option YD_OT_CallOption=1: call option YD_OT_PutOption=2: put option
StrikePrice	Strike price, valid only when the instrument aims to an option
m_pUnderlyingInstrument	The underlying instrument's pointer of an option instrument, which can be used to obtain the relevant information about the underlying instrument. This is only valid when the instrument aims to an option
UnderlyingMultiply	Underlying multiplier, the multiplier of the option relative to its underlying instrument, which is only valid when the instrument aims to an option It is always "1" for other non-option instruments
m_pMarketData	Pointer to the market data structure, which can be used to obtain market data. This market data structure can be refreshed continuously with the change of the market data.
m_pLegInstrument[2]	Only applicable to combined instruments. m_pLegInstrument[0] points to the left leg instrument of a combined instrument, m_pLegInstrument[1] points to the right leg instrument of a combined instrument
LegDirction[2]	Only applicable to combined instruments. LegDirction[0] represents the trading direction of the left leg instrument of a combined instrument, and LegDirction[1] represents the trading direction of the right leg instrument of a combined instrument
m_pCombPositionDef[2] [YD_MaxHedgeFlag]	Only applicable to combined instruments. m_pCombPositionDef[0] points to the combined definition array of each hedge flag when buying a combined instrument, m_pCombPositionDef[1] points to the combined definition array of each hedge flag when selling a combined instrument. The hedge flags should be converted to array subscripts through a reduction by "1".

Assume that the instrument multiplier of a futures is "5", which means that a per-lot futures instrument corresponds to 5 units of underlying assets. If the option of this per-lot futures instrument corresponds to two-lot of futures instrument, the "UnderlyingMultiply" of the option will be 2, and the instrument multiplier of this option will be $5 \times 2 = 10$, indicating that the option of the per-lot futures instrument corresponds to 10 units of underlying assets.

3.2.3.4 Traditional combined position definition

In the traditional margin model, the traditional combined position definition of DCE, GFEX, CZCE, SSE and SZSE can be queried through the following two APIs.

The first method aims to traverse all traditional combined position definitions. Firstly, obtain the count *n* of all traditional combined position definitions through the function "getCombPositionDefCount", and then traverse all traditional combined position definitions one by one from 0 to *n*-1 through the function "getCombPositionDef".

```
1 // Traversing all traditional combined position definitions by SNS
2 virtual int getCombPositionDefCount(void);
3 virtual const YDCombPositionDef *getCombPositionDef(int pos);
```

The second method aims to query a specific traditional combined position definition according to traditional combined position definition names and combined hedge flags. Traditional combined position definition names (combPositionID) are not completely consistent with those of exchanges. Please refer to the following text for the definitions of the traditional combined position definition names and combined hedge flags.

```
1 // Querying a traditional combined position definition according to traditional combined position
  definition names and hedge flags
2 virtual const YDCombPositionDef *getCombPositionDefByID(const char *combPositionID,int combHedgeFlag);
```

The information of "YDCombPositionDef" obtained through the above APIs is as follows:

Statement	Description
YDInstrumentID CombPositionID	Traditional combined position definition names, e.g.: pg2302,-eg2303 c2207-C-2240,-c2207-C-2240 20008571,-20008572
int CombPositionRef	Internal reference No. of traditional combined position definitions
int ExchangeRef	Internal reference No. of exchanges
int Priority	Priority of traditional combined position definitions. The smaller the number, the higher the priority. It may be -1, indicating that the traditional combined positions of an exchange are not subject to priority management, for example, CZCE
short CombHedgeFlag	Hedge flags of traditional combined position definitions, which can be of the following types: YD_CHF_SpecSpec: speculation - speculation YD_CHF_SpecHedge: speculation - hedge YD_CHF_HedgeHedge: hedge - hedge YD_CHF_HedgeSpec: hedge - speculation
short CombPositionType	Types of traditional combined position definitions. The types of traditional combined position definitions of each exchange are shown in the following table.
double Parameter	Parameter, provided with different meanings in different exchanges. At present, it is applicable only to option offset, buying option vertical spread and buying option and futures combination in DCE and GFEX. It represents a combined margin discount. 0.2 means the combined margin is 20% of the total of the original two-legged margins.
const YDExchange *m_pExchange	Pointer of YDExchange.
const YDInstrument *m_plnstrument[2]	Two-legged instrument of traditional combined position definition. The order of the two legs may not be the same as the order of CombPositionID, and YD can order them automatically depending on the business process.
int PositionDirection[2]	Corresponding to the two-leg position direction of m_plnstrument
int HedgeFlag[2]	Corresponding to the two-leg hedge flags of m_plnstrument
int PositionDate[2]	Corresponding to the two-leg position date of m_plnstrument. Currently, all positions are historical positions.

The types of traditional combined positions for each exchange are shown in the following table.

Exchange	Type of traditional combined positions	Description
DCE	YD_CPT_DCE_FuturesOffset=0	Futures offset
DCE	YD_CPT_DCE_OptionsOffset=1	Options offset
DCE	YD_CPT_DCE_FuturesCalendarSpread=2	Futures calendar spread
DCE	YD_CPT_DCE_FuturesProductSpread=3	Futures cross-product spread
DCE	YD_CPT_DCE_BuyOptionsVerticalSpread=4	Vertical spread of buy options
DCE	YD_CPT_DCE_SellOptionsVerticalSpread=5	Vertical spread of sell options

Exchange	Type of traditional combined positions	Description
DCE	YD_CPT_DCE_OptionsStraddle=7	Straddle put options
DCE	YD_CPT_DCE_OptionsStrangle=8	Strangle Put options
DCE	YD_CPT_DCE_BuyOptionsCovered=9	Buy option covered
DCE	YD_CPT_DCE_SellOptionsCovered=10	Sell option covered
GFEX	YD_CPT_GFEX_FuturesOffset=0	Futures offset
GFEX	YD_CPT_GFEX_OptionsOffset=1	Options offset
GFEX	YD_CPT_GFEX_FuturesCalendarSpread=2	Futures calendar spread
GFEX	YD_CPT_GFEX_FuturesProductSpread=3	Futures cross-product spread
GFEX	YD_CPT_GFEX_BuyOptionsVerticalSpread=4	Vertical spread of buy options
GFEX	YD_CPT_GFEX_SellOptionsVerticalSpread=5	Vertical spread of sell options
GFEX	YD_CPT_GFEX_OptionsStraddle=7	Straddle put options
GFEX	YD_CPT_GFEX_OptionsStrangle=8	Strangle Put options
GFEX	YD_CPT_GFEX_BuyOptionsCovered=9	Buy option covered
GFEX	YD_CPT_GFEX_SellOptionsCovered=10	Sell option covered
CZCE	YD_CPT_CZCE_Spread=50	Spread
CZCE	YD_CPT_CZCE_StraddleStrangle=51	Short straddle or strangle
CZCE	YD_CPT_CZCE_SellOptionConvered=52	Sell option covered
SSE, SZSE	YD_CPT_StockOption_CNSJC=100	Bull call spread
SSE, SZSE	YD_CPT_StockOption_CXSJC=101	Bear call spread
SSE, SZSE	YD_CPT_StockOption_PNSJC=102	Bull put spread
SSE, SZSE	YD_CPT_StockOption_PXSJC=103	Bear put spread
SSE, SZSE	YD_CPT_StockOption_KS=104	Short straddle
SSE, SZSE	YD_CPT_StockOption_KKS=105	Short strangle

3.2.3.5 Preday market data

To obtain preday market data, the market data structure "YDMarketData" should be accessed through the "m_pMarketData" pointer of instruments.

Most of the information from "YDMarketData" is dynamic. Only those fields related to the previous trading day are static data. The static data part is shown in the following table:

Field	Description
TradingDay	Current trading day
PreSettlementPrice	Pre-settlement price
PreClosePrice	Pre-close price
PreOpenInterest	Pre-position volume
UpperLimitPrice	Upper limit price
LowerLimitPrice	Lower limit price

3.2.3.6 Account

For ordinary investors, the YD system only provides one way to obtain their account information, which can only be used by investors. Administrator users are not allowed to use this method.

```

1 // ydApi and ydExtendedApi can be called
2 virtual const YDAccount *getMyAccount(void)
3
4 // Only ydExtendedApi can be called
5 virtual const YDExtendedAccount *getExtendedAccount(const YDAccount *pAccount=NULL)
```

The static data of YDAccount and YDExtendedAccount is shown in the following table:

Field	Description
AccountID	Fund account
PreBalance	Pre-balance

Field	Description
WarningLevel1	Risk warning level 1. Useless now, just ignore it
WarningLevel2	Risk warning level 2. Useless now, just ignore it
AccountFlag	Account function switch

AccountFlag is a bitmap where each binary position with 1 indicates the function is enabled and 0 indicates the function is disabled. Assuming the investor has enabled the functionality of the OMS Notification and Order Number Checking, then the decimal number of the AccountFlag will be 112, and the corresponding binary number will be 0b1110000.

The following code can be used to check whether the function of "YD_AF_NotifyOrderAccept" is enabled:

```
1  if (myAccount->AccountFlag & YD_AF_NotifyOrderAccept) {
2      // server notification enabled
3  }
```

The list of functionalities that can be controlled at the account level is as follows:

Switch value	Description
YD_AF_SelectConnection	For determining whether the result of connection selection can be uploaded to the OMS for all users' operation, refer to reporting the connection selection result for details.
YD_AF_AutoMakeCombPosition	Not Suggested. For determining whether it can be combined through a broker's ydAutoCP. Refer to Auto tool for details.
YD_AF_BareProtocol	For determining whether to allow orders made based on a raw protocol, refer to Raw Protocol for details.
YD_AF_DisableSelfTradeCheck	For determining whether to disable the self-trading check, refer to Self-Trading Check for details.
YD_AF_NotifyOrderAccept	For determining whether to send the OMS notifications through the OMS, refer to Order Notification for details.
YD_AF_OrderRefCheck	For determining whether to check the monotonic increase of OrderRef. Refer to Monotonic Increase Check of Order Numbers for details.

3.2.3.7 Preday derivatives positions

A derivative position is the position of futures and options. Since the basic operation of preday position is traversing, the real-time position can be calculated based on preday position, so YD only provides a method for traversing all preday positions.

```
1  // Derivatives
2  virtual int getPrePositionCount(void)
3  virtual const YDPrePosition *getPrePosition(int pos)
4
5  The method for traversing all positions is as follows:
6
7  ```C++
8  for(int i = 0; i < pApi->getPrePositionCount(); i++) {
9      YDPrePosition *prePosition = pApi->getPrePosition(i);
10 }
```

The preday position structure of derivatives is as follows:

Field	Description
m_pAccount	Position account
m_pInstrument	Position instrument
PositionDirection	Position direction
HedgeFlag	Hedge flag
PrePosition	Preday position
PreSettlementPrice	Pre settlement price
AverageOpenPrice	Average open price

3.2.3.8 Preday holdings

A holding is the position of stocks, funds and bonds on the SSE and SZSE. Since the basic operation of preday position is traversing, the real-time position can be calculated based on preday position, so YD only provides a method for traversing all preday positions.

```
1  virtual int getPreHoldingCount(void)
2  virtual const YDPreHolding *getPreHolding(int pos)
```

The method for traversing all positions is as follows:

```

1  for(int i = 0; i < pApi->getPreHoldingCount(); i++) {
2      YDPreHolding *prePosition = pApi->getPreHolding(i);
3  }

```

The preday position structure of holdings is as follows:

Field	Description
m_pAccount	Position account
m_pInstrument	Position instrument
PreHolding	Preday holding
AverageOpenPrice	Average open price

3.2.3.9 Preday spots positions

A spot position is the spot position of the underlying of the stock option. Since the basic operation of preday position is traversing, the real-time position can be calculated based on preday position, so YD only provides a method for traversing all preday positions.

```

1  virtual int getSpotPrePositionCount(void)
2  virtual const YDSpotPrePosition *getSpotPrePosition(int pos)

```

The method for traversing all positions is as follows:

```

1  for(int i = 0; i < pApi->getSpotPrePositionCount(); i++) {
2      YDSpotPrePosition *prePosition = pApi->getSpotPrePosition(i);
3  }

```

The preday position structure of spots is as follows:

Field	Description
m_pAccount	Position account
m_pInstrument	Position instrument
Position	Preday position
ExchangeFrozenVolume	Frozen volume of exchange
ExecAllocatedVolume	Net executed allocated volume - positive value means receipt of securities, negative value means delivery of securities, only works on E+1 day
ExecAllocatedAmount	Net executed allocated amount - positive value means receiving money, negative value means paying money, only works on E+1 day
ExecAllocatedFrozenVolume	Net executed allocated frozen volume, only works on E+1 day
ExecAllocatedFrozenAmount	Net executed allocated frozen amount, only works on E+1 day

3.2.3.10 Traditional combined preday position

In the traditional margin model, traditional combined preday position data is static. Theoretically, it should also be transmitted during static data transmission, namely, before the function "notifyFinishInit" operation. However, in order to achieve the compatibility with the old version of API, it is always transmitted to the client immediately after the function "notifyFinishInit" and before the function "notifyCaughtUp" operations. Although the transmission time is different from that for other static data, traditional combined preday positions, just like other static data, can only be pushed once at the first login. That is to say, if there is a "notifyFinishInit" callback, it can be pushed to the client, however, it will not be pushed repeatedly even due to disconnection and other reasons. The traditional combined preday positions can be introduced together with other static data for the sake of proper concept and easy understanding.

For an API, no any query interface has been provided, so it can only collect all traditional combined preday positions relying on the "notifyCombPosition" callback function.

```

1  virtual void notifyCombPosition(const YDCombPosition *pCombPosition, const YDCombPositionDef
    *pCombPositionDef, const YDAccount *pAccount)

```

All fields of the "YDCombPosition" structure are static data, which should be used together with other parameters involved in the callback:

Parameter	Field	Description
pAccount		Position account
pCombPositionDef		Traditional combined position definition
pCombPosition	Position	Position volume
	CombPositionDetailID	Traditional combined position details ID, which is only meaningful to SSE and SZSE

3.2.3.11 System parameter

System parameters are those used in the calculation process. Currently, they are only used for margin calculation.

YD provides two methods for traversing and searching for data.

```
1 virtual int getSystemParamCount(void)
2 virtual const YDSystemParam *getSystemParam(int pos)
3 virtual const YDSystemParam *getSystemParamByName(const char *name, const char *target)
```

The first two methods can be used for traversing all system parameters. The specific method is shown in the following code:

```
1 for(int i = 0; i < pApi->getSystemParamCount(); i++) {
2     YDSystemParam *param = pApi->getSystemParam(i);
3 }
```

The third method can be used for specifying and searching for system parameters. The following shows an example for call searching:

```
1 getSystemParamByName("MarginBasePrice", "Futures")
```

The YDSystemParam data is static, and its structure is as follows:

Field	Description
Name	Parameter name
Target	Applicable targets of parameters
Value	Parameter value

The above Name and Target are strings. Possible combinations are shown in the following table:

Name	Target	Value
MarginLowerBoundaryCoef	MarginCalcMethod1	The boundary coefficient in the stock index option margin algorithm, the floating point number, default: 0.5
MarginBasePrice	Futures or Options	The base prices used for calculating the margin of futures or short positions of option can be selected as follows: 0: Pre settlement price 1: Opening price 2: Latest price 3: Average market price 4: The higher value between the latest price and pre settlement price According to the current common rules, the base prices used for futures and options shall be 1 and 4, respectively
OrderMarginBasePrice	Futures or Options	The base prices used for calculating the frozen margin of open position orders for futures or put options can be selected as follows: 0: Pre settlement price 5: Order price (for market orders, use the upper limit price) 7: Use the MarginBasePrice price According to the current common rules, the base prices used for futures and options shall be 5 and 0, respectively Note that the frozen margin in the application for option execution is always calculated based on the pre settlement price
MarginBasePriceAsUnderlying	Options	The underlying product prices used for calculating the put option margin can be selected as follows: 0: Pre settlement price 2: Latest price 4: The higher value between the latest price and pre settlement price According to the current common rules, the underlying product price used shall be 0

Name	Target	Value
SellOrderPremiumBasePrice	Options	<p>The base prices used for calculating the reversely frozen premium when selling open position options can be selected as follows:</p> <p>0: Pre settlement price 5: Order price (for market orders, use the lower limit price) 6: Non-reverse freezing</p> <p>According to the current common rules, the base prices used shall be 6</p> <p>Note that the premium frozen when buying open position options is always the order price (for market orders, use the upper limit price)</p>
PortfolioMarginConcession	DCELongOptionPortfolio	<p>For the traditional combined positions with long option positions included in DCE and GFEX, the following values can be selected to determine whether to reduce the calculated margin:</p> <p>0: Without reduction 1: With reduction. Since YD does not check the funds when closing positions, in extreme cases, it may lead to overdraft of funds due to overcharge of margin when closing positions after buying option combinations.</p>
UseCollateral	Account	<p>When the collateral funds obtained from collaterals are included in the pre equity for open position, the following values can be selected:</p> <p>0: Disable 1: Enable</p>
MarketMaker	System	Does the OMS support market maker quote or not yes: Yes no: No
Test	System	Is the OMS a test system? A test system does not involve performance optimization and cannot be used for production
ServerVersion	System	Version number of the OMS. Refer to Version Rules for the detailed version number specification
ConnectionMode	System	<p>Connection modes of the OMS:</p> <p>0: All management seat 1: First mangement others non-management seat 2: All non-management seat 3: Management seats used as non-management seats, can only be used in SHFE and CFFEX</p>
MinApiVersion	System	Minimum API version number supported by the OMS. Refer to Version Rules for the detailed version number specification
MaxApiVersion	System	Maximum API version number supported by the OMS. Refer to Version Rules for the detailed version number specification
MinSelectConnectionGap	System	Minimum time gap for reporting the connection selection result, in milliseconds
MaxSelectConnectionGap	System	Maximum effective time for reporting the connection selection result, in milliseconds
MissingOrderGap	System	Timeout of an unknown order determined by the system, in milliseconds.
UDPTradingPort	System	UDP trading port, which will be 0 when the UDP service is disabled on the OMS.
XTCPTradingPort	System	XTCP trading port, which will be 0 when the XTCP service is disabled on the OMS.
TradingSegmentDetail	System	Is the detailed trading segment announcement enabled? yes: Yes no: No
ClientMinPasswordLen	System	Investor's minimum password length 0 indicates no minimum length limit.
ClientMinPasswordCharSet	System	Investor's password character set count 0 indicates no character set count limit
AdminMinPasswordLen	System	Administrator's minimum password length 0 indicates no minimum length limit
AdminMinPasswordCharSet	System	Administrator's password character set count 0 indicates no character set count limit

3.2.3.12 Cash Commission Rate

The setting values of the cash commission rate are sparse. For example, to set the cash commission rate for all investors at the product level, only one record needs to be set at the product level. The YD OMS will apply this setting value to all contracts belonging to this product for all investors. These setting values are mainly used to display in the GUI and have no practical meaning for investors.

Here is the method for obtaining the cash commission rate, for reference only, without detailed explanation.

```
1 // Cash Commission Rate
2 virtual int getCashCommissionRateCount(void)
3 virtual const YDCashCommissionRate *getCashCommissionRate(int pos)
```

Investors can use `getInstrumentCashCommissionRate` to get the cash commission rate of the instrument in corresponding hedge flag. It is not recommended to get it directly from [Account Level Information](#)'s `YDAccountInstrumentInfo`.

```
1 virtual const YDCashCommissionRate *getInstrumentCashCommissionRate(const YDInstrument *pInstrument, int
  ydOrderFlag, int direction, const YDAccount *pAccount=NULL)
```

The field descriptions for the returned cash commission rate `YDCashCommissionRate` are shown below:

Field	Description
SubProductClass	YD_SPC_Other=0: Other YD_SPC_Stock=1: Stock YD_SPC_Bond=2: Bond YD_SPC_Fund=3: Fund
YDOrderFlag	YD_YOF_Normal=0: Normal trading YD_YOF_Designation=11: SZSE transfer of custody
Direction	YD_D_Buy=0: Buy YD_D_Sell=1: Sell
RatePiece[YD_CCT_Count]	Cash commission rate item array, each array element is a type of fee YDCashCommissionRatePiece. YD_CCT_StampDuty: Stamp duty YD_CCT_SecuritiesManagementFee: Securities management fee YD_CCT_HandlingFee: Handling fee YD_CCT_TransferFee: Transfer fee
m_pInstrument	the pointer to the instrument
m_pProduct	the pointer to the product
m_pExchange	the pointer to the exchange
m_pAccount	the pointer to the account

Note

`m_pInstrument`, `m_pProduct`, `m_pExchange`, and `m_pAccount` point to the level of the current rate structure setting. These fields in the rate structure of a specific contract are usually irrelevant to the instrument, so investors usually do not need to pay attention to the values of these fields. For example, a rate setting on the SSE (`m_pExchange` points to the SSE) will be applied to all securities of the SSE. All SSE securities point to the same structure, and the `m_pExchange` of these structures are all SSE pointers, which have nothing to do with these instruments.

The structure of `YDCashCommissionRatePiece` is as follows:

Field	Description
RateByAmount	Rate based on amount
RateByVolume	Rate based on volume
MaxValue	Maximum value
MinValue	Minimum value

3.2.3.13 Brokerage Fee Rate

The setting values of the brokerage fee rate are sparse. For example, to set the brokerage fee rate for all investors at the product level, only one record needs to be set at the product level. The YD OMS will apply this setting value to all contracts belonging to this product for all investors. These setting values are mainly used for display in the GUI and have no practical meaning for investors.

Here is the method for obtaining the brokerage fee rate, for reference only, without detailed explanation.

```
1 // Brokerage Fee Rate
2 virtual int getBrokerageFeeRateCount(void)
3 virtual const YDBrokerageFeeRate *getBrokerageFeeRate(int pos)
```

Investors can use `getInstrumentBrokerageFeeRate` to get the brokerage fee rate of the instrument in corresponding hedge flag. It is not recommended to read it directly from [Account Level Information](#)'s `YDAccountInstrumentInfo`.


```
1 virtual const YDBrokerageFeeRate *getInstrumentBrokerageFeeRate(const YDInstrument *pInstrument, int
  ydorderFlag, int direction, const YDAccount *pAccount=NULL)
```

The field descriptions for the returned brokerage fee rate YDBrokerageFeeRate are shown below:

Field	Description
SubProductClass	YD_SPC_Other=0: Other YD_SPC_Stock=1: Stock YD_SPC_Bond=2: Bond YD_SPC_Fund=3: Fund
YDOrderFlag	YD_YOF_Normal=0: Normal trading YD_YOF_Designation=11: SZSE transfer of custody
Direction	YD_D_Buy=0: Buy YD_D_Sell=1: Sell
RatePiece	Brokerage fee detailed parameters, YDCashCommissionRatePiece please refer to Cash Commission Rate
m_pInstrument	pointer to the instrument
m_pProduct	pointer to the product
m_pExchange	pointer to the exchange
m_pAccount	pointer to the account

Note

m_pInstrument, m_pProduct, m_pExchange, and m_pAccount point to the level of the current rate structure setting. These fields in the rate structure of a specific contract are usually irrelevant to the contract, so investors usually do not need to pay attention to the values of these fields. For example, a rate set on the SSE (m_pExchange points to the SSE) will be applied to all securities on the SSE. All SSE securities point to the same structure, and the m_pExchange of these structures are all SSE pointers, which have nothing to do with these contracts.

3.2.3.14 Commission Rate

The setting values of the commission rate are sparse. For example, to set the commission rate for all investors at the product level, only one record needs to be set at the product level. The YD OMS will apply this setting value to all contracts belonging to this product for all investors. These setting values are mainly used for display in the GUI and have no practical meaning for investors.

Here is the method for obtaining the commission rate, for reference only, without detailed explanation.

```
1 // Commission Rate
2 virtual int getCommissionRateCount(void)
3 virtual const YDCommissionRate *getCommissionRate(int pos)
```

Investors can use getInstrumentBrokerageFeeRate to get the commission rate of the instrument in corresponding hedge flag. It is not recommended to get it directly from [Account Level Information](#)'s YDAccountInstrumentInfo.

```
1 virtual const YDCommissionRate *getInstrumentCommissionRate(const YDInstrument *pInstrument, int
  hedgeFlag, const YDAccount *pAccount=NULL)
```

The field descriptions for the returned commission rate are shown below:

Parameter	Field	Description
YDCommissionRate	OpenRatioByMoney	open ratio by money
	OpenRatioByVolume	open ratio by volume
	CloseRatioByMoney	close ratio by money
	CloseRatioByVolume	close ratio by volume
	CloseTodayRatioByMoney	close today ratio by money
	CloseTodayRatioByVolume	close today ratio by volume
	OrderCommByVolume	commission per order
	OrderActionCommByVolume	cancellation commission per order
	ExecRatioByMoney	exercise ratio by money
	ExecRatioByVolume	exercise ratio by volume

3.2.3.15 Message count commission rate

The message count commission rate encompasses the standards for commission charged by various exchanges. Given that certain products from some exchanges have not initiated the levying of the message count commission, the method described below will exclusively retrieve parameter information related to the products for which message count commission have been imposed:

```

1 // message count commission rate
2 virtual int getMessageCommissionRateCount(void)
3 virtual const YDMessageCommissionRate *getMessageCommissionRate(int pos)

```

The field information for YDMessageCommissionRate is as follows:

Field	Description
ProductRef	Product reference
MessageCount	Message count level
OTR	OTR level
CommissionRate	message count commission rate

The Order-to-Trade Ratio (OTR) and message count disclosed by exchanges are interval values, whereas YD provides level parameters. The correspondence between the two is depicted in the following two tables.

The following is the copper futures (cu) message count commission rate announced by SHFE in 2022:

	$OTR \leq 2$	$OTR > 2$
$1 \leq message\ count \leq 4000$	0 yuan/count	0 yuan/count
$4001 \leq message\ count \leq 8000$	0.25 yuan/count	0.5 yuan/count
$4001 \leq message\ count \leq 8000$	1.25 yuan/count	2.5 yuan/count
$40001 \leq message\ count$	25 yuan/count	50 yuan/count

The following are the corresponding settings for YD, assuming that the ProductRef for copper futures (cu) is 8:

ProductRef	OTR	MessageCount	CommissionRate
8	0	0	0
8	0	4000	0.25
8	0	8000	1.25
8	0	40000	25
8	2	0	0
8	2	4000	0.5
8	2	8000	2.5
8	2	40000	50

Let's presume that a certain investor has a message count of 9000 on a cu contract, with a corresponding OTR of 3. Therefore, the message count commission calculated using the segment accumulation method would be:

- First segment: 4000 total, commission of 0 yuan
- Second segment: 4000 total, commission of $4000 \times 0.5 = 2000$ yuan
- Third segment: 4000 total, commission of $1000 \times 2.5 = 2500$ yuan
- The total message count commission amounts to 4500 yuan.

3.2.3.16 Margin rate

The setting values of the margin rate are sparse. For example, to set the margin rate for all investors at the product level, only one record needs to be set at the product level. The YD OMS will apply this setting value to all instruments belonging to this product for all investors. These setting values are mainly used to display in the GUI and have no practical meaning for investors.

Here is the method for obtaining the margin rate, for reference only, without detailed explanation.

```

1 // Margin rate
2 virtual int getMarginRateCount(void)
3 virtual const YDMarginRate *getMarginRate(int pos)

```

Investors can use getInstrumentMarginRate to get the margin rate of the instrument in corresponding hedge flag. It is not recommended to get it directly from [Account Level Information](#)'s YDAccountInstrumentInfo.

```

1 virtual const YDMarginRate *getInstrumentMarginRate(const YDInstrument *pInstrument, int hedgeFlag, const
  YDAccount *pAccount=NULL)

```

The YDMarginRate structure has a large number of "union" fields for adapting to different margin models. For the sake of easy understanding, margin rate parameters under three different margin models are listed below.

The following fields are apply to futures margins of futures exchanges.

Field	Description
m_pAccount	Pointer of investors corresponding to margin rate
m_pProduct	Pointer of the product corresponding to margin rate
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of "YDExchange.UseArbitragePosition" is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge
LongMarginRatioByMoney	Long margin rate based on amount
LongMarginRatioByVolume	Long margin rate based on volume
ShortMarginRatioByMoney	Short margin rate based on amount
ShortMarginRatioByVolume	Short margin rate based on volume

The following fields are apply to the commodity option margins of futures exchanges and the stock index option margin of CFFEX.

Field	Description
m_pAccount	Pointer of investors corresponding to margin rate
m_pProduct	Pointer of the product corresponding to margin rate
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of YDExchange.UseArbitragePosition is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge
CallMarginRatioByMoney	Short margin rate of call option based on amount
CallMarginRatioByVolume	Short margin rate of call option based on volume
PutMarginRatioByMoney	Short margin rate of put option based on amount
PutMarginRatioByVolume	Short margin rate of put option based on volume

The following fields are apply to the stock option margins of SSE and SZSE.

Field	Description
m_pAccount	Pointer of investors corresponding to margin rate
m_pProduct	Pointer of the product corresponding to margin rate
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Normal=1: Normal YD_HF_Covered=3: Covered
BaseMarginRate	Margin rate of base instrument
LinearFactor	Linear factor
LowerBoundaryCoef	Minimum boundary coefficient

3.2.3.17 Margin adjustment during trading

Generally, the margin rate is fixed during trading, however, in some cases brokers may adjust the margin for some customers during trading. When a broker updates the margin rate, investors will receive the following callbacks:

```
1 | virtual void notifyUpdateMarginRate(const YDUpdateMarginRate *pUpdateMarginRate)
```

When the margin rate changes, it is unnecessary for investors using ydExtendedApi to worry about which instruments have changed and which position margins need to be updated. ydExtendedApi can properly manage the relevant change of the margin rate, however investors using ydApi should independently update the corresponding instrument position margins with the change of the margin rate. The following table shows the instrument filter fields of updated margin rate "YDUpdateMarginRate" affecting the margin rate. The margin rate related fields are not listed. See the margin rate descriptions corresponding to each margin model mentioned above for details if necessary.

Field	Description
AccountRef	Account reference No.

Field	Description
ProductRef	Product reference No.
InstrumentRef	Instrument reference No.
HedgeFlagSet	Hedge flag set. If a hedge flag is affected, the bit corresponding to "1<<HedgeFlag will be set"
OptionTypeSet	Option type set. If an option type is affected, the bit corresponding to "1<<OptionsType will be set"
ExpireDate	Instrument expiry date, mainly used for representing the option series
UnderlyingInstrumentRef	Underlying instrument reference No
Multiple	Instrument multiplier, the margin rate of stock options may be adjusted due to the affection of dividend

The following shows the example code for determining whether the change of the received margin rate will affect the positions (instrument and hedge flag):

```

1  bool applyToInstrument(const CYUpdateMarginRate *pUpdateMarginRate, const YDInstrument *pInstrument, int
   hedgeFlag) const
2  {
3      if ((pUpdateMarginRate->ProductRef>=0) && (pInstrument->ProductRef!=pUpdateMarginRate->ProductRef))
4      {
5          return false;
6      }
7      if ((pUpdateMarginRate->InstrumentRef>=0) && (pInstrument->InstrumentRef!=pUpdateMarginRate->
   InstrumentRef))
8      {
9          return false;
10     }
11     if ((pUpdateMarginRate->UnderlyingInstrumentRef>=0) && (pInstrument->UnderlyingInstrumentRef!=pUpdateMarginRate->UnderlyingInstrumentRef))
12     {
13         return false;
14     }
15     if ((pUpdateMarginRate->ExpireDate>0) && (pInstrument->ExpireDate!=pUpdateMarginRate->ExpireDate))
16     {
17         return false;
18     }
19     if ((pUpdateMarginRate->Multiple>0) && (pInstrument->Multiple!=pUpdateMarginRate->Multiple))
20     {
21         return false;
22     }
23     if ((pUpdateMarginRate->OptionTypeSet>0) && (((1<<pInstrument->OptionsType)&pUpdateMarginRate->OptionTypeSet)==0))
24     {
25         return false;
26     }
27     if ((pUpdateMarginRate->HedgeFlagSet>0) && (((1<<hedgeFlag)&pUpdateMarginRate->HedgeFlagSet)==0))
28     {
29         return false;
30     }
31     return true;
32 }

```

It is suggested that users of ydApi traverse positions to refresh the margin of each position. The pseudo-code is as follows:

```

1  virtual void notifyUpdateMarginRate(const YDUpdateMarginRate *pUpdateMarginRate)
2  {
3      for(auto pPosition : AllPositions)
4      {
5          if (applyToInstrument(pUpdateMarginRate, pPosition->pInstrument, pPosition->hedgeFlag))
6          {
7              // Using the investor's own margin refresh method
8              userDefinedUpdateMargin(pPosition, pUpdateMarginRate);
9          }
10     }
11 }

```

3.2.3.18 Account level information

The account level information provides exchange-level, product-level and instrument-level margin rates, commission rates, rights and traditional risk control parameters, which is a structure for investors to obtain these parameters. The following shows the description of these types of information:

- Commission rates are static information, while margins are dynamic data, which can be adjusted during trading. Therefore, the preday margins, commission rates and margin rates obtained through notifyFinishInit are instrument level information. For the sake of easy operation, API provides a method to directly obtain the instrument level margin rates and commission rates. Refer to [Margin Model](#) and [Commission](#) for details;

- Rights are dynamic data, which can be adjusted during trading. Margin parameters are set at all levels, however, they must be used at the instrument level. The API does not directly provide a method to query instrument level rights, which can be used automatically according to the method mentioned in [Trading Right](#);
- Traditional risk control parameters are static. The risk control parameters of YD can be set at the product and instrument levels. For example, the cancellation volume set at the product level means that the total all instrument-based cancellation volume involving the product is limited. Therefore, the risk control parameters should be obtained from the product and instrument levels, respectively. Refer to [Risk Control Parameters](#) for details.

The information at all levels regarding an account can be obtained directly through the following functions:

```

1  /// when trader call following 3 functions, pAccount should be NULL
2  virtual const YDAccountExchangeInfo *getAccountExchangeInfo(const YDExchange *pExchange, const YDAccount
   *pAccount=NULL)
3  virtual const YDAccountProductInfo *getAccountProductInfo(const YDProduct *pProduct, const YDAccount
   *pAccount=NULL)
4  virtual const YDAccountInstrumentInfo *getAccountInstrumentInfo(const YDInstrument *pInstrument, const
   YDAccount *pAccount=NULL)

```

The main fields of YDAccountExchangeInfo are as follows, and their primary keys are accounts and exchanges:

Field	Description
m_pAccount	Account structure pointer
m_pExchange	Exchange structure pointer
TradingRight	Trading right, dynamic data
IsDedicatedConnectionID	Dedicated seat array bitmap, refer to Designated Seat for details
TradingCode[YD_MaxHedgeFlag]	Array of trading codes, trading codes under different hedge flag can be obtained.

The main fields of "YDAccountProductInfo" are as follows, and their primary keys are accounts and products:

Field	Description
m_pAccount	Account structure pointer
m_pExchange	Exchange structure pointer
TradingRight	Trading right, dynamic data
TradingConstraints[YD_MaxHedgeFlag]	Traditional risk control parameter array for speculation, hedge and spread, indicating that the risk control indicators of all instruments regarding the product are added and then controlled. Refer to Risk Control Parameters for details

The main fields of "YDAccountInstrumentInfo" are as follows, and their primary keys are accounts and instruments:

Field	Description
m_pAccount	Account structure pointer
m_pExchange	Exchange structure pointer
TradingRight	Trading right, dynamic data
TradingConstraints[YD_MaxHedgeFlag]	Traditional risk control parameter array for speculation, hedge and spread, indicating that only the thresholds of the risk control indicators regarding an instrument is limited. Refer to Risk Control Parameters for details
m_pMarginRate[YD_MaxHedgeFlag]	Margin rate array, refer to Margin for details
m_pCommissionRate[YD_MaxHedgeFlag]	Commission rate array, refer to Commission for details
RFQCount	RFQ order count, only includes RFQ Order of options for Shanghai Futures Exchange and China Energy Exchange.
m_pCashCommissionRate[2]	Cash Commission Rate, m_pCashCommissionRate[0] is buy cash commission rate, m_pCashCommissionRate[1] is sell cash commission rate
m_pBrokerageFeeRate[2]	Brokerage fee rate, m_pBrokerageFeeRate[0] is buy brokerage fee rate, m_pBrokerageFeeRate[1] is sell brokerage fee rate.
ExemptMessageCommissionYDOrderFlag	If the YDOrderFlag of the order is greater than this flag, the information declaration fee will not be calculated.
MaxMessage	The maximum information limit is the upper limit, and exceeding this limit will prohibit order placement on this instrument.

Investors who use YDExtendedApi can also obtain more detailed account information through the following methods. The input parameters of these methods are the pointers to the various levels obtained above.

```

1 | virtual const YDExtendedAccountExchangeInfo *getExtendedAccountExchangeInfo(const YDAccountExchangeInfo
   | *pAccountExchangeInfo)
2 | virtual const YDExtendedAccountProductInfo *getExtendedAccountProductInfo(const YDAccountProductInfo
   | *pAccountProductInfo)
3 | virtual const YDExtendedAccountInstrumentInfo *getExtendedAccountInstrumentInfo(const
   | YDAccountInstrumentInfo *pAccountInstrumentInfo)

```

Compared with YDAccountExchangeInfo, the additional information provided by YDExtendedAccountExchangeInfo is as follows:

Field	Description
OptionLongPositionCost	Buy amount (total cost of long option position)
OptionLongPositionCostLimit	Buy amount limit(total cost limit for long option positions)
TradeControlFlag	Spot trading investor's control flag

Compared with YDAccountProductInfo, the additional information provided by YDExtendedAccountProductInfo is as follows:

Field	Description
MarginModelID	Margin Mode ID

Compared with YDAccountInstrumentInfo, the additional information provided by YDExtendedAccountInstrumentInfo is as follows:

Field	Description
MessageCommission	Current message commission
CashTradingConstraint	Bitmap of cash trading constraint: 0: prohibit buying 1: prohibit selling
MaxCashBuyVolume	Maximum cash buy volume
MaxOrderVolume	Maximum order volume
HoldingLimit	Holding limit

3.2.3.19 Combination margin parameters

In the portfolio margin model, the combination margin parameters are derived from exchange's preday data and cannot be modified during trading hours. To accommodate combination margin parameters from all exchanges, YD adopts a universal expression format using key-value pairs, where the meaning of each key-value pair is parsed by respective combination margin models. For more information about the portfolio margin model, please refer to [Portfolio Margin Model](#).

YD provides the following method to traverse all margin models:

```

1 | virtual int getMarginModelParamCount(void);
2 | virtual const YDMarginModelParam *getMarginModelParam(int pos);

```

The structure of YDMarginModelParam is as follows:

Field	Description
MarginModelID	Margin Model ID
ParamName	Parameter Name
ParamValue	Parameter Value

Taking the CZCE SPBM parameters as an example, the returned margin parameters are shown in the table below. The margin model ID for CZCE SPBM is 1.

MarginModelID	ParamName	ParamValue
1	MA.intraRateY	0.7
1	MA.fut.cvf	10
1	MA.fut.MA211.price	2603
1	MA.fut.MA211.marginRate	0.2
1	MA.fut.MA211.timeRange	SPOT
1	MA.fut.MA211.lockRateX	0.2
1	MA.fut.MA211.addOnRate	0

The above data is equivalent to the information in the original file issued by CZCE, as shown below:

```

1 | <?xml version="1.0" encoding="utf-8"?>

```

```

2  <spbmFile>
3      <version>1.00</version>
4      <exchange>ZCE</exchange>
5      <exchangeName>China Zhengzhou Commodity Exchange</exchangeName>
6      <createdDate>20221101</createdDate>
7      <createTime>150229</createTime>
8      <productFamily>
9          <productFamilyCode>MA</productFamilyCode>
10         <intraRateY>70.00</intraRateY>
11         <futPf>
12             <pfCode>MA</pfCode>
13             <pfName>Methanol futures</pfName>
14             <cvf>10</cvf>
15             <fut>
16                 <month>202211</month>
17                 <futCode>MA211</futCode>
18                 <price>2603.00</price>
19                 <marginRate>20.00</marginRate>
20                 <timeRange>SPOT</timeRange>
21                 <lockRateX>20.00</lockRateX>
22                 <addOnRate>0.00</addOnRate>
23             </fut>
24         </futPf>
25     </productFamily>
26 </spbmFile>

```

3.2.3.20 Account combination margin parameters

In the portfolio margin model, you can obtain the parameters of an investor on a specific margin model using the following method.

```

1  virtual const YDAccountMarginModelInfo *getAccountMarginModelInfo(int marginModelID, const YDAccount
    *pAccount=NULL)

```

The fields of the YDAccountMarginModelInfo structure are as follows:

Field	Description
AccountRef	Account number. obtained from YDAccount.
MarginModelID	Margin Model ID
CloseVerify	Whether to check usable funds when closing positions.
MarginRatio	Margin Ratio
ProductRange	Applicable product range. Separated by spaces, it represents the subset of products that are applicable to the corresponding portfolio margin model. An empty value indicates no restriction on the product range, consistent with the portfolio margin model.

If there are changes in 'CloseVerify' or 'MarginRatio' during trading hours, investors will be notified through the following callback function. Please note that 'ProductRange' cannot be modified during trading hours.

```

1  virtual void notifyAccountMarginModelInfo(const YDAccountMarginModelInfo *pAccountMarginModelInfo)

```

For more information about the portfolio margin model, please refer to [Portfolio Margin Model](#).

3.2.3.21 Risk control parameters

For the YD system, two types of risk control parameters can be provided according to technical implementation, which are traditional risk control parameters and general risk control parameters.

- Traditional risk control parameters are subject to the risk control rules supported by YD in the early stage, which mainly involve the control of common open position volume, trade volume, position volume and cancellation counts. Traditional risk control parameters do not increase continuously. Parameters subject to the new risk control rules are all set in the General Risk Control Parameters;
- General risk control parameters are introduced to adapt to the increasingly flexible and complex risk control rules. Their setting structure aims to general settings, which should be interpreted according to each risk control rule. Unlike those final values mentioned in [Account Level Information](#), general risk control parameters do not merge with those risk control parameters set at different dimensions, and therefore investors should merge them according to the levels supported by each risk control rule.

The fields of the traditional risk control parameter structure are shown in the following table. Combined with the "TradingConstraints[YD_MaxHedgeFlag]" of "YDAccountProductInfo" and "YDAccountInstrumentInfo", complete risk control parameters under different hedge flags can be obtained. Refer to the specific risk control rules mentioned in [Risk Control](#) for the detailed calculation method based on the risk control rules.

Field	Description
OpenLimit	Open volume limit
DirectionOpenLimit[2]	Buy/sell open volume limit
PositionLimit	Position limit

Field	Description
DirectionPositionLimit[2]	Long/short position limit
TradeVolumeLimit	Trade volume limit
CancelLimit	Order cancellation limit

Parameters based on general risk control rules can be obtained through the following traversing method:

```
1 virtual int getGeneralRiskParamCount(void);
2 virtual const YDGeneralRiskParam *getGeneralRiskParam(int pos);
```

The sample code for traversing all parameters based general risk control rules is as follows:

```
1 for(int i = 0; i < pApi->getGeneralRiskParamCount(); i++) {
2     YDGeneralRiskParam *param = pApi->getGeneralRiskParam(i);
3 }
```

The general risk control parameters from "YDGeneralRiskParam" are static data, their structure is as follows. The primary keys are (GeneralRiskParamType, AccountRef, ExtendedRef):

Field	Description
GeneralRiskParamType	Type of risk control rules
AccountRef	Account reference No., -1 means effective for all users, and other values mean effective for this user
ExtendedRef	Extended reference No., which can represent the serial numbers of exchanges, products and instruments according to the rule definition
IntValue1	Integer value 1
IntValue2	Integer value 2
FloatValue	Floating-point value

Not all the risk control rules involve all the above "IntValue1", "IntValue2" and "FloatValue". Most of the risk control rules only involve some of the parameters regarding the above rules. Refer to the specific risk control rules for details. Those parameter values not listed in the risk control rules mean that they are not used.

3.2.4 Receive dynamic data

After the YD OMS has completed the transmission of static data, the OMS port is ready to receive orders. If order submission starts at this time, the OMS will normally push dynamic data such as notifications. However, if the dynamic data is not sent completely to the client during an offline period of the client through the OMS, which means that the client's trading basis at this time may be wrong. In order to notify investors that all historical dynamic data have been received and a normal trading can be started, YD added this stage and notified investors through the callback function "notifyCaughtUp".

```
1 virtual void notifyCaughtUp(void)
```

The basic principle is that the OMS can show the investor the maximum SN when logging in. The API can be used to compare the SN of the current dynamic data with the maximum trading flow SN when logging in. If the current trading flow SN exceeds the maximum one when logging in, then the catching-up of the trading flow can be considered as completed. The callback of this method only indicates that it has caught up with the flow generated when logging in. In fact, there may be a trading flow generated after logging in. This message can be obtained again after disconnection and then reconnection, refer to [Trading Reconnection](#) for details.

The dynamic data collected during this stage is the same as that collected during normal trading, so the callback functions for collecting dynamic data will not be enumerated here. Refer to the description of callback functions in the corresponding operations.

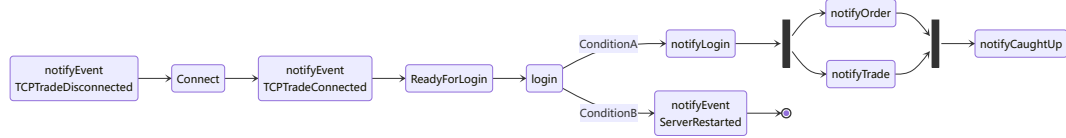
3.3 Reconnection

3.3.1 Trading reconnection

When detecting that the TCP trading connection with the OMS is interrupted or timed out, the API will call back "notifyEvent" (YD_AE_TCPTTradeDisconnected) to notify the trading thread that it has disconnected from the OMS. Then the API will go on to initiate the connection to the OMS until the reconnection is made, and the API will call back a "notifyEvent" (YD_AE_TCPTTradeConnected) notification. The subsequent process is similar to that for [Start](#) except the following differences:

- The static data will not be retransmitted during reconnection, because no "notifyFinishInit" callback will be made. After logging in, the orders and trading flow generated during disconnection will be transmitted;
- During logging in, it is not allowed to use another username. During calling login, the API can be used to determine whether to go to the next step or terminate the API depending on different factors. Refer to the following for the specific logic;
 - Check the data fingerprint of the OMS for being changed. If changed, enter the termination logic operation (ConditionB) directly. The data fingerprint of the OMS is calculated overall based on the preday data and the configuration information of YD. Uploading incorrect preday data, uploading new trading day data and restarting the OMS, and changing technical parameters such as exchange or seat configured in the OMS and restarting the OMS are common reasons for the data fingerprint change of the OMS.

- If the data fingerprint of the OMS does not change, the instance ID should be checked. The instance ID varies after the OMS is started each time, so it can be used to check whether the OMS has been restarted. The following three possibilities should be considered when checking whether the OMS is restarted and whether the HA function of the API is enabled;
 - If the OMS is not restarted, it means that the disconnection is caused by a network problem, and the system will go on to conduct the subsequent steps normally (ConditionA);
 - If the OMS is restarted and the HA function of API is enabled, the subsequent steps will be conducted normally (ConditionA). At this time, the notifyEvent (YD_AE_ServerSwitch) callback will be received before notifyLogin, indicating that a main /standby server switching has been conducted.
 - If the HA function of the API is not enabled, it will directly enter the termination logic operation (ConditionB)
- After entering the termination logic operation, generally, the API will call the "exit()" system call to exit the entire process, which, together with the strategy program, will exit the system. In order to prevent this, the API can be destroyed actively through [Destruction](#) under notifyEvent(YD_AE_ServerRestarted) to skip over calling "Exit" by the API.



In summary, when the OMS is restarted within the same trading day, YD API can provide investors with three different levels of processing methods:

- Restarting the process: This method is the cleanest, simplest and almost error-free in operation. The strategy program can receive trading flows from the OMS from the beginning again and restores to the latest state. The disadvantage is that the strategy program is liable to exit jointly, which may show a relatively low operability in production.
- Re-establishing API instances: This method is relatively clean and will not cause the policy program to exit. After instances are re-established, the API will push all trading flows again. If the strategy program aims to save and reuse the local trading flows, it may need to merge the trading flows retransmitted by the API, and the strategy program should ensure that the access to the destroyed instance data is prevented during the re-establishment, otherwise the program will crash, so the strategy program should be controlled carefully. During the re-establishment of the instances, some unknown timeout orders at the strategy program end and those orders from an external system may need to be handled. Refer to [High Availability](#) for details.
- Re-establishing the connection: Namely enabling the connection under the HA mode that has little impact on the strategy program. When an automatic switching is made, just notify the policy program of the HA switching through notifyEvent(YD_AE_ServerSwitch). Considering that the strategy program usually has the ability to handle unknown timeout orders at the strategy end and those orders from an external system, this method is the most friendly to the strategy program. Refer to [High Availability](#) for details under this mode.

3.3.1.1 High availability

In order to meet the requirements of market makers and investors attaching importance to availability, YD launched an HA cluster function, which can ensure the maximum business continuity. In cases of hardware crashes, program errors and other failures of the OMS, investors' trading can recover within the shortest period of time.

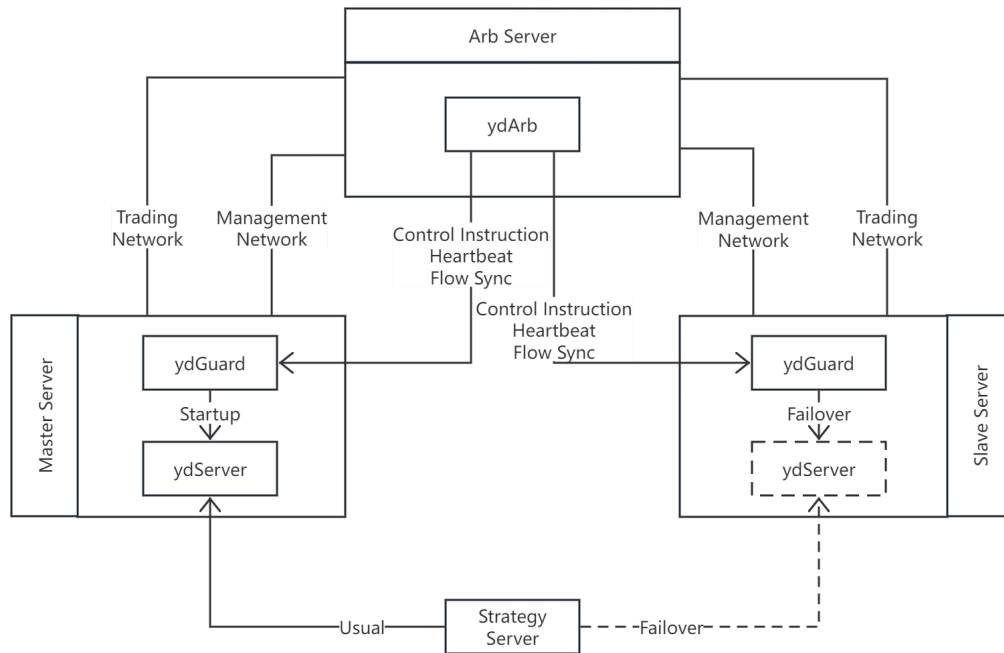
The HA cluster is a hot standby one composed of three servers, including two OMS servers running ydServer and one arbitration server running ydArb. In order to meet the balance between performance and availability of different users, the hardware configuration of the main/standby OMS servers can be subject to different combinations:

- Highest protection: Common main and standby servers are used to ensure that failures of the main server can be prevented as much as possible. Even if a failure occurs, to the utmost extent, the standby server can also be successfully started and used to take over the trading. For the current version, the look-through performance of common servers after performance optimization is 500ns worse than that of the overclocking machines, which is more suitable for market makers, broker-dealers and other users who extremely attach importance to system stability and trading availability.
- Performance maintenance and protection: The main and standby servers are an overclocking machine and a common server, respectively. Considering that current overclocking machines are relatively stable and usually do not fail, most of the orders are handled with overclocking machines relying on their highest performance. Once an overclocking machine fails, since the common server used by the standby machine is extremely stable, the standby server, to the utmost extent, can also be successfully started and used to take over the trading. This method attaches importance to both performance and availability. The performance, though being relatively low, can help to ensure the trading continuity in case of server switching.
- Highest performance: Both the main and standby servers are overclocking machines to ensure the trading performance to the utmost extent. However, since the shelf time and aging time for the main server are the same as those for the standby server, the standby server will not be started sometimes in case of main/standby switching. However, the stability of overclocking machines has been greatly enhanced, so almost no failure can be caused. This configuration combination is suitable for most investors.

An arbitration network should be established for all three servers in the high-availability (HA) cluster to transmit heartbeat messages, trading flows and start/stop control commands. The arbitration network only needs to ensure the arbiter is connected to main/standby servers, while the connectivity between main/standby servers is not required. Based on about 600MB per connection a day, a general gigabit network can meet the bandwidth requirements of the arbitration network. The network management system can be used as the arbitration network if the capacity of its system meets the bandwidth requirements. It is suggested to configure the investment trading network as a backup for the arbitration network to prevent invalid switching errors due to arbitration network failure. All trading flows are concentrated on the arbitration network when it is running smoothly, while there is no trading flow on the trading network. The trading flows will be switched to the trading network when the arbitration network fails. If the arbiter cannot be connected to the main server via the arbitration network, it will try to contact the main server through the standby network. If the connection fails again, it is more feasible to switch from the main server to the standby one as investors are most likely unable to connect to the OMS at this time.

The main component functions of the HA cluster are as follows:

- **ydArb**: YdArb: A perpetual arbitration program used for arbiters. The availability of the cluster can be monitored through heartbeat messages. When the main site fails, the standby site will be actively notified to start to replace the main site, and the trading data (trading flows of an exchange and local OMS) will be transmitted continuously from the main site to the standby site to ensure that the standby site can start up as quickly as possible.
- **ydGuard**: A perpetual daemon of ydServer used for main and standby servers. It is used for detecting the status of ydServer and reporting it to ydArb. The main server can transmit the trading data from ydGuard to ydArb, while the ydGuard of the standby OMS can receive the trading data from ydArb. The ydGuard data transmission of the main server is asynchronous with the trading through the ydServer. It is very likely that a notification has been sent to investors but not synchronized to the standby OMS.



Before connecting to an HA cluster, the API configuration files should be modified according to the following example.

```

1 RecoverySiteCount=2
2 TradingServerIP=<ip of master host>
3 TradingServerPort=<port of master host>
4 TradingServerIP2=<ip of slave host>
5 TradingServerPort2=<port of slave host>

```

During normal operation, the ydServer process of the main server is enabled, while the ydServer process of the standby server is stopped. The cluster composed of ydArb and ydGuard continuously transmits the trading flow from the main server to the standby OMS through the arbiter in real-time. At this time, the investor's API is connected to the ydServer of the main server for trading.

In case of main/standby server switching, the ydServer of the standby server will be started, and the synchronized trading data will be loaded to recover to the state before the main server fails. After YD API detects a disconnection, it will poll the IP addresses of the main and standby servers until a reconnection has been made. For the details of the subsequent steps after reconnection, refer to [Trading Reconnection](#) for more details. Due to the inherent complexity for main /standby server switching, the following problems are inevitable and should be handled by investors:

- **Local unknown timeout order**: If the OMS fails after an order is sent by the strategy program and before it is sent by the OMS to an exchange. The strategy program will always wait for the notification to update the order status, however, since the order is not submitted to the exchange, for the OMS and the exchange, this order does not exist, and of course it is impossible to cancel this order. Therefore, the strategy program should be established with a timeout mechanism. In case of main/standby server switching, those local timeout orders at the strategy end should be canceled actively. Please note that the causes and consequences regarding local timeout orders are different from those regarding the [Unknown Timeout Order](#), and accordingly the processing methods are also different.
- **Failing in determination of relationship between an order and the corresponding trade**: If the OMS fails after an order is sent by the OMS and before the trading flow of the main server is synchronized to the standby one, it is obvious that after the failure, the strategy program has not received the order notification from the corresponding exchange (otherwise, there will be no missed orders for special treatment). After the standby OMS is started, YD will ensure that no order missing will be caused, however, since the local trading flow of the main server has not been synchronized, the order notification re-sent by the standby OMS will contain OrderSysID, OrderLocalID and RealConnectionID except for OrderRef. After receiving this notification, the investor cannot match this order with that recorded by the strategy program end. When OrderRef is -1, the order can be handled according to the common external order submission logic, while if the order fails to match that of the strategy program end, it should be handled according to the local unknown timeout order processing logic.

The single-host HA mode is a special case for the dual-host HA mode, which means making two IP addresses be the same under the dual-host HA mode. Their actual effects are basically the same. The single-host HA mode simplifies the handling process when the strategy program and the OMS are restarted within the same trading day, and therefore investors are suggested to choose the single-host HA mode as much as possible. The configuration example of the corresponding configuration files for the single-host HA mode is as follows:

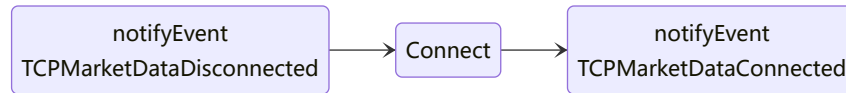
```

1 RecoverySiteCount=1
2 TradingServerIP=<ip of master host>
3 TradingServerPort=<port of master host>

```

3.3.2 Market data reconnection

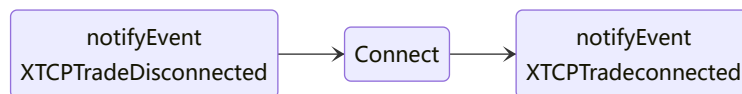
When the API detects that the TCP market data connection with the OMS is interrupted or timed out, the API will call back the `notifyEvent(YD_AE_TCPMarketDataDisconnected)` to notify of the disconnection between the market data thread and the OMS, then the API will continue to initiate the connection to the OMS and call back the `notifyEvent(YD_AE_TCPMarketDataConnected)` notification after reconnecting. Compared with the process of trading reconnection, market data reconnection is much simpler. After the reconnection, re-subscription to the market data will be unnecessary since the API will automatically re-subscribe to the instrument subscribed before the disconnection and continue to collect the TCP market data.



3.3.3 XTCP reconnection

When the API detects that the XTCP trading connection with the OMS is interrupted or timed out, the API will call back the `notifyEvent(YD_AE_XTCPTradeDisconnected)` to notify of the disconnection between the XTCP thread and the OMS, then the API will continue to initiate the connection to the OMS and call back the `notifyEvent(YD_AE_XTCPTradeConnected)` notification after reconnecting.

If investors order during reconnection after disconnection, the API will downgrade to choose the normal TCP order submission mode.



3.4 Destruction

The strategy program can be used for actively destroying API instances to avoid memory leakage or API's exiting the process. Since the structure of YD API is relatively complex and cannot be cleaned up by deleting instances, YD provides `startDestroy` in `ydApi` for instance cleaning. Please never try to delete API instances directly.

```

1 virtual void startDestroy(void)

```

The `startDestroy` initiates a two-stage asynchronous destruction process. The notification regarding this function does not mean a complete cleanup.

- The main task in the first stage is to try to disable all connections and threads regarding API, and then the completion of which can be notified through "YDListener.notifyBeforeApiDestroy". In this process, all trading functions including the callback function of "notifyBeforeApiDestroy" will be disabled for API, and the trading callback function of YDListener will not be recalled. However, all query functions of `ydApi` and the data managed in `ydExtendedApi` can still be used.

```

1 virtual void notifyBeforeApiDestroy(void)

```

- The main task of the second stage is to destroy all data. and then the completion of cleaning will be notified through "YDListener.notifyAfterApiDestroy". In this process, all API functions including the callback function of `notifyAfterApiDestroy` and all pointers obtained from API will be disabled.

```

1 virtual void notifyAfterApiDestroy(void)

```

After the cleanup, please destroy the YDListener subclass instance you implemented and instantiated. This function does not destroy the instance for you. In the strategy program, you can re-create new API instances.

4 Fund

4.1 Equity

4.1.1 Pre balance

Pre balance, also known as pre-day equity, is mainly composed of pre equity. The pre balance calculation methods of futures exchanges and stock exchanges are different, the following will introduce respectively.

In order to facilitate brokers to configure and use the YD system and reduce unnecessary configuration errors, YD directly uses the pre-day documents of the existing primary connection system as the pre-day data of YD after conversion, which include pre balance, pre-day position, margin rate, commission rate and some risk control rules, etc. At present, the futures exchange pre-day data of YD can be read directly from CTP syncmerge or CTP heterogeneous data interface, while the stock option pre-day data of stock exchanges is obtained from different data sources. In order to meet the demand and ensure the checking relationship of different data sources, YD has introduced some internal calculation methods. If the primary OMS used by brokers is of other brands, YD will continue to add new pre-day data conversion programs to meet the unrestricted demand of investors to participate in trading in brokers' sub-positions.

Pre balance can be obtained through "YDAccount.PreBalance".

4.1.1.1 Pre balance of futures exchange

The data of futures exchanges is simply calculated after reading from CTP syncmerge or CTP heterogeneous data interface. The calculation formula is as follows: Where, pre equity, pledged amount, currency pledge-in and pledge-out amounts are directly read from the source data, and fixed deduction is set in the YD system.

$$\text{PreBalance} = \text{PreEquity} + \text{Pledged} + \text{CurrencyPledgeIn} - \text{CurrencyPledgeOut} - \text{FixedDeduction}_{\text{YD}}$$

At present, in the market, brokers and investors always sign agreements to agree on fixed equities of each secondary OMS. Brokers modify pre-day data for fund splitting. This approach is simple, but the opportunity brought by a market quote of an exchange is always hard to seize due to lack of funds, even if the funds can be allocated through depositing/withdrawing, the complex review of manual operation always cause missing of trading opportunities. Therefore, YD considered this at the beginning of the design. When investors trade through different YD OMSs at the same time, it is suggested that the brokers should not split the investors' pre-equities, so that the pre-equities read on each OMS can be the same, and then the proportions of investors' pre-equities can be controlled through each OMS based on maximum money usage parameters. Under the help of YD's fund transfer manager, when the sum of the maximum money usage of the same investor for all OMSs is less than or equal to 100%, the investor can modify the maximum money usage at its own discretion in order to achieve the effect of real-time fund transfer. For the meaning of maximum money usage, refer to [Today Balance](#) for more details.

Fixed deduction refers to a function designed by YD that can allow investors use maximum money usage on the OMS and other secondary OMSs for trading at the same time. As shown in the following table, assuming that an investor has a 1 million-equity, 800,000 is allocated on the OMS, and 200,000, allocated on other secondary OMSs, then after a fixed deduction, the maximum money usage function can still work normally.

Exchange	Type of OMS	Total equity of different OMSs	maximum money usage	Equity proportion
SHFE	YD	80	40%	32
DCE	YD	80	50%	40
CFEX	YD	80	10%	8
CZCE	Other	20		20

4.1.1.2 Pre balance of stock exchange

Pre-equities of investors in the stock markets of Shanghai and Shenzhen can be directly read by stock exchanges and futures exchanges and then dynamically split based on the fund use proportion, which is a practice recommended by YD. In order to meet the practical and actual needs of investors in current markets, YD also supports splitting based on a proportion of usable funds set at the level of each investor (hereinafter referred to as the split proportion, which is defined in the background and cannot be read on API). This function can be used by calculating data from the three data sources namely cfmmc reported documents (from the monitoring center, only data of the stock markets of Shanghai and Shenzhen are involved), CSDC Shanghai data and CSDC Shenzhen data. The whole calculation process is relatively complex. Although it is hard for YD to guarantee that the usable funds displayed on the OMS after splitting based on the proportions are the same as the results calculated based on the splitting proportion shown on the primary OMS, YD can guarantee that the sum of the pre-equities in the stock markets of Shanghai and Shenzhen is always equal to the pre equity shown on the primary OMS. Just read the following for the specific calculation method of YD. If you are not interested in it, just skip it directly.

On the exercise delivery date, due to the difference in the checking relationship among cfmmc reported documents (from the monitoring center, only data of the stock markets of Shanghai and Shenzhen are involved), CSDC Shanghai data and CSDC Shenzhen data, the usable funds applicable to the YD system should be calculated first and used as the splitting basis.

$$\text{Usable}_{\text{YD}} = \text{PreEquity}_{\text{cfmmc}} - \text{SSEMargin}_{\text{cfmmc}} - \text{SZSEMargin}_{\text{cfmmc}} + \text{NetExerciseAllocationFrozen}_{\text{CSDCSH}} + \text{NetExerciseAllocationFrozenFund}_{\text{CSDCSZ}}$$

Then the pre equity of YD system can be obtained by splitting the calculated available fund.

$$\text{SSEPreEquity}_{\text{YD}} = \text{SSESplittingProp}_{\text{YD}} \times \text{Available}_{\text{YD}} + \text{SSEMarg}_{\text{cfmmc}} - \text{NetExerciseAllocationFrozenFund}_{\text{CSDCSH}}$$

$$\text{SZSEPreEquity}_{\text{YD}} = \text{SZSESplittingProp}_{\text{YD}} \times \text{Available}_{\text{YD}} + \text{SZSEMarg}_{\text{cfmmc}} - \text{NetExerciseAllocationFrozenFund}_{\text{CSDCSZ}}$$

After a simple combination of the three formulas, it can be seen that the total of pre-equities in the YD system is equal to that in the document of the monitoring center.

$$\text{pre equity}_{\text{cfmmc}} = \text{SSZE pre equity}_{\text{YD}} + \text{SZSE pre equity}_{\text{YD}}$$

The fixed deduction function, just like that of futures exchanges, is still effective, however, if other secondary OMS systems support proportion-based available fund splitting, the fixed deduction function will be unavailable, so it is almost meaningless in the whole process. Generally, the fixed deduction value is 0.

$$\text{pre balance} = \text{pre equity} - \text{fixed deduction}_{\text{YD}}$$

4.1.2 Today balance

In the design of YD, the static equities refer to the overall equity of the day at the investor's level. The static equities of an investor on any YD OMS are the same, and the deposit / withdrawal amounts are also at the investor's level, that is to say, as long as the investor conducts depositing / withdrawing operations through the primary OMS, the corresponding deposit / withdrawal amounts of this investor can be shown on any YD OMS and the amounts are the same as those shown on the primary OMS. Thus, the concept introduction of maximum money usage can be allowed.

If a broker splits its fund based on a fixed amount or an available fund proportion, the deposit/withdrawal amounts can be expressed as those for single OMS. The static equity only represents the investor's equity of that very day shown on the corresponding OMS, and in this case, the maximum money usage should not be used, namely the maximum money usage should be kept at 100%, otherwise a fund confusion will be caused.

Investors using `ydExtendedApi` can call "`YDExtendedAccount.staticCashBalance()`" to obtain static equities. Please refer to the calculation method of net deposit and withdrawal amount: [Net deposit and withdrawal](#)

$$\text{StaticEquity} = \text{PreBalance} + \text{Net deposit and withdrawal}$$

Regardless of the operation mode, the current balance represents the current investor's equity shown on the corresponding OMS at the very day. If you use the OMS according to the original design of YD, the maximum money usage can be adopted to allow automatic deposit/withdrawal synchronization and investors' independent fund transfer.

Investors using `ydExtendedApi` can obtain the current balance through "`YDExtendedAccount.Balance`".

$$\text{TodayBalance} = (\text{StaticEquity} - \text{FrozenWithdrawal}) \times \text{MaxMoneyUsage}$$

4.1.2.1 Maximum money usage

YD supports splitting the pre-day equity of different systems by setting maximum money usage. As long as the total money usage of each OMS does not exceed 100%, the margin will not be overdrawn. The maximum money usage can be obtained from "`YDAccount.MaxMoneyUsage`".

When brokers do not split the funds and reasonably set the maximum money usage for different YD OMSs, investors can use the automatic depositing/withdrawing and independent transfer functions, see the following example.

For example, an investor trades through three YD systems in CFFEX, DCE and SHFE at the same time. Its static equity is CNY 1 million and the maximum money usage are 50%, 30% and 20%, respectively. For simplicity, the impact of margin is ignored in the example, and those cases such as withdrawal amount and set maximum money usage causing insufficient fund will be intercepted by the system, therefore, the following, by default, do not trigger the limits.

	CFFEX	DCE	SHFE
Initial maximum money usage	50%	30%	20%
Current balance	50	30	20
Current balance after a deposit of CNY 1 million	100	60	40
Emergency fund transfer required in SHFE and set maximum money usage	10%	20%	70%
Current balance after transfer	20	40	140
Current balance after a withdrawal of CNY 500,000	15	30	105

The fund synchronization system (`ydSync`) can be used for reading the deposit/withdrawal flow of the CTP risk control system in real time and then synchronize it to all YD OMSs; The fund transfer manager (`ydFundManager`) allow the fund transfer of different YD OMSs. Brokers can issue transfer accounts for investors so that they can log in to the fund transfer manager to transfer funds. If necessary, just contact the brokers for installation, however the prerequisite for the above functions is that brokers have not split the fund.

4.1.3 Other equities

In order to finally calculate the fund and market value equities, YD calls the trading-related part of the fund and market value equity formulas as dynamic equity. The dynamic equity is only related to the exchange where the OMS belongs to. The dynamic equities of two OMSs with management seats arranged in the same exchange are the same for the same investor. However, if one of the above two OMSs is configured with an all non-management seats, it cannot receive the trading data from other OMSs. At this time, only the dynamic equity of the OMS with a management seat is accurate. Of course, if this exchange has only one OMS, its dynamic equity will always be accurate regardless of the used seat. The formula for calculating the dynamic equity of a single OMS is as follows:

$$\text{DynamicEquity} = \text{CloseProfitAndLoss} + \text{FuturePositionProfitAndLoss} + \text{CashIn} - \text{Commission}$$

Under the concept of dynamic equity, the formulas for calculating the fund and market value equities of a single OMS are as follows. Based on the above reasons, the fund and market value equities of a single OMS are also simple and addible:

$$\text{TodayFundEquity} = \text{StaticEquity} \times \text{MaxMoneyUsage} + \text{DynamicEquity}$$

$$\text{PreFundEquity} = \text{PreBalance} \times \text{MaxMoneyUsage}$$

$$\text{TodayMarketValueEquity} = \text{TodayFundEquity} + \text{TodayOptionMarketValue}$$

$$\text{PreMarketValueEquity} = \text{PreFundEquity} + \text{PreOptionMarketValue}$$

The above equities can only be obtained through `ydExtendedApi`:

- Dynamic equity: Obtained through calling "`YDExtendedAccount.dynamicCashBalance()`"
- Today fund equity: Obtained through calling "`YDExtendedAccount.cashBalance()`"
- Pre fund equity: Obtained through calling "`YDExtendedAccount.preCashBalance()`"
- Today market value equity: Obtained through calling "`YDExtendedAccount.marketValue()`"
- Pre market value equity: Obtained through calling "`YDExtendedAccount.preMarketValue()`"

4.2 Usable

$$\text{Usable} = \text{TodayBalance} + \text{CloseProfitAndLoss} + \min(\text{FuturesPositionProfitAndLoss}, 0) + \text{CashIn} - \text{Commission} - \text{FuturesMargin} - \text{SellOptionMargin} - \text{OptionExerciseMargin} - \text{NetExerciseFrozen}$$

Investors using `ydExtendedApi` can call "`YDExtendedAccount.usable()`" to obtain usable funds. Regulatory regulations require that position losses be deducted, so the YD OMS deducted position losses when calculating available funds. After the promulgation of the Futures Law, the requirement that position profits are not included in available funds was deleted. Local securities regulatory bureaus also have different regulations on this. Therefore, starting from version `1.386.40.38`, brokers can set a function switch for customers to include position profits in available funds. After turning it on, the investor's position profits will be included in available funds. Investors can determine whether position profits are included in available funds by checking whether `AccountFlag` in `YDAccount` is set to `YD_AF_UsePositionProfit`.

"`YDExtendedAccount.Available`" is similar to a usable fund. It can be seen from the following formula that `Available` does not include the position loss, so an `Available` cannot be considered as a usable fund.

$$\text{Available} = \text{TodayBalance} + \text{CloseProfitAndLoss} + \text{CashIn} - \text{Commission} - \text{FuturesMargin} - \text{SellOptionMargin} - \text{OptionExerciseMargin} - \text{NetExerciseFrozen}$$

Since the time and price settings for API and the OMS refreshing are not exactly the same, it may lead to the situation that the API shows sufficient funds, but the OMS refuses due to insufficient funds. Generally, this happens when the fund utilization rate is high. For confirming an order refused by the OMS due to insufficient funds, just contact the broker to obtain the fund information from the OMS log when the order is rejected. Please note that `Available` does not mean usable funds, the loss of "`PositionProfit`" should be deducted.

The latest usable funds and `Available` can only be displayed after refreshed and calculated according to the latest price. Refer to [Fund Refresh Mechanism](#) for the specific refresh method.

4.3 Net deposit and withdrawal

The calculation formula for net deposit and withdrawal is:

$$\text{Netdepositandwithdrawal} = \text{Cumulativedepositamount} - \text{Cumulativewithdrawalamount}$$

When an investor has deposits and withdrawals, he/she can obtain the accumulated deposit amount through "`YDAccount.Deposit`", as well as obtain the accumulated withdrawal amount through "`YDAccount.Withdraw`". "`YDAccount.Deposit`" and "`YDAccount.Withdraw`" are monotonically increasing within a trading day. The net deposit and withdrawal of the day can be obtained by "`YDAccount.Deposit - YDAccount.Withdraw`".

The frozen withdrawal amount can be obtained through "`YDAccount.FrozenWithdraw`". The frozen withdrawal amount will continue to accumulate as the broker freezes withdrawals. When the broker retreats the frozen withdrawals, the amount of "`YDAccount.FrozenWithdraw`" will reduce accordingly, or the broker can also deduct the frozen amount that is the same as the withdrawal amount when making a formal withdrawal.

When a deposit or withdrawal is frozen, the related investor will be notified through "`notifyAccount`". Since the investor will also be notified when other fields of "`YDAccount`" change, he/she should check the above three fields in the program for deposits or withdrawals.

```
1 | virtual void notifyAccount(const YDAccount *pAccount)
```

4.4 Cash Income and Expenditure

Cash income and expenditure can be read from `YDExtendedAccount.CashIn`. Option premiums and spot income and expenditure are both included in cash income and expenditure.

When option and spot buyers place orders, cash income and expenditure will be frozen. As orders are gradually traded, the order-frozen funds in cash income and expenditure will be gradually released, and accordingly the transaction funds deduction in cash income and expenditure will be increased. If the order is eventually cancelled, the order-frozen funds will be released at one time.

When option and spot sellers place orders, cash income and expenditure are not changed. As orders are gradually executed, the transaction funds are gradually added to cash income and expenditure. If the order is eventually cancelled, cash income and expenditure will not be changed.

4.4.1 Spot Order Income and Expenditure

The calculation formula for spot buy order income and expenditure is as follows. When the order is a market order, the order price is the upper limit price. When the order is a limit order/FAK/FOK, the order price is the order price. The bond price only has a value when the trading security is a bond, and other spot is 0:

$$\text{StockBuyOrderIncomeandExpenditure} = -(\text{OrderPrice} + \text{Bondinterest}) \times \text{ordervolume}$$

Stock sell order income and expenditure should always be 0.

4.4.2 Spot Trade income and expenditure

The calculation formula for spot buy trade income and expenditure is as follows:

$$\text{spotbuytradeincomeandexpenditure} = -(\text{transactionprice} + \text{bondinterest}) \times \text{transactionvolume}$$

The calculation formula for spot sell trade income and expenditure is as follows:

$$\text{spotselltradeincomeandexpenditure} = (\text{transactionprice} + \text{bondinterest}) \times \text{transactionvolume}$$

Bond prices are only valuable when the trading securities are bonds, and other spot prices are 0.

4.4.3 Option Order premium

The calculation formula for option buyers' order premium is:

$$\text{LongOrderPremium} = -\text{OrderPrice} \times \text{InstrumentMultiplier} \times \text{OrderVolume}$$

Where, when the order is a market price one, the upper limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail.

The calculation formula for option sellers' order premium is:

$$\text{ShortOrderPremium} = \text{OrderPrice} \times \text{InstrumentMultiplier} \times \text{OrderVolume}$$

Where, the option sell order price is controlled by parameters (Name, Target) namely (SellOrderPremiumBasePrice, Options) in "YDSystemParam". The following shows how to handle different parameters by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price;
- YD_CBT_OrderPrice: When the order is a market price one, the lower limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail;
- YD_CBT_None: No premium will be added, which is equal to the price of 0, namely the option seller will not get the premium when submitting orders. This is a default configuration and popular for mainstream systems.

4.4.4 Option Trade premium

The calculation formula for option buyers' trade premium is:

$$\text{LongTradePremium} = -\text{TradePrice} \times \text{InstrumentMultiplier} \times \text{TradeVolume}$$

The calculation formula for option sellers' trade premium is:

$$\text{ShortTradePremium} = \text{TradePrice} \times \text{InstrumentMultiplier} \times \text{TradeVolume}$$

4.4.5 Profit / loss of futures positions

$$\begin{aligned} \text{LongFuturePositionProfit/Loss} = & ((\text{LatestPrice} - \text{OpenPrice}) \times \text{TodayPositionVolume} \\ & + (\text{LatestPrice} - \text{PreSettlementPrice}) \times \text{PrePositionVolume}) \times \text{InstrumentMultiplier} \end{aligned}$$

$$\begin{aligned} \text{ShortFuturePositionProfit/Loss} = & ((\text{OpenPrice} - \text{LatestPrice}) \times \text{TodayPositionVolume} \\ & + (\text{PreSettlementPrice} - \text{LatestPrice}) \times \text{PrePositionVolume}) \times \text{InstrumentMultiplier} \end{aligned}$$

For the refresh logic of position profit / loss, see [Fund Refresh Mechanism](#).

4.5 Close profit/loss of futures

$$\text{ClosingTodayLongFuturePositionProfit/Loss} = (\text{ClosePrice} - \text{OpenPrice}) \times \text{InstrumentMultiplier} \times \text{ClosePositionLots}$$

$$\text{ClosingPreLongFuturePositionProfit/Loss} = (\text{ClosePrice} - \text{PreSettlement}) \times \text{InstrumentMultiplier} \times \text{ClosePositionLots}$$

$$\text{ClosingTodayShortFuturePositionProfit/Loss} = (\text{OpenPrice} - \text{ClosePrice}) \times \text{InstrumentMultiplier} \times \text{ClosePositionLots}$$

$$\text{ClosingPreShortFuturePositionProfit/Loss} = (\text{PreSettlement} - \text{ClosePrice}) \times \text{InstrumentMultiplier} \times \text{ClosePositionLots}$$

For the logic of close pair position details, refer to [Position Structure](#)..

4.6 Option market value

$$\begin{aligned} \text{PreOptionMarketValue} = & \text{LongOptionPositionVolume} \times \text{PreSettlementPrice} \times \text{InstrumentMultiplier} \\ & - \text{ShortOptionPositionVolume} \times \text{PreSettlementPrice} \times \text{InstrumentMultiplier} \end{aligned}$$

$$\begin{aligned} \text{TodayOptionMarketValue} = & \text{LongOptionPositionVolume} \times \text{LatestPrice} \times \text{InstrumentMultiplier} \\ & - \text{ShortOptionPositionVolume} \times \text{LatestPrice} \times \text{InstrumentMultiplier} \end{aligned}$$

The pre-option market value and current option market value can be obtained through "YDExtendedAccount.PrePositionMarketValue" and "YDExtendedAccount.PositionMarketValue", respectively.

The current option market value calculated according to the latest price can only be shown after refresh. Refer to [Fund Refresh Mechanism](#) for the specific refresh method.

4.7 Commission

Investors using ydExtendedApi can obtain the total commission of the account through YDExtendedAccount.Commission.

4.7.1 Cash Commission

The order handling fee is charged after the order is successfully placed, the order cancellation fee is charged after the order is successfully cancelled, and the transaction fee is charged when the transaction is completed. The maximum possible transaction fee is frozen in advance after the spot buy and sell orders of SSE and SZSE are sent. The difference in transaction fee will be unfreeze accordingly when all orders are traded or cancelled .

$$\begin{aligned} StampDuty &= \min(\max(\text{Volume} \times (\text{TradingPrice} \times \text{StampDutyByAmount} + \text{StampDutyByVolume}), \text{MinStampDutyValue}), \text{MaxStampDutyValue}) \\ SecuritiesManagementFee &= \min(\max(\text{Volume} \times (\text{TradingPrice} \times \text{SecuritiesManagementFeeByAmount} + \text{SecuritiesManagementFeeByVolume}), \text{MinSecuritiesManagementFeeValue}), \text{MaxSecuritiesManagementFeeValue}) \\ HandlingFee &= \min(\max(\text{Volume} \times (\text{TradingPrice} \times \text{HandlingFeeByAmount} + \text{HandlingFeeByVolume}), \text{MinHandlingFeeValue}), \text{MaxHandlingFeeValue}) \\ TransferFee &= \min(\max(\text{Volume} \times (\text{TradingPrice} \times \text{TransferFeeByAmount} + \text{TransferFeeByVolume}), \text{MinTransferFeeValue}), \text{MaxTransferFeeValue}) \\ BrokerageFee &= \min(\max(\text{Volume} \times (\text{TradingPrice} \times \text{BrokerageFeeByVolume} + \text{BrokerageFeeByAmount}), \text{MinBrokerageFeeValue}), \text{MaxBrokerageFeeValue}) \end{aligned}$$

For more information about commission rate, please refer to [Cash Commission Rate](#) and [Brokerage Fee Rate](#).

4.7.2 Derivative Commission

The order commission fee is charged after the order is successfully placed, the cancel commission fee is charged after the order is successfully cancelled, and the transaction fee is charged when the transaction is completed. The transaction fee is frozen in advance after the stock option buy open orders (SSE, SZSE) as well as open orders (other futures exchanges), the calculation formula of frozen transaction fee is same as open commission fee, that order volume is used as open volume. Order price varies depending on the order's type: for market orders, the upper limit price is used, while for non-market-price orders, the order price is used. Transaction fees are not frozen for closing orders.

$$\text{OrderCommission} = \text{OrderCount} \times \text{OrderCommByVolume}$$

$$\text{CancelCommission} = \text{CancelCount} \times \text{OrderActionCommByVolume}$$

$$\text{OpenCommission} = \text{OpenVolume} \times (\text{OpenRatioByMoney} \times \text{OrderPrice} \times \text{InstrumentMultiplier} + \text{OpenRatioByVolume})$$

$$\text{CloseTodayCommission} = \text{CloseTodayVolume} \times (\text{CloseTodayRatioByMoney} \times \text{TradePrice} \times \text{InstrumentMultiplier} + \text{CloseTodayRatioByVolume})$$

$$\text{CloseHistoryCommission} = \text{CloseHistoryVolume} \times (\text{CloseRatioByMoney} \times \text{TradePrice} \times \text{InstrumentMultiplier} + \text{CloseRatioByVolume})$$

$$\begin{aligned} \text{Commission} &= \sum_{\text{AllOrders}} \text{OrderCommission} + \sum_{\text{AllCancelOrders}} \text{CancelCommission} + \sum_{\text{AllOpenTrades}} \text{OpenCommission} \\ &+ \sum_{\text{AllCloseTodayTrades}} \text{CloseTodayCommission} + \sum_{\text{AllCloseHistoryTrades}} \text{CloseHistoryCommission} \end{aligned}$$

$$\begin{aligned} \text{ExerciseCommission} &= \text{ExerciseQuantity} \times (\text{ExerciseRatioByVolume} + \text{ExerciseRatioByMoney} \\ &\times \text{ExercisePrice} \times \text{OptionInstrumentMultiplier}) \end{aligned}$$

For information on how to obtain the commission rate, see [Commission Rate](#).

The commission calculation methods for position closing in different exchanges are different. YD maintains the pre position volume for commission calculation through "YDExtendedPosition.YDPositionForCommission". Investors can obtain the volume for today's position closing through "YDExtendedPosition.Position-YDExtendedPosition.YDPositionForCommission" for calculating the commission.

In order to determine whether the corresponding instrument supports a today's position closing priority or a pre position closing priority, investors can check whether the corresponding exchange regarding the instrument prefers today's position closing through "YDExchange.CloseTodayFirst".

The following is a brief description of the detailed today's position and pre position sequence each exchange when calculating the commission for position closing:

Please note that the position details for commission calculation, position closing profit/loss calculation and combined margin calculation are maintained separately, and therefore the position information should be properly selected according to the actual operation.

- For SHFE and INE, dependent on the today's position closing or pre position closing instructions;
- For CFFEX, DCE, GFEX and CZCE, today's position closing is preferred;
- For SSE and SZSE, pre position closing is preferred.

The SSE does not charge any commission for short order selling, which is independent of the set commission parameters.

4.7.3 Exercise commission rate

Exercise commission rate's calculation formula is:

$$\text{ExerciseCommission} = \text{ExerciseQuantity} \times (\text{ExerciseRatioByVolume} + \text{ExerciseRatioByMoney} \times \text{ExercisePrice} \times \text{OptionInstrumentMultiplier})$$

Obtaining exercise commission rate, please refer to [Commission Rate](#).

4.7.4 Derivatives message count commission

Currently, all products from SHFE and INE, as well as some products from DCE and CZCE, have commenced collecting commissions based on message count. YD supports the collection of message count commission according to the OTR and message count tiers. For specific fee standards, please refer to the relevant notices of each exchange.

In certain circumstances, the message count commission can be waived. For instance, RFQ orders from exchanges other than SHFE and INE are not counted towards the message count, and market makers are not charged an message count commission. Investors can inspect "YDAccountInstrumentInfo.ExemptMessageCommissionYDOrderFlag" and the "YDOrderFlag" of the order to determine whether the order is included in the message count. If "YDOrderFlag > YDAccountInstrumentInfo.ExemptMessageCommissionYDOrderFlag", it is not included; otherwise, it should be included.

To ensure the rationality of the OTR under extreme conditions while aligning with the regulatory formula, YD employs the following calculation formula for the OTR:

$$OTR = \frac{\max(\text{message count}, 1)}{\max(\text{number of successful orders}, 1)} - 1$$

In the above formula, the definition of a successful order is as follows: if an order is partially or fully filled, then this order is counted as one successful order. Multiple executions from one order are not repeatedly counted.

The method for calculating the message count is as follows:

- Limit order: If it's completely filled, only one order is counted. If it's cancelled, both one order and one cancel are counted.
- FAK/FOK order: If completely filled, only one order is counted. If not filled or not completely filled and a cancel is generated, both one order and one cancel are counted.
- Market order: If completely filled, only one order is counted. If not filled or not completely filled and a cancel is generated, both one order and one cancel are counted.
- RFQ order: For options RFQ orders from the SHFE and INE, one message count is added. For futures RFQ orders from the SHFE and RFQ orders from other exchanges, they are not counted.
- Combination (Arbitrage) order: the message count of each leg of a combination order is separately counted on each leg.
- Exchange forced liquidation orders and non-futures company forced liquidation orders: both are included in the message count.
- Forced reduction orders: are not included in the message count.
- Erroneous orders for limit orders, FAK orders, FOK orders, and market orders: are not included in the message count.
- Erroneous cancel orders for limit orders: are not included in the message count.
- Combination and decomposition instructions, option exercise and waiver, options hedging, hedging after performance, erroneous orders, erroneous cancel: are not included in the message count.
- Market makers are exempted from the message count commission for market-making varieties.
- The message count declaration fee for options on the GFEX is calculated based on the aggregate calculation of option series. The option contracts of the same product and the same expiration date are aggregated and charged according to the message count gradient. Futures and options on other exchanges are calculated based on the aggregate calculation of the contract.

When placing an order, YD will charge commissions according to the worst possible scenario for the order. After receiving the notification, it will deduct the over-calculated message count according to the actual situation of the notification. For example, if the OMS receives a limit order and counts 2 message counts, if the order is rejected, 2 message counts will be deducted. If the order is subsequently cancelled, no message count will be deducted. The message count of RFQ orders can be obtained through "YDAccountInstrumentInfo.RFQCount". Since the RFQ order does not send a report to ordinary investors, when placing an option RFQ order in the SHFE and INE, if the message count commission generated by the RFQ order is not passed, the RFQ order will be directly discarded without notifying the customer. However, this error message will be recorded in the OMS's inputFlow.txt, which is the same as other failed RFQ orders.

The message count commission uses a segmented accumulation calculation method. Please refer to the calculation example in the [Message Count Commission Rate](#). Investors using "ydExtendedApi" can obtain the total message count commission of the account through "YDExtendedAccount.MessageCommission".

Due to improper control of the investor's program, it may cause a large amount of unexpected message count commissions. In order to avoid the significant loss caused by the message count commissions to the investors, please contact the broker to set the [Message Count Risk Control](#).

4.8 Fund refresh mechanism

The YD margin, position profit/loss and position market value are not queried from an OMS but are calculated directly at the client according to the same method as that used on the OMS. Therefore, it is necessary to refresh the margin and position profit/loss during the trading session in order to keep the available funds in line with the OMS's.

Since '1.98', YD API has provided a variety of mechanisms to automatically refresh the margin rate and position profit/loss during trading, and to some great extents, supported investors using different APIs. By setting the RecalcMode parameter in the API configuration file, three refresh mechanisms of APIs can be specified: Close, Subscribe to Market Data Only and Auto. Three mechanisms are introduced separately in the following.

4.8.1 Close

"Close" means that the automatic refresh mechanism can be completely closed. It is up to investors to decide the refresh time and method.

For investors using ydApi, because no funds and positions are calculated through ydApi, all the refresh work should be completed by investors rather than the API.

For investors using ydExtendedApi, when necessary, they can refresh margins and position profit/loss by calling "recalcMarginAndPositionProfit" and refresh position market value by calling "recalcPositionMarketValue". Both functions are provided without input parameters. Since the "Close" mode does not support automatical subscribing to the market data, TCP market data (ConnectTCPMarketData=yes) collection must be ensured, otherwise the calculation and refresh should be conducted based on the pre-day market data.

4.8.2 Subscribe to market data

The "Subscribe to Market Data" mode can help investors automatically subscribe to and refresh the market data in relation to all current traded instruments or position instruments. If an option instrument is automatically subscribed to, its underlying instrument will be included automatically for the sake of accurate calculation of the option margin. However, the automatically subscribed market data will not be directly sent to investors through "notifyMarketData". It can be obtained through "notifyMarketData" if necessary. The subscription should be made separately through "Subscribe". This model is helpful to investors using ydApi and ydExtendedApi.

For investors using ydApi, the "YDMarketData" market data of the API can be refreshed when new market data arrives. Investors can directly read the price of the corresponding instrument for refresh.

For investors using "ydExtendedApi", the market data can be saved in "ydExtendedApi". When necessary, the investors can refresh the margin and position profit/loss by calling "recalcMarginAndPositionProfit", and refresh the position market value by calling "recalcPositionMarketValue". Both functions are provided without input parameters.

4.8.3 Auto mode

The "Auto" mode covers all functions of the "Subscribe to Market Data" mode, and based on which, further supports for different types of APIs are provided.

For investors using ydApi, in order to help to stagger the possible order handling time, the API can notify them of the safe refresh time detected by the system through "notifyRecalcTime". They can call their own refresh method through this function.

The safe refresh time is calculated based on the last received TCP market data time and the settings of "RecalcMarginPositionProfitGap" and "RecalcFreeGap" of the API configuration file. "RecalcMarginPositionProfitGap" refers to the minimum gap between two consecutive refreshes, which can be set to 1,000 ms. The setting below 1,000 ms should be adjusted to 1,000 ms. "RecalcFreeGap" refers to the safe gap between the safe refresh time and the time before and after the arrival of market data, the setting range of which is 0~100 ms. All settings beyond the range should be adjusted to the nearest valid value.

The specific calculation method is that after the last safe refresh time reaches RecalcMarginPositionProfitGap, assuming that the TCP market data received by API last time is 0 ms, and the 250 ms is a detection period. In this detection period, the start time of the remaining time period after cancelling the RecalcFreeGap ms before and after the detection period should be the safe refresh time. Assuming RecalcFreeGap is 100 ms, the time receiving the market data is 0 ms, the feasible time period is 100 ms~150 ms, the remaining time period is 50 ms and then the safe refresh time should be 100 ms.

Considering that the market data arrives continuously, causing the safe refresh time to be missed continuously, YD has set a protection time, namely if the last refresh time is more than 3 times of the RecalcMarginPositionProfitGap, ydApi will notify the customers to force the refresh through notifyRecalcTime.

For investors using ydExtendedApi, the API will automatically call "recalcMarginAndPositionProfit" and "recalcPositionMarketValue" to refresh at the safe refresh time, so coding is unnecessary. At this time, notifyRecalcTime will be used to notify investors after calling and completing the above two refresh functions, and investors can decide whether to do other work except refresh at this time.

YD has tried its best to help investors stagger those possible order handling time periods, but investors' order submission behavior might exceed YD's expectations. Therefore, if the refresh time under the "Auto" mode clashes with the order submission time, the parameters can be adjusted or the "Subscribe to Market Data" mode can be selected to determine the refresh time.

5 Positions

5.1 Derivatives position model

Different exchanges have different position management methods. Some exchanges strictly make a distinction between today's positions and pre positions and have special instructions for current/pre position closing; Some exchanges do not strictly make a distinction between today's positions and pre positions, nor do they have special instructions for current/pre position closing except for individual operations (such as commission for today's position closing). Therefore, YD has established the corresponding position model according to the position model of different exchanges. In order to distinguish the position model used by current exchanges, investors can check the value of "YDExchange.UseTodayPosition": When this value is "True", it means that the exchange makes a distinction between today's positions and pre positions, and two corresponding position model will be adopted by the YD system to represent today's positions and pre positions; When the value is "False", it means that the exchange does not make a distinction between today's positions and pre positions, and only one position structure will be adopted by the YD system to represent the corresponding positions.

The position organization mode of YDExtendedApi is basically the same as the internal one of YD OMSs, so the following will take the "YDExtendedPosition" position of YDExtendedApi as an example to introduce the rules and related logic used to calculate the commissions, position closing profit/loss and combined service through the above two position model. The fields related to the position structure of YDExtendedPosition are shown below. The primary keys are instrument, position date, position direction and hedge flag.

Field	Description
getInstrument()	Instrument
PositionDate	Position date YD_PSD_History: Pre position YD_PSD_Today: Today's position If the exchange does not make a distinction between today's positions and pre positions, YD_PSD_History should be used to represent them
PositionDirection	Position direction YD_PD_Long: long position YD_PD_Short: short position
HedgeFlag	The definition of hedge flags is different for futures exchanges and stock option exchanges. The definition of hedge flag for futures exchanges is as follows: YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of "YDExchange.UseArbitragePosition" is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support YD_HF_Hedge=3: hedge The definition of hedge flag for stock option exchanges is as follows: YD_HF_Normal=1: Normal YD_HF_Covered=3: covered
Position	For the total positions under the current primary keys, if a distinction is made between today's positions and pre positions, it means the today's positions and pre positions of an instrument regarding a direction and a hedge flag, respectively; if no any distinction is made between today's positions and pre positions, it means the total positions of the instrument regarding a position direction and a hedge flag
PositionDetailList	The position chain list used for profit/ loss calculation under the current primary keys is composed of positions that have not been closed according to the trading sequence Position closing always starts one by one from the head of the chain list, namely, according to the principle of Opening-Sequence Based Closing.
YDPositionForCommission	Pre positions used for commission calculation under current primary keys are only valid when no distinction is made between today's positions and pre positions. At this time, the today's positions used for commission calculation is "Position-YDPositionForCommission"

For a position chain list involved in combinations, the traditional combined positions are sequenced according to the combination sequence; For a position chain list not involved in combinations, the single-leg positions are sequenced according to the trading sequence. Due to the frequent changes in the structure of the traditional combined position details, YD did not directly open ports for traditional combined positions and single-leg positions in the above position model. Refer to the relevant query functions for querying the traditional combined positions. At present, only the exchanges that do not make a distinction between today's positions and pre positions support combination and decombination.

- If an exchange makes a distinction between today's positions and pre positions,
 - The following operations will be performed on the position with PositionDate set to YD_PSD_Today when opening a position.
 - Adding "Position" of the corresponding position to update the total positions
 - Adding the new trade volume after opening to the end of the position chain list for calculating the profit/loss of the above-mentioned positions

- When closing, investors should specify "YDInputOrder.OffsetFlag" as "YD_OF_CloseToday" or "YD_OF_CloseYesterday", which mean today's position closing or pre position closing, respectively (if set to "YD_OF_Close", it will be automatically converted to "YD_OF_CloseYesterday"), and the following operations will be completed for the affected positions:
 - Deducting the "Position" of today's positions or pre positions to update the volume of today's positions or pre positions
 - Updating the position chain list for profit/loss calculation according to the principle of Opening-Sequence Based Closing.
- When an exchange does not make a distinction between today's positions and pre positions,
 - The following operations will be performed on the position with PositionDate set to YD_PSD_Yesterday when opening a position.
 - Adding "Position" of the corresponding position to update the total positions
 - Adding the new opening transaction to the tail of the position chain list for calculating the profit/loss of the above-mentioned positions
 - If the exchange supports traditional combined position service, adding the new opening transaction to the tail of the traditional combined position chain list of the above-mentioned positions.
 - When closing, investors should specify YDInputOrder.OffsetFlag as YD_OF_Close, which means position closing (if set to YD_OF_CloseToday or YD_OF_CloseYesterday, it will be automatically converted to YD_OF_Close), and the following operations will be completed for the positions when PositionDate involves YD_PSD_Yesterday:
 - Deducting "Position" of corresponding positions to update the total positions
 - Updating the position chain list for profit/loss calculation according to the principle of "Opening-Sequence Based Closing".
 - If the value of "YDExchange.CloseTodayFirst" is "False", YDPositionForCommission will be deducted first; If the value of "YDExchange.CloseTodayFirst" is "True", only when "the Position closing volume" > "Today's positions used for commission calculation", the part that cannot be covered by today's positions used for commission calculation should be deducted from YDPositionForCommission
 - If the exchange supports automatic decombination of traditional combined positions, the items in the single-leg positions should be matched according to the principle of Opening-Sequence Based Closing until the closing volume is covered; If a complete coverage fails, the items in the traditional combined positions should be matched according to the principle of Opening-Sequence Based Closing until the remaining closing volume is covered, and the traditional combined positions and single-leg positions of the opposite leg in the traditional combined positions involved should be modified
 - When combining, the items that can cover the traditional combination volume from the head of the two-position single-leg positions for traditional combination should be selected and added to the end of the two-position combinations; When decombining, the items that can cover the decombination volume from the combined position chain list with two-position combinations should be selected and inserted into the appropriate points of the two-position single-leg positions, namely, the items for keeping the single-leg positions should be sequenced according to the trading sequence.

In addition to the aforementioned fields related to the position model, YDExtendedPosition also provides the following extended position fields.

Field	Description
getYDPosition()	Pre-position volume. Pre-position volume calculated according to the principle of Opening-Sequence Based Closing.
PositionByOrder	Order position volume. The position volume calculated based on the executed quantity in the order report. Available Closing Quantity can be $\min(\text{Position}, \text{PositionByOrder}) - \text{CloseFrozen}$
PossibleOpenVolume	Potential Position Opening Quantity. The sum of the order quantity for outstanding opening orders corresponding to the position. After the orders are completed (fully filled, canceled, or rejected), the potential opening quantity will be reduced by the unfilled quantity in the orders.
OpenFrozen	Frozen Opening Quantity. The sum of the unfilled quantity from outstanding opening orders corresponding to the position. After the order is completed (fully filled, canceled, or rejected), the frozen opening quantity of that order becomes zero.
CloseFrozen	Frozen Closing Quantity, the position that is frozen for closing cannot be liquidated or applied for exercise or abandonment. The sum of the unfilled quantity from outstanding closing orders corresponding to the position, exercise freeze quantity, abandonment freeze quantity, and the traditional portfolio position of stock options in Shanghai and Shenzhen. After the order is completed (fully filled, canceled, rejected, or position unwound), the closing freeze quantity of that order becomes zero.
ExecFrozen	Frozen Exercise Quantity. The sum of the application quantity for exercising options corresponding to the position.
AbandonExecFrozen	Frozen Abandonment Quantity. The sum of the application quantity for abandoning the right to exercise options corresponding to the position.
TotalCombPositions	Position Quantity in the Portfolio.
CombPositionCount	The number of records detailing the position quantity involved in the traditional portfolio.
TotalOpenPrice	Total cost of average open price. When opening a position, increase the opening cost of the open transaction. $\text{OpeningCost} = \text{OpeningPrice} \times \text{OpenPositionVolume}$. When closing a position, subtract the opening cost of the open transaction in the order of first to open and first to close. If more than one open transaction is involved, the cumulative deduction of the opening cost of the involved open transactions is subtracted.
getOpenPrice()	Average opening price. $\text{getOpenPrice} = \text{TotalOpenPrice} \div \text{OpenPosition}$ The average opening price is 0 when the position is 0.
TotalOriginalOpenPrice	Total cost of average position price. When opening a position, increase the opening cost of the open transaction. $\text{OpeningCost} = \text{OpeningPrice} \times \text{OpenPositionVolume}$. When closing a position, subtract the cost calculated at the average price of the position. $\text{times}(\text{getOriginalOpenPrice}) \times \text{times}(\text{ClosePosition})$, keep the average price of the position unchanged.

Field	Description
getOriginalOpenPrice()	Average position price. $\text{getOriginalOpenPrice} = \text{TotalOriginalOpenPrice} \div \text{Position}$ The average position price is 0 when the position is 0.
PositionProfit	Futures Position Profit/Loss. For more details, please refer to Profit/Loss on Futures Positions .
CloseProfit	Daily Mark-to-Market Contract's Closing Profit/Loss. In DCE's RULE margin system, options contracts are considered as daily mark-to-market contracts. The profit/loss from closing options positions is calculated in this field. In other cases, the profit/loss from closing options positions is calculated in the non-daily mark-to-market closing profit/loss. For more details, please refer to Closing Profit/Loss .
OtherCloseProfit	Closing Profit/Loss of Non-Daily Mark-to-Market Instruments. For more details, please refer to Closing Profit/Loss .
Margin	The margin calculated based on the set price is always zero after the new portfolio margin business is enabled. For reference on the set price, please consult YDSystemParam .
MarginPerLot	The margin per lot calculated based on the previous settlement price.

5.2 Spot position model

In spot trading, different trading businesses will generate or use different position types. Currently, YD supports three types of positions: history, today's trading, and today's creation and redemption.

Field	Description
m_pAccountInstrumentInfo	Pointer of account instrument info
HoldingPiece[YD_CHT_History]	Preday positions, fields refer to YDHoldingPiece structure
HoldingPiece[YD_CHT_TodayTrading]	Today buy position, the field refers to the YDHoldingPiece structure
HoldingPiece[YD_CHT_TodayCreationRedemption]	Today's creation and redemption positions, the fields refer to the YDHoldingPiece structure
TotalHolding	Total holdings, the fields refer to the YDHoldingPiece structure
ExternalSellFrozen	External freezes volume, including judicial freezes and stock option takeover freezes
HoldingAdjustStatus	The external system transfers in and out of the position status. This part of the position has been actually included in various types of actual positions

The description of the YDHoldingPiece structure is as follows:

Field and method	Description
Holding	Position Balance
BuyFrozen	Buy freeze volume
SellFrozen	Sell frozen volume
TotalCost	Buying cost, selling does not affect
BuyFrozenCost	Buying frozen cost
double AverageCost(int multiple=1)	Average opening price, calculation formula: $\text{TotalCost}/\text{Holding}$
int MaxPossibleHolding()	Maximum possible position, calculation formula: $\text{Holding} + \text{BuyFrozen}$
int MaxPossibleSellVolume()	Maximum possible short position, calculation formula: $\text{Holding} - \text{SellFrozen}$
double MaxPossibleCost()	Maximum possible cost, calculation formula: $\text{TotalCost} + \text{BuyFrozenCost}$

The description of the YDHoldingAdjustStatus structure is as follows :

Field	Description
TotalTransferInVolume	Open volume transferred in from external system
TotalTransferOutVolume	Open volume transferred out from external system
TotalTransferInPrice	Open cost transferred in from external system
TotalTransferOutPrice	Open cost transferred out from external system

Due to the complexity of calculating the available position during bidding, YD provides getMaxSellVolume in YDExtendedApi to obtain the current available sell volume during bidding.

```
1 | virtual int getMaxSellVolume(const YDExtendedHolding *pHolding)
```

5.3 Stock Option Spot Position Model

Stock option spot positions will only include securities that are the underlying assets of stock options on SSE and SZSE, namely ETFs and stocks that will be launched after SSE and SZSE launch individual stock options. Stock option spot positions are only used for stock option business, and are used in conjunction with the covered call and exercise business in the stock option business. Even if stock options and spot business are deployed on the same counter, for the same securities, stock option spot positions are completely independent of spot positions.

The position organizing method of YDExtendedApi is basically the same as the organizing method of the YD OMS, so the following will take the position YDExtendedSpotPosition of YDExtendedApi as an example to introduce the relevant concepts of option spot positions. The fields related to the position model of YDExtendedSpotPosition are as follows, with the primary keys being investors and securities.

Field	Description
m_pAccountInstrumentInfo	Pointer of the account instrument info.
Position	Stock option spot's position
ExchangeFrozenVolume	Exchange Frozen Volume for covered call opening
CoveredVolume	Covered call position volume
ExecVolume	E-day put option exercise freeze amount
ExecAllocatedVolume	Exercise allocated volume, always equal to YDSpotPrePosition's, no change during the trading
ExecAllocatedAmount	Exercise allocated amount, always equal to YDSpotPrePosition's, no change during the trading
ExecAllocatedFrozenVolume	Exercise allocated frozen volume, always equal to YDSpotPrePosition's, no change during the trading
ExecAllocatedFrozenAmount	Exercise allocated frozen amount, always equal to YDSpotPrePosition's, no change during the trading

Unlike other position models, the daily initial value and intraday real-time value of stock option spot positions are related to the adopted business model. YD supports both incremental and full business models. The incremental model is more in line with the essence of stock option business and more versatile than the full model. Therefore, YD will migrate the full model to the incremental model at the appropriate time. At present, the YD stock option counter deployed by futures companies uses the full model, while the securities company uses the incremental model.

The stock option spot position business model refers to the relationship between the positions and amounts between the stock option counter at the beginning of the day and during the trading day, the spot counter, the Shanghai and Shenzhen Stock Exchange trading system, and the China Securities Depository and Clearing Corporation's settlement system. When different business models are adopted, the meaning of fund positions is different, and investors also need to use different API interfaces when trading. Stock option spot positions, exercise allocation frozen volume, and exercise allocation frozen amount are related to the model, and their calculation methods are explained in the specific model; exchange locked volume, covered volume, exercise frozen volume, net exercise allocation volume, and net exercise allocation amount are not related to the model, as explained below:

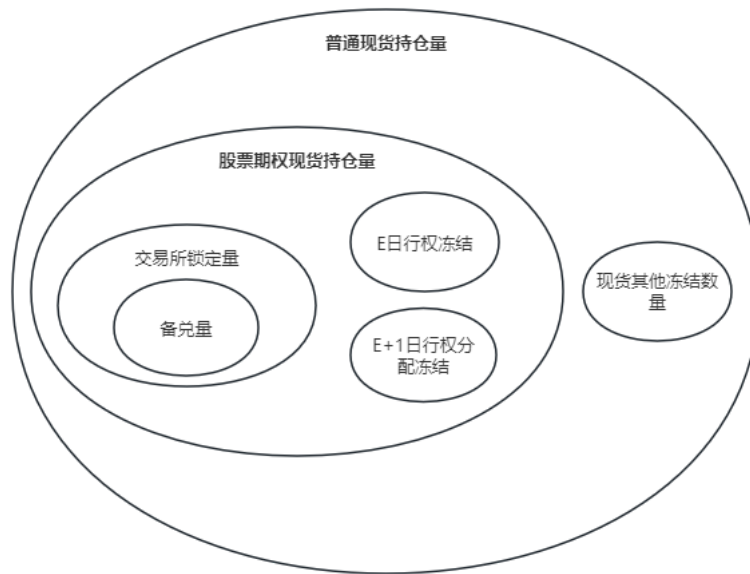
- Covered Volume is calculated based on option covered positions and is always calculated using the exchange contract multiplier
- Exchange locked volume ExchangeFrozenVolume is usually equal to covered volume at the beginning of the day, but when insufficient coverage occurs on the ex-rights and ex-dividend date or the exercise allocation date, the exchange locked volume is less than the covered volume. The specific amount is calculated according to the exchange business rules and is related to the ordinary spot positions and exercise allocation frozen volume. If the intraday exchange locked volume is less than the covered volume, investors need to buy spot in time to prevent the covered position from being liquidated by SSE or converted to ordinary option positions by SZSE
- ExecVolume is the stock option spot position frozen when investors initiate put option exercise during the exercise day
- ExecAllocatedVolume is the total exercise settlement amount based on the difference between the after-hours pairing settlement results on the E day. Positive values represent receipts, negative values represent delivery, and the beginning of the day and the intraday values are the same and unchanged
- ExecAllocatedAmount is the total exercise settlement amount based on the difference between the after-hours pairing settlement results on the E day. Positive values represent receipts, negative values represent delivery, and the beginning of the day and the intraday values are the same and unchanged

The maximum exercise application, SSE lock, and SZSE covered quantity that investors can initiate during the trading day is the available quantity of stock option spot, and its calculation formula is:

$$Availablequantityofstockoptionsspot = stockoptionsspotholdings - exercisefrozenquantity - exerciseallocationfrozenquantity - exchangelocked$$

5.3.1 Stock Option Increment Model

In the incremental model, the stock option spot position is the amount of ordinary spot positions occupied and frozen by the stock option business. This part of the position can only be used for the exercise and covered call business of stock options. When investors need more spot positions for the exercise and covered call business of stock options, they need to take over the ordinary spot positions to the stock option spot positions. Similarly, when investors want to sell excess stock option spot positions, they need to take the initiative to return the stock option spot positions to the ordinary spot positions. The relationship diagram of each position is shown below.



According to the instructions on freezing spot positions on the exercise settlement day in the "Guidelines for the Pilot Settlement of Stock Options of the Shenzhen Branch of China Securities Depository and Clearing Co., Ltd.", China Securities Depository and Clearing Co., Ltd. requires that the exercise allocated frozen volume `ExecAllocatedFrozenVolume` be the sum of the spot positions corresponding to the put option positions that are continuously frozen during the E+1 day, the assigned matured covered call positions, and the unexpired covered call positions.

- 1 Unlock all covered securities locked in the previous trading day
- 2 Conduct an exercise validity check (i.e. check the number of exercise contracts in the contract account of the exercise declaring party and check whether the number of underlying securities available in the securities account of the put option exercising party at the end of the day is sufficient), and lock the underlying securities required for the exercise of the put option contract
- 3 Assign all valid exercise declarations (including calls and puts)
- 4 After the exercise is assigned, cancel the undeclared exercise and unassigned expired contracts
- 5 Release the maintenance margin in the fund margin account of the settlement participant corresponding to the contract that has not been assigned to exercise
- 6 Re-lock the covered securities corresponding to the assigned expired covered open contracts and the covered securities corresponding to all unexpired covered open contracts at the end of the day

According to the "Rules of China Securities Depository and Clearing Co., Ltd. on the Pilot Settlement of Stock Options of Shanghai Stock Exchange": Based on the inspection results, the Company assigns the effective exercise and the exercised party according to the principles of "proportional allocation" and "allocation according to the size of the tail number", sends the exercise assignment results to the settlement participants, and locks the contract subject required for the exercise and delivery in the securities account corresponding to the contract account of the put option exerciser. **The maintenance margin corresponding to the assigned contract of the exercised settlement participant shall not be released.** If the settlement participant of the contract subject fails to deliver the contract subject, **the Company shall make cash settlement for the undelivered part at 110% of the closing price of the contract subject on the day.**

Although China Securities Depository and Clearing Co., Ltd. has stipulated the calculation method of the exercise allocation frozen amount `ExecAllocatedFrozenAmount`, some securities companies have formulated business rules for the exercise allocation frozen amount based on the actual business development situation under the premise of not less than the supervision regulations of China Securities Depository and Clearing Co., Ltd., and YD calculates the exercise allocation frozen amount according to the business rules of each securities company. Due to the confidentiality requirements of securities companies for their business rules, the specific calculation method is not listed here. Investors are requested to consult securities companies for specific calculation methods.

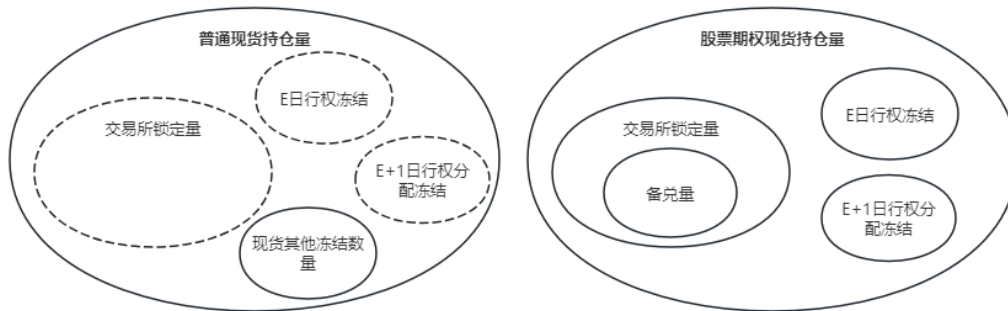
According to the definition of this model, the stock option spot position at the beginning of the day should be equal to the position that the spot system considers to be initially taken over by the stock option system at the beginning of the day. When the ordinary spot position is sufficient, its value should be equal to the sum of the covered position and the exercise allocation frozen amount. When the ordinary spot position is not enough to cover the covered position and the exercise allocation frozen spot position, its value will be adjusted accordingly according to the rules of the spot system so that it is the same as the position that the spot system considers to be initially taken over by the stock option system at the beginning of the day. When investors take over spot positions during the trading session, the ordinary spot positions during the trading session will be reduced and the stock option spot positions during the trading session will be increased accordingly. The takeover quantity cannot exceed the takeover quantity of the spot system. The calculation formula is

$Takeoverquantity = ordinaryspotposition - otherspotfrozenquantity - stockoptionspotposition.$

5.3.2 Stock Option Full Model

In the full model, the stock option spot position is a mirror image of the ordinary spot position. The two are kept consistent through a two-way synchronization mechanism. The ordinary spot position is synchronized to the stock option spot position at a regular interval, and the sum of the exchange lock-up amount, exercise freeze amount and exercise allocation freeze amount of the stock option system is synchronized to the spot system. The full model has two problems: regulatory risk and narrow applicability:

- Since the exchange lock-up amount of the stock option system is synchronized to the spot system after the business occurs, there is a time difference between the two. During this time difference, the spot position has been occupied in the exchange, but the spot system believes that the spot position has not been locked. If the investor sells this part of the position in the spot system at this time, the spot system will not be able to effectively intercept the sell order and send it to the exchange, which will be rejected by the exchange as a wrong order. This situation will be regarded by the regulator as a loophole in the counter money warehouse management
- Most spot counters do not support the functions of reading the total position and locking the absolute number, which limits the full model from being used in most scenarios



The formula for the ExecAllocatedFrozenVolume is:

$$ExecAllocatedFrozenVolume = -\min(netexerciseallocationamount, 0)$$

ExecAllocatedFrozenVolume is defined in accordance with the business rules of futures companies, and its calculation formula is:

$$Netexerciseallocationamount = \min(netexercisettlementamount + netdifferencebonddefaultpenalty, netmaintenancemargin)$$

The definitions of each item in the above formula are as follows:

- Net exercise settlement amount is the original settlement amount calculated based on the after-hours matching settlement results on E day. Positive values represent receipt of money, and negative values represent delivery of money
- Net difference bond default penalty is the funds frozen in advance to prevent bond default. Negative values represent the need to freeze the default penalty. 0 means no default penalty, and it cannot be a positive value. In the pairing settlement results after the E-day trading,
 $thenumberofdifferencebonds = \max((payablebonds - receivablebonds) - (totalnumberofbonds - frozenbonds), 0)$
, then
 $netdifferencebondpenalty = -numberofdifferencebonds \times theclosingpriceofthecurrentbondontheE - day \times (1 + 10\%) \times (1 + 10\%)$
, where the first 10% is the regulatory requirement, and the second 10% cannot determine which contract is in default, so it can only be frozen according to the upper limit
- Net maintenance margin is the funds frozen by the option contract in accordance with regulatory requirements during the E+1 day trading. Negative values mean that the margin needs to be frozen, 0 means no margin, and it cannot be a positive value

YD provides a spot counter connector to achieve real-time two-way position synchronization during the trading session. If the broker enables the spot counter connector, the API will notify the spot position synchronization through the following callback function that it is activated. The notification message is sent continuously. If no notification is received within 30 seconds, it means that the synchronization mechanism is interrupted. Please contact the broker in time to troubleshoot the problem.

```
1 | virtual void notifySpotAlive(const YDExchange *pExchange)
```

In the synchronous activation state, if the position on the spot counter changes, ydApi and ydExtendedApi will be notified through the following callbacks, and newPosition is the updated spot position:

```
1 | virtual void notifySpotPosition(const YDInstrument *pInstrument, const YDAccount *pAccount, int newPosition)
```

If investors use YDExtendedListener, then notifyExtendedSpotPosition will be called back at any time when notifySpotPosition is called back:

```
1 | virtual void notifyExtendedSpotPosition(const YDExtendedSpotPosition *pSpotPosition)
```

If the synchronization mechanism is not configured or fails during the trading session, the YD stock options counter will skip checking the spot positions of locked positions, covered call openings, and put option exercises, which may cause the following problems:

- If the actual lockable positions are insufficient when locking positions
- If the actual available spot positions are insufficient when covered call openings, the exchange will return an opening error
- If the actual available spot positions are insufficient when exercising put options, the **Exchange will not check the spot positions and will accept the exercise instructions**, which will cause investors to mistakenly believe that the spot positions are sufficient. However, the exercise instructions will fail during settlement, causing investors to suffer losses.

5.4 Position query

This section mainly introduces the method for querying real-time positions through ydExtendedApi. All queries are made based on the local data of ydExtendedApi and will not be conducted at the OMSs. Since all query services are locked, there will be a certain loss in performance. All query methods must be called after notifyCaughtUp, otherwise the position data will be inaccurate due to incomplete order trading data. To query preday-positions, see [Preday-Position](#) and [traditional-combined-preday-position](#).

5.4.1 Derivatives position query

A single futures and options position can be queried by calling "getExtendedPosition". For the notified YDExtendedPosition structure, please refer to [Position Structure](#).

```
1 virtual const YDExtendedPosition *getExtendedPosition(int positionDate,int positionDirection,int
  hedgeFlag,const YDInstrument *pInstrument,const YDAccount *pAccount=NULL,bool create=false);
```

The parameters for calling the above method are as follows:

Parameter	Description
positionDate	Date of position to be queried YD_PSD_History: Pre position YD_PSD_Today: Today's position If the exchange does not make a distinction between today's positions and pre positions, the YD_PSD_History should be used
positionDirection	Position direction to be queried YD_PD_Long: Long position YD_PD_Short: Short position
hedgeFlag	Position hedge flag to be queried, the definition of hedge flags is different for futures exchanges and stock option exchanges. The definition of hedge flag for futures exchanges is as follows: YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of YDExchange.UseArbitragePosition is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge The definition of hedge flag for stock option exchanges is as follows: YD_HF_Normal=1: Normal YD_HF_Covered=3: covered
pInstrument	Instrument pointer to be queried
pAccount	When NULL is filled in, it means that the current API login account is used
create	For determining whether to create an empty position when no position is found. If "True", when no position is found, a newly initialized position of 0 will be sent back. If "False", a NULL will be sent back when no position is found

YD provides following two different methods for multi-position query, which have the same query parameters:

The first method requires investors to allocate a fixed-length YDExtendedPosition pointer array in advance. If the pre-allocated length is not enough to accommodate, only the positions of the pre-allocated array length will be filled. Regardless of whether the pre-allocated length is sufficient, the return value of this method is the total number of positions that meet the query conditions. Investors can use this feature to call findExtendedPositions(pFilter, 0, NULL) to quickly obtain the total number of positions that meet the conditions.

- The advantage of this method is that when there are not many positions and the pre-allocated array length is sufficient, the pre-allocated pointer array can be reused without allocating it for each query
- The disadvantage of this method is that if there are many positions, it may take two calls (the first to obtain the total number of positions, the second to allocate an array that can accommodate all positions and then call again) to fully obtain all positions that meet the conditions

The second method can help investors allocate space that can accommodate all positions that meet the conditions, but investors need to actively call YDQueryResult.destroy() to destroy the allocated space after use. Delete cannot be used to delete. Compared with the first method:

- The advantage of this method is that all positions are returned after one call
- The disadvantage of this method is that investors need to actively release the space allocated by this method, and new space will be allocated for each call. Frequent allocation and release is very unfriendly to the cache

```
1 // positions must have spaces of count, return real number of positions(may be greater than count). Only
  partial will be set if no enough space
2 virtual unsigned findExtendedPositions(const YDExtendedPositionFilter *pFilter,unsigned count,const
  YDExtendedPosition *positions[]);
3
4 // User should call destroy method of return object to free memory after using following method
5 virtual YDQueryResult<YDExtendedPosition> *findExtendedPositions(const YDExtendedPositionFilter *pFilter);
```

The instructions for filling in each field of parameter YDExtendedPositionFilter are as follows::

Field	Description
PositionDate	The date of the position to be queried. Setting it to -1 means that this parameter is not effective. YD_PSD_History: Yesterday's position YD_PSD_Today: Today's position

Field	Description
PositionDirection	Position direction. Setting it to -1 means that this parameter is not effective. YD_PD_Long: Long position YD_PD_Short: Short position
HedgeFlag	Hedge flag of the position to be queried. It has different definitions between futures exchanges and stock options exchanges. Setting it to -1 means that this parameter is not effective. YD_PD_Long: Long position YD_PD_Short: Short position For future exchanges: YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only available for CFFEX. To facilitate programming, YDExchange provides a parameterized representation of whether arbitrage trading and positions are supported. When YDExchange.UseArbitragePosition is true, it means that the exchange supports arbitrage trading and positions; otherwise, it does not support. YD_HF_Hedge=3: Hedge For stock exchanges: YD_HF_Normal=1: normal YD_HF_Covered=3: covered
pInstrument	Instrument pointer. Set to NULL for no limit.
pProduct	Product pointer, Set to NULL for no limit.
pExchange	Exchange pointer, Set to NULL for no limit.
pAccount	Investor should always set to NULL

The pseudo-code logic for determining whether a position meets the query criteria is as follows:

```

1  if YDExtendedPositionFilter.PositionDate>=0 and
   YDExtendedPosition.PositionDate!=YDExtendedPositionFilter.PositionDate:
2      return false;
3
4  if YDExtendedPositionFilter.PositionDirection>=0 and
   YDExtendedPosition.PositionDirection!=YDExtendedPositionFilter.PositionDirection:
5      return false;
6
7  if YDExtendedPositionFilter.HedgeFlag>=0 and
   YDExtendedPosition.HedgeFlag!=YDExtendedPositionFilter.HedgeFlag:
8      return false;
9
10 if YDExtendedPositionFilter.pInstrument!=NULL and
   YDExtendedPosition.Instrument!=YDExtendedPositionFilter.pInstrument:
11     return false;
12
13 if YDExtendedPositionFilter.pProduct!=NULL and
   YDExtendedPosition.Product!=YDExtendedPositionFilter.pProduct:
14     return false;
15
16 if YDExtendedPositionFilter.pExchange!=NULL and
   YDExtendedPosition.Exchange!=YDExtendedPositionFilter.pExchange:
17     return false;
18
19 return true;

```

The description of each field of the parameter YDExtendedPositionFilter is as follows:

Field	Description
PositionDate	Date of position to be queried, when set to 0, it means that this parameter is invalid. YD_PSD_History: pre position YD_PSD_Today: today's position
PositionDirection	Position direction to be queried, when set to 0, it means that this parameter is invalid YD_PD_Long: Long position YD_PD_Short: Short position
HedgeFlag	Position hedge flag to be queried, the definition of hedge flags is different for futures exchanges and stock option exchanges. When set to 0, it means that this parameter is invalid The definition of hedge flag for futures exchanges is as follows: YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of "YDExchange.UseArbitragePosition" is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge The definition of hedge flag for stock option exchanges is as follows: YD_HF_Normal=1: Normal YD_HF_Covered=3: covered

Field	Description
pInstrument	Instrument pointer, when set to NULL, it means that no limit will be made
pProduct	Product pointer, when set to NULL, it means that no limit will be made
pExchange	Exchange pointer, when set to NULL, it means that no limit will be made
pAccount	The "Investor" should always be set to NULL

The first method requires investors to allocate a fixed-length "YDExtendedPosition" pointer array in advance. If the pre-allocated length is not enough for the space, only the positions with the pre-allocated array length can be filled. Whether the pre-allocated length is enough or not for the space, the return values regarding this method are the total positions meeting the query criteria. Investors, relying on this, can call "findExtendedPositions(pFilter, 0, NULL)" to quickly obtain the total positions meeting the criteria.

- The advantage of this method is that when there are not many positions and the length of the pre-allocated array is enough, the pre-allocated pointer array can be reused without allocating for each query;
- The disadvantage of this method is that if there are many positions, it may need to be called twice (the first time aims to obtain the total positions, the second time, to allocate the array that can include all positions before calling again) to fully obtain all positions meeting the criteria.

The second method can help investors allocate a space for all positions meeting the criteria. However, the allocated space, after being used, should be destroyed actively by calling "YDQueryResult.destory()" since it is not suitable for being deleted by "Delete". Compared with the first method:

- The advantage of this method is that all positions can be notified by one call
- The disadvantage of this method is that the investors need to actively release the space allocated according to this method, and each call will lead to the allocation of a new space, however, frequent allocation and release are very unfriendly to the cache.

5.4.2 Spot position query

To obtain the spot position of a single stock option, you can call the getExtendedSpotPosition method. For the returned YDExtendedPosition structure and meaning, please refer to [Stock Option Spot position model](#).

```
1 virtual const YDExtendedSpotPosition *getExtendedSpotPosition(const YDInstrument *pInstrument, const
  YDAccount *pAccount=NULL, bool create=false)
```

The parameters for calling the above method are described as follows:

Parameter	Description
pInstrument	Instrument pointer to be queried
pAccount	When NULL is filled in, it means that the current API login account is used
create	For determining whether to create an empty position when no position is found. If "True", when no position is found, a newly initialized position of 0 will be sent back. If "False", a NULL will be sent back when no position is found

YD provides a method for multi-position query. The method can help investors allocate a space for all positions meeting the criteria. However, the allocated space, after being used, should be destroyed actively by calling "YDQueryResult.destory()".

```
1 /// user should call destroy method of return object to free memory after using following method
2 virtual YDQueryResult<YDExtendedSpotPosition> *findExtendedSpotPositions(const
  YDExtendedSpotPositionFilter *pFilter)
```

The instructions for filling in each field of the parameter YDExtendedSpotPositionFilter are as follows:

Field	Description
pInstrument	Instrument pointer. If set to NULL, no limit.
pProduct	Product pointer. If set to NULL, no limit.
pExchange	Exchange pointer. If set to NULL, no limit.
pAccount	Investors please always set to NULL.

The pseudo-code logic for determining whether a position meets the query criteria is as follows:

```

1  if YDExtendedSpotPositionFilter.pInstrument!=NULL and
    YDExtendedSpotPosition.Instrument!=YDExtendedSpotPositionFilter.pInstrument:
2      return false;
3
4  if YDExtendedSpotPositionFilter.pProduct!=NULL and
    YDExtendedSpotPosition.Product!=YDExtendedSpotPositionFilter.pProduct:
5      return false;
6
7  if YDExtendedSpotPositionFilter.pExchange!=NULL and
    YDExtendedSpotPosition.Exchange!=YDExtendedSpotPositionFilter.pExchange:
8      return false;
9
10 return true;

```

5.4.3 Traditional combined position details query

YD provides the following method for querying the combined position details of SSE and SZSE. The "combPositionDetailID" is the only primary key for determining combined details:

```

1  /// getCombPositionDetail can only be used in SSE/SZSE
2  virtual const YDExtendedCombPositionDetail *getCombPositionDetail(int combPositionDetailID);

```

Note

YD API does not provide method to query traditional combined position, investors should maintain their data based on [Traditional Combined Preday Position](#).

The fields of the notified "YDExtendedCombPositionDetail" structure are as follows:

Field	Description
m_pAccount	Pointer of investors' traditional combined position details
m_pCombPositionDef	Pointer of traditional combined position definition
Position	Detailed traditional combined positions
CombPositionDetailID	Traditional combined position details ID, valid only in SSE and SZSE

YD provides the above two different methods for multi-query of traditional combined position details, which have the same query parameters:

```

1  /// combPositionDetails must have spaces of count, return real number of combPositionDetails(may be
    greater than count). Only partial will be set if no enough space
2  virtual unsigned findCombPositionDetails(const YDCombPositionDetailFilter *pFilter,unsigned count,const
    YDExtendedCombPositionDetail *combPositionDetails[]);
3
4  /// User should call destroy method of return object to free memory after using following method
5  virtual YDQueryResult<YDExtendedCombPositionDetail> *findCombPositionDetails(const
    YDCombPositionDetailFilter *pFilter);

```

The pseudo-code logic for determining whether traditional combined position details meet the query criteria is as follows. For readability, YDExtendedCombPositionDetail is abbreviated as Detail, and YDCombPositionDetailFilter is abbreviated as Filter:

```

1  if Filter.IncludeSplit==false and Detail.Position<=0:
2      return false
3
4  if Filter.pAccount!=NULL and Detail.m_pAccount!=Filter.pAccount:
5      return false
6
7  if Filter.pCombPositionDef!=NULL and Detail.m_pCombPositionDef!=Filter.pCombPositionDef:
8      return false
9
10 if Filter.pInstrument!=NULL:
11     hasMatchLeg=false
12     for (int legID=0;legID<2;legID++):
13         if matchLeg(Detail->m_pCombPositionDef,legID,Filter):
14             hasMatchLeg=true
15             break
16     if hasMatchLeg==false:
17         return false
18
19 return true
20
21 bool matchLeg(YDCombPositionDef Def,int legID,YDCombPositionDetailFilter Filter):
22     if Def.m_pInstrument[legID]!=Filter.pInstrument:
23         return false
24
25     if Filter.PositionDirection!=0 and Filter.PositionDirection!=Def.PositionDirection[legID]:
26         return false
27
28     return true

```

Each field of the parameter YDExtendedPositionFilter to be filled out is described as follows:

Field	Description
pCombPositionDef	Pointer of traditional combined position definition. When set to NULL, it means that no limit will be made.
pInstrument	Pointer of instrument. When set to NULL, it means that no limit will be made.
PositionDirection	Position direction of traditional combined position details to be queried, when set to 0, it means that this parameter is invalid. YD_PD_Long: Long position YD_PD_Short: Short position
IncludeSplit	For determining whether to include traditional combined position details that have been decombined.

The first method requires investors to allocate a fixed-length "YDExtendedCombPositionDetail" pointer array in advance. If the pre-allocated length is not enough for the space, only the traditional combined position details with the pre-allocated array length can be filled. Whether the pre-allocated length is enough or not for the space, the return values regarding this method are the total traditional combined position details meeting the query criteria. Investors, relying on this, can call "findCombPositionDetails(pFilter, 0, NULL)" to quickly obtain the total traditional combined position details meeting the criteria.

- The advantage of this method is that when there are not many traditional combined position details and the length of the pre-allocated array is enough, the pre-allocated pointer array can be reused without allocating for each query;
- The disadvantage of this method is that if there are many traditional combined position details, it may need to be called twice (the first time aims to obtain the total traditional combined position details, the second time, to allocate the array that can include all traditional combined position details before calling again) to fully obtain all traditional combined position details meeting the criteria.

The second method can help investors allocate a space for all traditional combined position details meeting the criteria. However, the allocated space, after being used, should be destroyed actively by calling "YDQueryResult.destory()" since it is not suitable for being deleted by "Delete". Compared with the first method:

- The advantage of this method is that all traditional combined position details can be notified by one call
- The disadvantage of this method is that the investors need to actively release the space allocated according to this method, and each call will lead to the allocation of a new space, however, frequent allocation and release are very unfriendly to the cache.

6 Margin model

Starting from 2023, various exchanges have already or are about to introduce new portfolio margin models. In order to differentiate, YD refers to the previous margin model as the traditional margin model. After analyzing the solutions provided by various exchanges, it has been determined that for a considerable period, both the traditional margin and portfolio margin will coexist. This means that for the same investor, there may be some products that use the traditional margin model while others use the portfolio margin model. As a result, YD has established the concept of margin models and applies them at the investor and product level. This means that products using the same margin model can participate together in margin benefits such as hedging and portfolio optimization.

Currently, YD supports the following margin models:

- YD_MM_Normal=0: Traditional Margin Model
- YD_MM_SPBM=1: CZCE SPBM Portfolio Margin Model
- YD_MM_RULE=2: DCE RULE Portfolio Margin Model

For investors using the ydApi, YD does not provide a method to determine which trading model a specific product or instrument belongs to. Investors need to parse this information from the [Combination margin parameters](#). For investors using the ydExtendedApi, the following method can be used to obtain the margin model to which a specific instrument or product belongs.

```
1 virtual int getMarginModel(const YDInstrument *pInstrument, const YDAccount *pAccount=NULL)
2 virtual int getMarginModel(const YDProduct *pProduct, const YDAccount *pAccount=NULL)
```

6.1 Traditional margin model

The margin collection and deduction principle of YD Futures Exchange is to **keep consistent with that of the mainstream primary OMS** as much as possible. It should be noted that YD and the mainstream primary OMS can help to calculate the large-side margin together with the pending orders and positions of SSE, INE and CFFEX, however at present, the following two points are inconsistent with those of the mainstream primary OMS in terms of business rules:

- For the combined option margin of DCE, the calculation methods of YD and mainstream primary OMS are different;
- For locked orders of CZCE, considering the positions and pending orders, YD deducts the margin according to the large-side margin calculation method. At present, for the mainstream primary OMS, only the large-side margin deduction of positions rather than that of pending orders can be calculated.

Therefore, except for the above two business cases, the margin authorization of YD and the mainstream primary OMS should be completely consistent when exchanges are closed, however, it may vary slightly during trading considering different notification sequences, market data update time and OMS recalculation time.

YD does not make a distinction between the frozen order margin and the position margin but considers it as a margin as a whole. A large-side margin is also calculated according to the sum of the order margin and position margin. For the sake of investors' easy understanding, this text makes a distinction between the order margin and position margin though they are not distinct by OMSs.

6.1.1 Margin rate

To calculate the position margin, the margin rate of the instrument about the corresponding hedge flag should be obtained first. It is suggested that investors directly use "getInstrumentMarginRate" to obtain the corresponding margin rate instead of reading it directly from "YDAccountInstrumentInfo".

```
1 virtual const YDMarginRate *getInstrumentMarginRate(const YDInstrument *pInstrument, int hedgeFlag, const
  YDAccount *pAccount=NULL)
```

The notified YDMarginRate structure has a large number of "union" fields for adapting to different margin models. For the sake of easy understanding, margin rate parameters under three different margin models are listed below.

The following fields are apply to futures margins of futures exchanges.

Field	Description
m_pAccount	Pointer of investors' margin rate
m_pProduct	Product margin rate pointer
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of "YDExchange.UseArbitragePosition" is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge
LongMarginRatioByMoney	Long margin rate based on amount
LongMarginRatioByVolume	Long margin rate based on lots
ShortMarginRatioByMoney	Short margin rate based on amount
ShortMarginRatioByVolume	Short margin rate based on lots

The following fields are apply to the commodity option margins of futures exchanges and the stock index option margin of CFFEX.

Field	Description
m_pAccount	Pointer of investors' margin rate
m_pProduct	Product margin rate pointer
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. In order to facilitate the coding, YD provides a parameterized representation to determine whether to support arbitrage trading and positions. When the value of YDExchange.UseArbitragePosition is "True", it means that the exchange supports arbitrage trading and positions, otherwise it does not support. YD_HF_Hedge=3: hedge
CallMarginRatioByMoney	Short margin rate of call option based on amount
CallMarginRatioByVolume	Short margin rate of call option based on lots
PutMarginRatioByMoney	Short margin rate of put option based on amount
PutMarginRatioByVolume	Short margin rate of put option based on lots

The following fields are apply to the stock option margins of SSE and SZSE.

Field	Description
m_pAccount	Pointer of investors' margin rate
m_pProduct	Product margin rate pointer
m_pInstrument	Pointer of instrument corresponding to margin rate
HedgeFlag	YD_HF_Normal=1: Normal YD_HF_Covered=3: Covered
BaseMarginRate	Margin rate of base instrument
LinearFactor	Linear factor
LowerBoundaryCoef	Minimum boundary coefficient

6.1.2 Futures margin

The calculation formula for long futures position margin is:

$$\text{LongFuturePositionMargin} = (\text{Price} \times \text{InstrumentMultiplier} \times \text{LongPositionMarginRatePerAmount} + \text{LongPositionMarginRatePerLot}) \times \text{Quantity}$$

The calculation formula for short futures position margin is:

$$\text{ShortFuturePositionMargin} = (\text{Price} \times \text{InstrumentMultiplier} \times \text{ShortPositionMarginRatePerAmount} + \text{ShortPositionMarginRatePerLot}) \times \text{Quantity}$$

For **orders**, the order volume is used for calculating the futures order margin. The price is controlled when the parameter value (Name, Target) of YDSystemParam is (OrderMarginBasePrice, Futures). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price
- YD_CBT_OrderPrice: When the order is a market price one, the upper limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_SamePrice: The same price used as that for the margin rate of futures positions, namely the parameter value obtained when (Name, Target) of YDSystemParam is (MarginBasePrice, Futures) is used. When the used margin rate of futures positions is subject to YD_CBT_OpenPrice at this time, the price used for futures order margin will be a pre settlement one

For **pre positions**, the position volume is used for calculating the futures position margin, and the pre settlement price is always used. The pre positions here refer to the remaining positions after settlements made based on the mark-to-market rules, rather than the concept of today's position / pre position of [Derivative Position Model](#).

For **today's positions**, the position volume is used for calculating the futures position margin. The price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Futures). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price
- YD_CBT_OpenPrice: Open price, The margins corresponding to the position details is calculated one by one according to the open price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_LastPrice: Latest price
- YD_CBT_MarketAveragePrice: Average market price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. When the trading volume of the instrument is zero for the day, the previous settlement price is used.

6.1.3 Option margin

6.1.3.1 Commodity option margin

For commodity options traded in SHFE, INE, DCE, GFEX and CZCE.

The margin calculation formulas for call commodity options are as follows. OTM in the following formulas stands for out of the money.

$$\text{CommodityCallOptionMargin} = [\text{OptionPrice} \times \text{OptionInstrumentMultiplier} + \max(\text{CallOptionBaseMargin} - \text{CallOptionOTM}/2, \text{CallOptionBaseMargin}/2)] \times \text{Quantity}$$

$$\begin{aligned} \text{CallOptionBaseMargin} &= \text{UnderlyingInstrumentPrice} \times \text{UnderlyingInstrumentMultiplier} \\ &\times \text{CallOptionShortPositionMarginRatePerAmount} + \text{CallOptionShortPositionMarginRatePerLot} \\ \text{CallOptionOTM} &= \max((\text{ExercisePrice} - \text{UnderlyingInstrumentPrice}) \times \text{OptionInstrumentMultiplier}, 0) \end{aligned}$$

The margin calculation formulas for put commodity options are:

$$\text{CommodityPutOptionMargin} = [\text{OptionPrice} \times \text{OptionInstrumentMultiplier} + \max(\text{PutOptionBaseMargin} - \text{PutOptionOTM}/2, \text{PutOptionBaseMargin}/2)] \times \text{Quantity}$$

$$\begin{aligned} \text{PutOptionBaseMargin} &= \text{UnderlyingInstrumentPrice} \times \text{UnderlyingInstrumentMultiplier} \\ &\times \text{PutOptionShortPositionMarginRatePerAmount} + \text{PutOptionShortPositionMarginRatePerLot} \end{aligned}$$

$$\text{PutOptionOTM} = \max((\text{UnderlyingInstrumentPrice} - \text{ExercisePrice}) \times \text{OptionInstrumentMultiplier}, 0)$$

For **orders**, the order volume is used for calculating the futures order margin. The price is controlled when the parameter value (Name, Target) of YDSystemParam is (OrderMarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_OrderPrice: When the order is a market price one, the upper limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail.
- YD_CBT_SamePrice: The same price used as that for the margin rate of futures positions, namely the parameter value obtained when (Name, Target) of YDSystemParam is (MarginBasePrice, Options) is used. When the used margin rate of futures positions is subject to YD_CBT_OpenPrice at this time, the price used for futures order margin will be a pre settlement one

For **pre positions** and **today's positions**, the position volume is used for calculating the futures position margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price
- YD_CBT_OpenPrice: The margins corresponding to the position details is calculated one by one according to the open price.
- YD_CBT_LastPrice: Latest price
- YD_CBT_MarketAveragePrice: Average market price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS

For **orders**, **pre positions** and **today's positions**, the underlying instrument price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePriceAsUnderlying, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_LastPrice: Latest price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. When the trading volume of the contract is zero for the day, the previous settlement price is used.

6.1.3.2 Stock index option margin

For stock index options traded in CFFEX.

The margin calculation formulas for call stock index options are:

$$\begin{aligned} \text{CallStockIndexOptionMargin} &= (\text{OptionPrice} \times \text{OptionInstrumentMultiplier} \\ &+ \max(\text{CallOptionBaseMargin} - \text{CallOptionOTM}(\text{out-of-money}), \\ &\text{MinimumBoundaryCoefficient} \times \text{CallOptionBaseMargin})) \times \text{Quantity} \end{aligned}$$

$$\text{CallOptionBaseMargin} = \text{UnderlyingInstrumentPrice} \times \text{UnderlyingInstrumentMultiplier} \times \text{CallOptionShortMarginRatePerAmount}$$

$$\text{CallOptionOTM} = \max((\text{ExercisePrice} - \text{UnderlyingInstrumentPrice}) \times \text{OptionInstrumentMultiplier}, 0)$$

The margin calculation formulas for put stock index options are:

$$\begin{aligned} \text{PutStockIndexOptionMargin} &= (\text{OptionPrice} \times \text{OptionInstrumentMultiplier} \\ &+ \max(\text{PutOptionBaseMargin1} - \text{PutOptionOTM}, \\ &\text{MinimumBoundaryCoefficient} \times \text{PutOptionBaseMargin2})) \times \text{Quantity} \end{aligned}$$

$$\text{PutOptionBaseMargin1} = \text{UnderlyingInstrumentPrice} \times \text{UnderlyingInstrumentMultiplier} \times \text{PutOptionShortMarginRatePerAmount}$$

$$\text{PutOptionBaseMargin2} = \text{ExercisePrice} \times \text{UnderlyingInstrumentMultiplier} \times \text{PutOptionShortMarginRatePerAmount}$$

$$\text{PutOptionOTM} = \max((\text{UnderlyingInstrumentPrice} - \text{ExercisePrice}) \times \text{OptionInstrumentMultiplier}, 0)$$

The minimum boundary coefficient is obtained when the parameter value (Name, Target) of YDSystemParam is (MarginLowerBoundaryCoef, MarginCalcMethod1). At present, the minimum boundary coefficient of stock index options should be 0.5.

For **orders**, the order volume is used for calculating the option order margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (OrderMarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_OrderPrice: When the order is a market price one, the upper limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail.
- YD_CBT_SamePrice: The same price used as that for the margin rate of option positions, namely the parameter value obtained when (Name, Target) of YDSystemParam is (MarginBasePrice, Options) is used. When the used margin rate of option positions is subject to YD_CBT_OpenPrice at this time, the price used for option order margin will be a pre settlement one

For **pre positions** and **today's positions**, the position volume is used for calculating the option position margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price.
- YD_CBT_OpenPrice: Open price. The margins corresponding to the position details is calculated one by one according to the open price.
- YD_CBT_LastPrice: Latest price
- YD_CBT_MarketAveragePrice: Average market price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. When the instrument's trading volume is zero for the day, the previous settlement price is used. It is a default configuration mode and the current mode used by the mainstream OMS

For **orders**, **pre positions** and **today's positions**, the position volume is used for calculating the option position margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Previous Settlement Price. It is a default configuration mode and the current mode used by the mainstream OMS.
- YD_CBT_LastPrice: Latest price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. When the trading volume of the instrument is zero for the day, the previous settlement price is used.

6.1.3.3 Stock option margin

In production, the margin models of stock options can be divided into linear and nonlinear ones. YD OMSs can help to realize these two margin models simultaneously by controlling the parameters of the calculation formula. For example, when the linear coefficient is 1.2 and other parameters are standard values of exchanges (at present, the margin rate of underlying instruments is 12%, and the minimum boundary coefficient is 7%), the linear model should be used; When the margin rate of underlying instruments is 15%, the minimum boundary coefficient is 8%, and the linear coefficient is 1, the non-linear model should be used.

The margin calculation formulas for call stock options are:

$$\text{CallStockOptionMargin} = \text{LinearCoefficient} \times (\text{OptionPrice} + \max(\text{UnderlyingInstrumentMarginRate} \times \text{UnderlyingInstrumentPrice} - \text{CallOTMLevel}, \text{MinimumBoundaryCoefficient} \times \text{UnderlyingInstrumentPrice})) \times \text{OptionInstrumentMultiplier} \times \text{Quantity}$$

$$\text{CallOTMLevel} = \max(\text{ExercisePrice} - \text{UnderlyingInstrumentPrice}, 0)$$

The margin calculation formulas for put stock options are:

$$\text{PutStockOptionMargin} = \text{LinearCoefficient} \times (\text{OptionPrice} + \min(\max(\text{UnderlyingInstrumentMarginRate} \times \text{UnderlyingInstrumentPrice} - \text{PutOTMLevel}, \text{MinimumBoundaryCoefficient} \times \text{ExercisePrice}), \text{ExercisePrice} - \text{LatestOptionPrice})) \times \text{OptionInstrumentMultiplier} \times \text{Quantity}$$

$$\text{PutOTMLevel} = \max(\text{UnderlyingInstrumentPrice} - \text{ExercisePrice}, 0)$$

For **orders**, the order volume is used for calculating the option order margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (OrderMarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_OrderPrice: When the order is a market price one, the upper limit price shall prevail; When the order is a price-limited one/FAK/FOK, the order price shall prevail.
- YD_CBT_SamePrice: The same price used as that for the margin rate of option positions, namely the parameter value obtained when (Name, Target) of YDSystemParam is (MarginBasePrice, Options) is used. When the used margin rate of option positions is subject to YD_CBT_OpenPrice at this time, the price used for option order margin will be a pre settlement one

For **pre positions** and **today's positions**, the position volume is used for calculating the option position margin. The option price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price.

- YD_CBT_OpenPrice: Open price. The margins corresponding to the position details is calculated one by one according to the open price.
- YD_CBT_LastPrice: Latest price
- YD_CBT_MarketAveragePrice: Average market price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS

For **orders**, **pre positions** and **today's positions**, the underlying instrument price is controlled when the parameter value (Name, Target) of YDSystemParam is (MarginBasePriceAsUnderlying, Options). The following shows how to handle different parameter values by the system:

- YD_CBT_PreSettlementPrice: Pre settlement price. It is a default configuration mode and the current mode used by the mainstream OMS
- YD_CBT_LastPrice: Latest price
- YD_CBT_MaxLastPreSettlementPrice: The higher one between the latest price and pre settlement price.

6.1.4 Option exercise margin

6.1.4.1 Commodity option exercise margin

The calculation formula for call option margin of commodity options is:

$$\text{CallCommodityOptionExerciseMargin} = ((\text{UnderlyingInstrumentPreSettlementPrice} \times \text{UnderlyingInstrumentMultiplier} \\ \times \text{UnderlyingInstrumentLongMarginRatePerAmount} + \text{UnderlyingInstrumentLongMarginRatePerLot}) \\ \times \text{UnderlyingOptionMultiplier} + \max(\text{ExercisePrice} - \text{UnderlyingInstrumentPreSettlementPrice}, \\ 0) \times \text{OptionInstrumentMultiplier}) \times \text{Quantity}$$

The calculation formula for put option margin of commodity options is:

$$\text{PutCommodityOptionExerciseMargin} = ((\text{UnderlyingInstrumentPreSettlementPrice} \times \text{UnderlyingInstrumentMultiplier} \\ \times \text{UnderlyingInstrumentShortMarginRatePerAmount} + \text{UnderlyingInstrumentShortMarginRatePerLot}) \\ \times \text{OptionUnderlyingMultiplier} + \max(\text{UnderlyingInstrumentPreSettlementPrice} - \text{ExercisePrice}, \\ 0) \times \text{OptionInstrumentMultiplier}) \times \text{Quantity}$$

The underlying multiplier is the UnderlyingMultiply of option instruments.

6.1.4.2 Stock option exercise margin

The calculation formula for exercise margin of call stock options is:

$$\text{CallStockOptionExerciseMargin} = \text{ExercisePrice} \times \text{OptionMultiplier}$$

No margin is required for the exercise of put stock options.

6.1.5 Margin deduction

The main deduction margins of exchanges are divided into one-way large-side margins and combined margins. Although their final effects are similar, their usages and complexities are different. The two deduction methods should be distinguished properly. At present, CFFEX, SHFE, INE and CZCE support the large-side margin service, while DCE, CZCE, SSE and SZSE support the combined margin service.

6.1.5.1 One-way large-side margin

CZCE supports **one-way large-side margin under the same instrument**, which is dependent on the sum of long/short speculation, hedging position and order margins under the same instrument, the margin which is relatively higher will be considered as the actual margin of this instrument to be collected.

SHFE and INE support **one-way large-side margin under the same product**, which is dependent on the sum of long/short speculation, hedging position and order margins under the same product, the margin which is relatively higher will be considered as the actual margin of this product to be collected.

CFFEX supports **cross-product one-way large-side margin**, which is dependent on the sum of long/short speculation, hedging position and order margins under the same product group, the margin which is relatively higher will be considered as the actual margin of this product group to be collected. Product groups are defined through "YDProduct.m_pMarginProduct", this field points to the leader product of a product group (the leader product is not fixed and may vary with the initialization data), the leader product and all m_pMarginProduct pointing to the leader product belong to the same product group. At present, CFFEX has two product groups. All stock index futures IF, IC, IH and IM belong to one product group, and all T-bond futures TF, TS and T belong to the other product group.

In order to make it easy for the strategy program to identify instruments that need to be involved in the calculation of one-way large-side margins, when "YDInstrument.SingleSideMargin" is "True", it means that an instrument should be involved in the calculation of a one-way large-side margin, otherwise it will not be involved. Generally, the SingleSideMargin of futures instruments of SHFE, INE and CFFEX are "True", but because the futures instruments that are close to delivery should not be involved in the calculation of one-way large-side margins, they should be set as "False" ones through the SingleSideMargin. The settings of this part are made based on the preday data from the mainstream OMS.

6.1.5.2 Traditional combined margin

YD has developed different combined margin deduction methods for different combination types of exchanges, which are introduced one by one below.

6.1.5.2.1 Futures combination

The following combination types apply to this deduction algorithm:

- YD_CPT_DCE_FuturesOffset: Futures offset of DCE
- YD_CPT_DCE_FuturesCalendarSpread: Futures calendar spread of DCE
- YD_CPT_DCE_FuturesProductSpread: Futures cross-product spread of DCE
- YD_CPT_GFEX_FuturesOffset: Futures offset of GFEX
- YD_CPT_GFEX_FuturesCalendarSpread: Futures calendar spread of GFEX
- YD_CPT_GFEX_FuturesProductSpread: Futures cross-product spread of GFEX
- YD_CPT_CZCE_Spread: Spread of CZCE

A combined futures margin can be obtained by selecting a small side according to the rules first, making the selected small side position margin be a savings margin and then subtracting the savings margin from the sum of the original two-leg margins. The selection rules for small sides are:

- Compare the two-leg futures margins of exchanges and select the smaller side. If they are the same, go to the next step. The calculation formula for exchange futures margins is the same as that for [Futures Margin](#). The pre settlement price and exchange margin rate are used for the calculation. The exchange margin rate can be obtained through "YDInstrument.m_pExchangeMarginRate".
- If combination type is YD_CPT_CZCE_Spread, select the right leg directly, otherwise go to the next step
- Compare the delivery months in relation to the two legs. The farther month side should be selected. If they are the same, go to the next step
- Compare the two-leg instrument codes. The longer instrument code should be selected. An instrument code comparison means a string comparison

6.1.5.2.2 Option straddle

The following combination types apply to this deduction algorithm:

- YD_CPT_DCE_OptionsStraddle: Put option straddle of DCE
- YD_CPT_DCE_OptionsStrangle: Put option strangle of DCE
- YD_CPT_GFEX_OptionsStraddle: Put option straddle of GFEX
- YD_CPT_GFEX_OptionsStrangle: Put option strangle of GFEX
- YD_CPT_CZCE_StraddleStrangle: Put option straddle or strangle of CZCE

The above mentioned combined option margin can be obtained by selecting a small side according to the rules first, calculating the savings margin through the selected small side and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is: $\text{Exchange option margin} \times \text{position volume}$. The calculation formula of the exchange option margin is the same as that of [Option Margin](#), the pre settlement price and exchange margin rate should be used for the calculation. The exchange margin rate can be obtained through "YDInstrument.m_pExchangeMarginRate". The small side selection rules are:

- Compare the two-leg option margins and select the smaller side. If they are the same, go to the next step;
- Calculate the two-leg savings margins, and select the lower savings margin as the final result.

The following combination types apply to this deduction algorithm:

- YD_CPT_StockOption_KS: Straddle short positions of SSE and SZSE
- YD_CPT_StockOption_KKS: Strangle short positions of SSE and SZSE

The above mentioned combined option margin can be obtained by selecting a small side according to the rules first, calculating the savings margin through the selected small side and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is:

$(\text{OptionMargin} - \text{OptionPreSettlementPrice} \times \text{OptionInstrumentMultiplier} \times \text{LinearCoefficient}) \times \text{PositionVolume}$.

The small side selection rules are:

- Compare the two-leg option margins and select the smaller side. If they are the same, go to the next step;
- Calculate the two-leg savings margins, and select the lower savings margin as the final result.

6.1.5.2.3 Sell option coverage

The following combination types apply to this deduction algorithm:

- YD_CPT_DCE_SellOptionsCovered: Sell options covered of DCE
- YD_CPT_GFEX_SellOptionsCovered: Sell options covered of GFEX
- YD_CPT_CZCE_SellOptionCovered: Sell options covered of CZCE

The combined margin can be obtained by calculating the savings margin relying on the left leg directly and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is: $\text{Exchange option margin} \times \text{position volume}$. The option price specified by the parameter value obtained when (Name, Target) of YDSystemParam is (MarginBasePrice, Options) is used in the formula. The calculation formula of the exchange option margin is the same as that of [Commodity Option Margin](#). The pre settlement price and exchange margin rate should be used for the calculation. The exchange margin rate can be obtained through "YDInstrument.m_pExchangeMarginRate".

6.1.5.2.4 Buy option coverage

The following combination types apply to this deduction algorithm:

- YD_CPT_DCE_OptionsOffset: Options offset of DCE
- YD_CPT_DCE_BuyOptionsVerticalSpread: Vertical spread of buy options of DCE
- YD_CPT_DCE_BuyOptionsCovered: Buy options covered of DCE
- YD_CPT_GFEX_OptionsOffset: Options offset of GFEX
- YD_CPT_GFEX_BuyOptionsVerticalSpread: Vertical spread of buy options of GFEX
- YD_CPT_GFEX_BuyOptionsCovered: Buy options covered of GFEX

When investors close the buy legs of the above combination, an additional margin will be required. If the available funds on the mainstream OMS are insufficient for the additional margin, the position closing service will be prohibited. YD believes that the prohibition of position closing will affect the release of risks and may cause higher risks instead, which also violates YD's principle of Position Closing without Fund Checking. Therefore, the available funds may become negative when buy legs are closed by YD. The above measure is at the discretion of brokers. If the brokers have any objection to this measure, they can disable the margin deduction function of this combination. After the function is disabled, combination is still allowed, but no margin deduction will be allowed, thus margin call liquidation will not be caused during position closing. If the brokers accept YD's measure, they can enable this function. For the initial installation, this function is disabled by default. Investors can check the function for being enabled when the parameter value of (Name, Target) of YDSystemParam is (PortfolioMarginConcession, DCELongOptionPortfolio).

If the deduction is enabled, the combined margin can be obtained by calculating the savings margin relying on the right leg directly and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is: $(1 - \text{combined margin discount}) \times \text{option margin} \times \text{position volume}$. The combined margin discount in the formula is "YDCombPositionDef.Parameter". Please refer to [Traditional combined Position Definition](#) for details.

6.1.5.2.5 Vertical spread of sell options

The following combination types apply to this deduction algorithm:

- YD_CPT_DCE_SellOptionsVerticalSpread: Vertical spread of sell options of DCE
- YD_CPT_GFEX_SellOptionsVerticalSpread: Vertical spread of sell options of GFEX

When investors close the buy legs of the above combination, an additional margin will be required. If the available funds on the mainstream OMS are insufficient for the additional margin, the position closing service will be prohibited. YD believes that the prohibition of position closing will affect the release of risks and can cause higher risks instead, which also violates YD's principle of Position Closing without Fund Checking. Therefore, the available funds may become negative when buy legs are closed by YD. The above measure is at the discretion of brokers. If the brokers have any objection to this measure, they can disable the margin deduction function of this combination. After the function is disabled, combination is still allowed, but no margin deduction will be allowed, thus margin call liquidation will not be caused during position closing. If the brokers accept YD's measure, they can enable this function. For the initial installation, this function is disabled by default. Investors can check the function for being enabled when the parameter value of (Name, Target) of YDSystemParam is (PortfolioMarginConcession, DCELongOptionPortfolio).

If the deduction is enabled, the combined margin can be obtained by calculating the savings margin relying on the right leg directly and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is:

$$\max(\text{RightLegOptionMargin} - |\text{LeftLegExercisePrice} - \text{RightLegExercisePrice}| \times \text{LeftLegInstrumentMultiplier}, 0) \times \text{PositionVolume}$$

6.1.5.2.6 Bull and bear spreads

The following combination types apply to the deduction algorithms below:

- YD_CPT_StockOption_CNSJC: Bull call spreads of SSE and SZSE
- YD_CPT_StockOption_PXSJC: Bear put spreads of SSE and SZSE

The combined margin can be obtained by calculating the savings margin relying on the right leg directly and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is:

Right leg margin per lot \times position volume. The right leg margin per lot means the actual right leg margin, refer to [Stock Option Margin](#) for the specific calculation method.

The following combination types apply to the deduction algorithms below:

- YD_CPT_StockOption_CXSJC: Bear call spreads of SSE and SZSE
- YD_CPT_StockOption_PNSJC: Bull put spreads of SSE and SZSE

The combined margin can be obtained by calculating the savings margin relying on the right leg directly and then subtracting the savings margin from the sum of the original two-leg margins. The calculation formula for saving margins is:

$$(\text{RightLegMarginPerLot} - |\text{LeftLegExercisePrice} - \text{RightLegExercisePrice}| \times \text{RightLegInstrumentMultiplier} \times \text{RightLegLinearCoefficient}) \times \text{TraditionalCombinedPositionVolume}$$

The right leg margin per lot means the actual right leg margin, refer to [Stock Option Margin](#) for the specific calculation method.

6.1.6 Trial calculation of margins

The calculation of margins and combined margin deductions is relatively complex. In order to facilitate investors using ydExtendedApi to pre-calculate margins at a specified price, YD provides a series of methods.

6.1.6.1 Trial calculation of order margins

Investors can obtain the futures margin per order lot at any time through the following method. The trial calculation does not involve the large-side margin except the order margin. This method is not apply to combined instruments.

```
1 | virtual double getMarginPerLot(const YDInstrument *pInstrument, int hedgeFlag, int anyDirection, double
   | openPrice, const YDAccount *pAccount=NULL)
```

The parameters involved in the above method are as follows:

Parameter	Description
pInstrument	Instrument pointer
hedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. YD_HF_Hedge=3: Hedge
anyDirection	Trading direction, YD_D_Buy or YD_PD_Long can be used for representing "Buy", YD_D_Sell or YD_PD_Short can be used for representing "Sell"
openPrice	The price used is determined when the parameter value (Name, Target) of YDSystemParam is (MarginBasePrice, Options): YD_CBT_PreSettlementPrice: The pre settlement price shall always be used, openPrice is invalid YD_CBT_OpenPrice: The value specified by openPrice shall be used YD_CBT_LastPrice: The latest price shall always be used, openPrice is invalid YD_CBT_MarketAveragePrice: The average market price shall always be used, openPrice is invalid YD_CBT_MaxLastPreSettlementPrice: Usually, the higher value between the latest price and the previous settlement price is used. When the instrument's daily trading volume is zero, the previous settlement price is used. OpenPrice is invalid
pAccount	Set to NULL

Investors can obtain the option margin per order lot at any time through the following method. The price used for the trial calculation is selected by API according to the price settings.

```
1 | virtual double getOptionsShortMarginPerLot(const YDInstrument *pInstrument, int hedgeFlag, bool
   | includePremium, const YDAccount *pAccount=NULL)
```

The parameters involved in the above method are as follows:

Parameter	Description
pInstrument	Instrument pointer
hedgeFlag	Hedge flag. The definition of hedge flags is different for futures exchanges and stock option exchanges. The definition for futures exchanges is as follows: YD_HF_Speculation=1: Speculation YD_HF_Arbitrage=2: Arbitrage, only supported by CFFEX. YD_HF_Hedge=3: Hedge The definition for stock option exchanges is as follows: YD_HF_Normal=1: Normal YD_HF_Covered=3: covered
includePremium	Including a premium or not
pAccount	Set to NULL

6.1.6.2 Trial calculation of position margins

The following method can be used for calculating the margin per position lot under different prices. A full position margin can be obtained by multiplying it by the position volume. For futures positions, the large-side margin is not calculated. Unlike the trial calculation of order margin, the openPrice here is not affected by YDSystemParam, and the price specified by openPrice in the parameters is directly used for calculating the underlying price.

```
1 | virtual double getMarginPerLot(const YDExtendedPosition *pPosition, double openPrice)
```

6.1.6.3 Trial calculation of combined margins

The following method can be used for trial calculation for a combination which does not exist. The pre settlement price is used for the trial calculation, so it may be different from the actual margin to be collected after combination.

```
1 | virtual double getCombPositionMarginPerLot(const YDCombPositionDef *pCombPositionDef, const YDAccount
   | *pAccount=NULL)
```

The following method can help to obtain the true two-leg margin, savings margin and combined margin in relation to position details, Combined margin = left-leg margin + right-leg margin – savings margin, where the left/right-leg margins are legMargins[0] and legMargins[1], respectively, and the savings margin is the return value based on this method.

```
1 | virtual double getCombPositionMarginSaved(const YDExtendedCombPositionDetail *pCombPositionDetail, double  
   | legMargins[])
```

6.2 Portfolio margin model

Due to the specific logic of margin calculation algorithms being publicly disclosed by various exchanges, this article will not further discuss it. For more details, please refer to the relevant documents provided by the exchanges.

The YD portfolio margin model is consistent with CTP, and the following will introduce the common and differential parts of the portfolio margin model.

6.2.1 General concepts of portfolio margin model

6.2.1.1 Investor margin coefficient

Similar to traditional margin, brokers multiply the investor margin coefficient (typically greater than 1) with the exchange portfolio margin to determine the frozen margin for investors during trading, in order to control risks in extreme situations. In the traditional margin model, the investor margin coefficient can be set at the product level. However, in the portfolio margin model, the investor margin coefficient is set at the portfolio margin model level. The investor margin coefficient is derived from CTP's initial data and can be obtained through 'YDAccountMarginModelInfo.MarginRatio'. It can be dynamically adjusted during trading hours. For more details, please refer to [Account Combination Margin Parameters](#).

Therefore, in the portfolio margin model, the margin formula for investors in a specific portfolio margin model is as follows:

Investor Margin = Exchange Margin × Investor Margin Coefficient

6.2.1.2 Closing position check

In some portfolio margin models, additional margin may be required upon closing a position. If the available funds are insufficient to cover the additional margin, it may result in fund overdraft. To address this, YD provides a Closing Position Check mode.

The Closing Position Check mode is used to control whether YD counter checks the available funds upon receiving a closing order. This parameter is derived from CTP's initial data and can be obtained through 'YDAccountMarginModelInfo.CloseVerify'. It can be dynamically adjusted during trading. For more details, please refer to [account-combination-margin-parameters](#). The currently supported modes are as follows:

- YD_CV_NotVerify: No Verification
- YD_CV_Verify: Verification. The verification logic is as follows:
 - If it results in an increase in available funds, regardless of whether the available funds are less than 0 at this point, it is approved. Otherwise, continue to evaluate.
 - If the portfolio margin model specifically requests no verification for the current situation, it is approved. Otherwise, continue to evaluate.
 - If the available funds, after deducting the additional margin, are greater than or equal to 0, it is approved. Otherwise, it is not approved.

6.2.1.3 Applicable range

In some portfolio margin models, a subset of products can be set as the applicable range for investors in that portfolio margin model. For example, the products that are applicable for the SPBM model include MA, PF, and SR, but it is possible to set that a particular investor only uses MA and PF. The applicable range is derived from CTP's preday data and can be obtained through 'YDAccountMarginModelInfo.ProductRange'. It cannot be dynamically adjusted during trading. For more details, please refer to [Account Portfolio Margin Parameters](#).

For products that belong to the portfolio margin model but are not applicable to a specific investor, the traditional margin model is still used to calculate the margin for those products.

6.2.2 CZCE SPBM

Based on the standard model of SPBM, the following revisions are made according to the business logic of the mainstream OMS.

6.2.2.1 Freezing additional margin

According to the SPBM standard model, margin reduction is possible upon order submission. To avoid such issues, when the discount ratio of the intracommodity lock fee rate within a product family (IntraRateY) is less than 1, an additional margin freeze will be imposed. The calculation formula for the freezing of additional margin is as follows:

$$\text{FreezingAdditionalMargin} = (\text{IntraCommodityHedgeMargin} - \min(\text{LongPositionMargin}, \text{ShortPositionMargin})) \times (1 - \text{IntraRateY})$$

The aforementioned long position margin and short position margin are calculated based on the SPBM standard model, where the calculation of buying margin and selling margin for a commodity only includes the portion held in positions.

Similarly, a similar issue arises when the discount ratio of the intracommodity lock fee rate within a product family (IntraRateZ) is less than 0.5. However, considering the complexity of the calculation and the fact that there is currently no such situation in the SPBM production parameters, it is not implemented at the moment.

6.2.2.2 Closing position verification

In the closing position verification, this portfolio margin model does not require verification for closing long positions in options.

6.2.2.3 Exercise margin

When exercising and abandoning automatic exercise, the corresponding position needs to be deducted from the value of the long option, using the same deduction method as closing the option. For exercise requests, exercise margin needs to be frozen simultaneously.

$$\text{ExerciseMargin} = \text{ExerciseQuantity} \times (\text{UnderlyingMarginPerLot} + \text{OptionOTM})$$

$$\text{UnderlyingMarginPerLot} = \text{InvestorMarginCoefficient} \times \text{InstrumentMultiplier} \times \text{PreSettlementPrice} \\ \times (\text{InstrumentMarginRate} + \text{IncreasedInstrumentMarginRate})$$

Please refer to the content in the [Option margin](#) section for the calculation formula for out-of-the-money options. In the calculation of SPBM's exercise margin, the out-of-the-money options always use the previous settlement price.

6.2.3 DCE RULE

Based on the Rule standard model, the following revisions are made according to the business logic of the mainstream OMS.

6.2.3.1 Exercise margin

When exercising and abandoning automatic exercise, the portfolio margin is recalculated using the closing position freeze method. The exercise margin per lot is 0.

6.2.4 SHFE and INE SPMM

6.2.4.1 Close-out freezing margin

The closing freeze margin is calculated separately for each commodity group. First, the closing order's pure price risk is calculated separately for long and short positions within the commodity group, using the same method as for execution. Then, the sum of the pure price risk for long closing orders (sell to close order) and the sum of the pure price risk for short closing orders (buy to close order) within the commodity group are accumulated. Finally, the closing freeze margin for a single commodity group is obtained using the following formula. The closing freeze margins for all commodity groups are then added together to obtain the investor's total closing freeze margin, which is included as an addition to the SPMM investor's margin as the final margin.

$$\text{ClosingCommodityGroupFrozenMargin} = \max(\text{InCommPref} - 1, 0) * [\min(L1, S1) - \min(L1 - L2, S1 - S2)]$$

The symbols in the above formula are defined as follows:

- InCommPref: inter-period discount factor
- L1: pure price risk of long position in commodity portfolio.
- S1: pure price risk of short position in commodity portfolio.
- L2: The sum of pure price risks of closing out long positions in the commodity portfolio (sell to close order)
- S2: The sum of pure price risks of closing out short positions in the commodity portfolio (buy to close order)

6.2.5 CFFEX RCAMS

The combined positions of arbitrage accounts are not supported. If an investor applies for combined positions of arbitrage accounts, it will be automatically discarded by the counter.

7 Trade

7.1 Normal trading service

7.1.1 Orders

Order instructions are the most common instructions. Investors can call an order instruction to initiate a futures and options trade. Instructions that can be used for orders are as follows.

```
1 virtual bool insertOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
   *pAccount=NULL);
2 virtual bool insertMultiOrders(unsigned count, YDInputOrder inputOrders[], const YDInstrument
   *instruments[], const YDAccount *pAccount=NULL);
3
4 // only available in ydExtendedApi
5 virtual bool checkAndInsertOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
   *pAccount=NULL);
6 virtual bool checkOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
   *pAccount=NULL);
```

7.1.1.1 Normal orders

First, the order instruction "insertOrder" is the most normal instruction. If a "False" regarding this function is sent back, it means that the network to an OMS is interrupted.

```
1 virtual bool insertOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
   *pAccount=NULL);
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For providing YD_YOF_Normal information
	OrderRef	Customer order reference No. The OMS can show notified customer order reference numbers, and investors can match their orders with the notifications.
	Direction	YD_D_Buy: Buy YD_D_Sell: Sell
	OffsetFlag	For Normal instruments: YD_OF_Open: Open position YD_OF_Close: Close position, which are unapplicable to SHFE and INE. If used in SHFE and INE, they will be regarded as YD_OF_CloseYesterday YD_OF_CloseYesterday: Close pre position, which is applicable to SHFE and INE. It will be regarded as YD_OF_Close YD_OF_CloseToday: Close today's position if not used in SHFE and INE. It is applicable to SHFE and INE. It will be regarded as YD_OF_Close if not used in SHFE and INE. For Combined instruments: YD_OF_Open: Open the left and right legs at the same time YD_OF_Close: Close the left and right legs at the same time YD_OF_Open1Close2: Open the left leg and close the right leg YD_OF_Close1Open2: Close the left leg and open right leg
	HedgeFlag	YD_HF_Speculation: Speculation YD_HF_Arbitrage: Arbitrage, only supported by CFFEX YD_HF_Hedge: Hedge
	OrderType	YD_ODT_Limit: Price-limited order YD_ODT_Market: market order YD_ODT_FAK: FAK order YD_ODT_FOK: FOK order
	Price	Providing price information. The price shall not exceed the upper/lower limits. YD does not control the price fluctuation zone
	OrderVolume	The open and close position volumes are subject to the following: They should not be higher than the MaxMarketOrderVolume and MaxLimitOrderVolume of instruments They should not be lower than MinMarketOrderVolume and MinLimitOrderVolume of instruments They should not be higher than the position limit. Note: For SSE and SZSE, the combined position volume is excluded

Parameter	Field	Description
	OrderTriggerType	Trigger type YD_OTT_NoTrigger: No trigger YD_OTT_TakeProfit: Profit taking trigger YD_OTT_StopLoss: Loss stopping trigger At present, triggered orders of DCE and GFEX are supported
	TriggerPrice	Trigger price
	ExchangeOrderAttribute	The order attributes of the exchange, which shall be interpreted by the exchange to which the order belongs.
YD_EOA_DCE_GIS=1: DCE GIS(Good in Session).		
YDInstrument		For specifying a trading instrument pointer
YDAccount		The given "NULL" means that the current API login account is used

The parameters of spot orders are as follows:

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For providing YD_YOF_Normal
	OrderRef	Customer order reference No. The OMS can show notified customer order reference numbers, and investors can match their orders with the notifications.
	Direction	YD_D_Buy: Buy YD_D_Sell: Sell
	OffsetFlag	When direction is YD_D_Buy, use YD_OF_Open When direction is YD_D_Sell, use YD_OF_Close
	HedgeFlag	Use YD_HF_Speculation
	OrderType	YD_ODT_Limit: Price-limited order YD_ODT_Market: market order YD_ODT_FAK: FAK order YD_ODT_FOK: FOK order
	Price	Providing price information. The price shall not exceed the upper/lower limits. YD does not control the price fluctuation zone
	OrderVolume	The open and close position volumes are subject to the following: They should not be higher than the MaxMarketOrderVolume and MaxLimitOrderVolume of instruments They should not be lower than MinMarketOrderVolume and MinLimitOrderVolume of instruments They should not be higher than the position limit. 1 Buy/Sell quantity always means 1 share of stock, 1 fund and 1 bond
	OrderTriggerType	Use YD_OTT_NoTrigger
	TriggerPrice	Use 0
	ExchangeOrderAttribute	Use 0
YDInstrument		For specifying a trading instrument pointer
YDAccount		The given "NULL" means that the current API login account is used

Tip

In spot trading, stocks are traded in units of shares, and funds are traded in units of shares. One trading quantity of stocks represents one share, and one trading quantity of funds represents one fund. For bonds, the Shenzhen Stock Exchange uses sheets as units, and one trading quantity of bonds represents one bond, but the Shanghai Stock Exchange uses ten sheets as units, and one trading quantity of bonds represents ten bonds.

To provide a unified user experience, the Shanghai and Shenzhen Stock Exchange bonds traded in the YD OMS are all traded in sheets, that is, one trading quantity of bonds represents one bond, and the YD OMS is responsible for the conversion of quantity relations between the exchange and other systems.

The OrderType defined in the YD system is the combination of exchange order type conditions. The combination being the closest to YD's order types for each exchange should be set. The following shows the comparison relationship for each exchange:

Exchange	YD_ODT_Limit	YD_ODT_FAK	YD_ODT_Market	YD_ODT_FOK
CFEX	FFEX_FTDC_TC_GFD FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_AnyPrice FFEX_FTDC_VC_AV	FFEX_FTDC_TC_IOC FFEX_FTDC_OPT_LimitPrice FFEX_FTDC_VC_CV

Exchange	YD_ODT_Limit	YD_ODT_FAK	YD_ODT_Market	YD_ODT_FOK
SHFE	SHFE_FTDC_TC_GFD SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_AV	SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_AV	Not supported. Before use, it should be converted to: SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_AnyPrice SHFE_FTDC_VC_AV	SHFE_FTDC_TC_IOC SHFE_FTDC_OPT_LimitPrice SHFE_FTDC_VC_AV
DCE	OT_LO OA_NONE	OT_LO OA_FAK	OT_MO OA_FAK	OT_LO OA_FOK
CZCE	FID_OrderType=0 // Limited price FID_MatchCondition=3 // Valid on the current day	FID_OrderType=0 // Limited price FID_MatchCondition=2 // Immediate partial traded	FID_OrderType=1 // Market price FID_MatchCondition=2 // Immediate partial traded	FID_OrderType=0 // Price FID_MatchCondition=1 // Immediate all traded
GFEX	OT_LO OA_NONE	OT_LO OA_FAK	OT_MO OA_FAK	OT_LO OA_FOK
SSE	OrderType=Limit TimeInForce=GFD	Not supported. It should be converted to YD_ODT_FOK before use	OrderType=Market TimeInForce=IOC	OrderType=Limit TimeInForce=FOK
SZSE	OrderType=Limit TimeInForce=GFD MinQty=0	Not supported. It should be converted to YD_ODT_FOK before use	OrderType=Market TimeInForce=IOC MinQty=0	OrderType=Limit TimeInForce=GFD MinQty=OrderVolume
SSE Stock Options Platform	OrderType=Limit TimeInForce=GFD	It should be converted to YD_ODT_FOK before use.	OrderType=Market TimeInForce=IOC	OrderType=Limit TimeInForce=FOK
SSE Trading Platform	OrderType=Limit	Not supported	OrderType=Market	Not supported
SSE Bond Platform	OrderType=Limit	Not supported	Not supported	Not supported
SZSE Stock Option	OrderType=Limit TimeInForce=GFD MinQty=0	Not supported. It should be converted to YD_ODT_FOK before use	OrderType=Market TimeInForce=IOC MinQty=0	OrderType=Limit TimeInForce=GFD MinQty=OrderVolume
SZSE Spot	OrderType=Limit TimeInForce=GFD MinQty=0	Not supported		

7.1.1.2 Multi-orders

Multi-orders include not more than 16 orders each time, meeting customers' demand for simultaneous multi-leg order submission. Compared with repeated common report calling, it is not superior in overall performance. It is not suggested to select multi-orders for performance purposes. The following shows the analysis from both sending and receiving:

- When sent by a client, the advantage of multi-orders is that the number of API calls can be reduced, but the sending is only allowed after all orders are prepared, while normal orders can be prepared one by one and sent immediately, which is more efficient considering resource utilization; In addition, multi-orders are more time-consuming since being always sent from large packages.
- When receiving at an OMS, multi-orders are also received and processed in sequence, so the processing performance is basically the same as that for multiple normal orders

```
1 virtual bool insertMultiOrders(unsigned count, YDInputOrder inputOrders[], const YDInstrument
  *instruments[], const YDAccount *pAccount=NULL)
```

When all multi-orders are sent, a "True" will be sent back. When part of or multi-orders are not sent, a "False" will be sent back. The parameters can be given according to the following method:

Parameter	Description
unsigned count	Count of orders: 16 at most
YDInputOrder inputOrders[]	Order arrays corresponding to instrument pointer arrays
YDInstrument *instruments[]	Trading instrument pointer array
YDAccount	The given "NULL" means that the current API login account is used

After receiving the multi-orders, the processing method of YD OMSs for each order is the same as that of normal single orders: namely, each order means an independent trade, and successfully processed orders will not be returned as a whole even when risk control of the subsequent order fails, and those false orders in the front will not affect the processing of the subsequent orders. Therefore, the notification and cancellation in relation to multi-orders are the same as those to ordinary orders, except that there will be multiple notification callbacks.

7.1.1.3 Local risk control orders

In ydExtendedApi, a series of instructions for risk control and order submission at the client are added. These instructions have been additionally provided with local money position and risk control checks to the corresponding orders. If the risk control fails, a "False" will be sent back directly. The error reason can be obtained from ErrorNo under YDInputOrder. Compared with original order instructions, when an instruction is easier to be intercepted by the risk control function at an OMS, the local risk control order instruction can help to more quickly show whether the orders can pass the money position and risk control checks at the OMS with **a high probability**, which facilitates the communication with the OMS and, correspondingly, increases the time cost of orders at the client. Therefore, the corresponding order instructions shall be chosen according to the actual conditions.

```
1 virtual bool checkAndInsertOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
  *pAccount=NULL)
```

When the local risk control function is used for order submission, the OrderRef under YDInputOrder is coded by the API from 1. Even if a user assigns a value to OrderRef, it will be overwritten by the API. The OrderRef encoding mechanism makes use of the connection number encoding mechanism of ydExtendedApi, namely, local risk control and order submission naturally support multiple connection numbers. Please refer to [Multi connection](#) for details. After the call function is notified, the OrderRef used for order submission can be found in the incoming YDInputOrder.OrderRef. If the checks fail, the reason for the failure of risk control and order submission can also be found in YDInputOrder.ErrorNo.

Orders passing the local risk control checks do not mean that they can pass the risk control check at the OMS, which may be caused by, but are not limited to the following reasons:

- The market fluctuates greatly, and the time for position profit/loss and margin refresh through the OMS is inconsistent with that through the client, resulting in insufficient funds at the OMS though sufficient at the client;
- If investors conduct trade via the same account relying on more than one strategy, the OMS check may fail due to the impact of other strategy orders;
- Some risk control measures are only checked at the OMS.

When using the multi-order instructions regarding local risk control, the orders can only be submitted after passing the risk control check, which is different from the processing behavior of the multi-orders at the OMS.

Investors can also use the following instructions for checking without submitting orders:

```
1 | virtual bool checkOrder(YDInputOrder *pInputOrder, const YDInstrument *pInstrument, const YDAccount
    *pAccount=NULL)
```

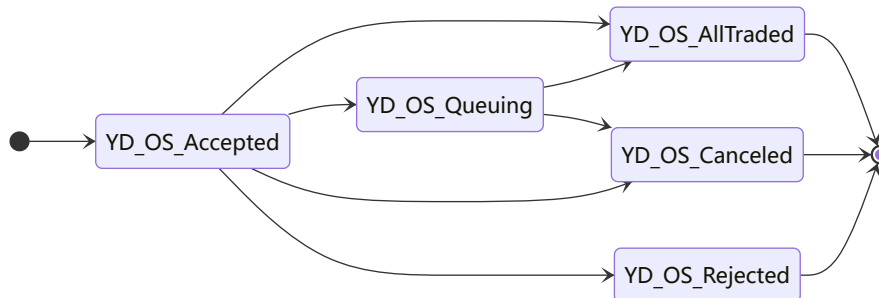
7.1.1.4 Order notification

All correct or incorrect exchange notifications will be sent back through notifyOrder. After receiving a notification, if the value of "YDOrder.ErrorNo" is 0, it will be process as a correct one; If the value of "YDOrder.ErrorNo" is not 0, it will be processed as a incorrect one.

```
1 | virtual void notifyOrder(const YDOrder *pOrder, const YDInstrument *pInstrument, const YDAccount *pAccount)
```

If an order passes the OMS risk control check and is sent to an exchange, the notification will not be sent to the investor by default before receiving the notification from the exchange. However, the investor can ask the broker to enable the OMS notification function to receive the notification from the OMS. The investor can check the OMS notification function for being enabled depending on whether YD_AF_NotifyOrderAccept is set through AccountFlag of YDAccount. The OMS notification can be used as a credential for the OMS receiving the investor's order application. Investors worrying about missing of UDP orders can check for missed UDP package earlier through this mechanism. OMS notifications are sent back through notifyOrder, whose OrderStatus is YD_OS_Accepted. Except OrderSysID and InsertTime, other fields have been assigned correctly and can be used normally. Except those RFQ orders and option hedging instructions of DCE and GFEX, OMS notifications in relation to all other orders using the insertOrder series order instructions can be received.

When receiving notifications from an exchange, YD OMS will send order notifications to investors through notifyOrder accordingly. The status, quantity and sequence of notifications are related to the specific behavior of the exchange. For example, the YD_OS_Queueing statuses of some exchange FAK and FOK orders will not be sent back except final order notifications. For another example, if the price-limited orders of CFFEX are all traded when involved in the matching queue for the first time, only an all-traded notification rather than a queueing notification will be sent back. The exchanges do not and will not commit to the behavior of sending notifications and have the right to change this behavior at any time. However, both YD and exchanges ensure that the notification status is always transferred depending on the order status machine. Therefore, investors should not rely on the notification behavior of exchanges when developing strategy programs but should ensure that they can correctly handle regardless of the notification sequence received. The status machine diagram of orders is shown below.



The parameters sent back by notifyOrder are described as follows. The fields sent back through YDInputOrder will not be described:

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed to YD_YOF_Normal
	TradeVolume	Accumulated order trade volume
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	For futures exchanges, it refers to a system order ID number sent back by an exchange For spot exchanges, it refers to a converted order ID number of a virtual exchange. To obtain the real exchange order ID numbers, refer to the following contents If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated

Parameter	Field	Description
	LongOrderSysID	For futures exchanges, it refers to a system order ID number sent back by an exchange For spot exchanges, it refers to a converted order ID number of a virtual exchange. To obtain the real exchange order ID numbers, refer to the following contents The exchange order ID number will not be truncated for the sake of a full-accuracy
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	CancelTimeStamp	Millisecond counts from the beginning (17:00) to the order cancellation time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	OrderTriggerStatus	Order trigger status YD_OTS_NotTriggered: Not triggered YD_OTS_Triggered: Triggered
	OrderStatus	The possible statuses and notification status machine are shown below: YD_OS_Accepted: the OMS accepted, which will only appear in OMS notifications YD_OS_Queueing: Queueing YD_OS_AllTraded: All traded YD_OS_Canceled: Canceled, including those orders traded or canceled YD_OS_Rejected: Failed orders of exchanges
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs
YDInstrument		Order Instrument pointer
YDAccount		Current API login account

Since the exchange order ID numbers, combination numbers and exercise numbers used by SSE and SZSE are not integers, YD cannot directly put these numbers sent back by exchanges into the OrderSysID field but fill in it with digitally converted native exchange numbers. The following two methods show native exchange numbers.

For investors using ydApi and ydExtendedApi, the native order ID numbers can be received through the notifyIDFromExchange callback function of YDListener. Such callback can only be received after the exchange number is converted, which will not be received when the trade is made in futures exchanges.

```
1 | virtual void notifyIDFromExchange(const YDIDFromExchange *pIDFromExchange, const YDExchange *pExchange)
```

The notified parameters are described as follows:

Parameter	Field	Description
YDIDFromExchange	IDType	ID type, possible values are as follows: YD_IDT_NormalOrderSysID: Normal order ID No. YD_IDT_QuoteDerivedOrderSysID: Quote Derived order ID No. YD_IDT_OptionExecuteOrderSysID: Option exercise order ID No. YD_IDT_OptionAbandonExecuteOrderSysID: option abandon exercise order ID No. YD_IDT_RequestForQuoteOrderSysID: RFQ order ID No. YD_IDT_CombPositionOrderSysID: Combined order ID No. YD_IDT_OptionExecuteTogether: Combined exercise order ID No. YD_IDT_Mark: Combined order ID No. YD_IDT_OptionSelfClose: Option Hedge ID No. YD_IDT_FreezeUnderlying: Bond Locking and Unlocking ID No. YD_IDT_Cover: Covered order ID No. YD_IDT_TradeID: Traded ID No. YD_IDT_CombPositionDetailID: Combination detail ID No. YD_IDT_QuoteSysID: Quote ID No.
	IDInSystem	The ID value converted by YD OMS may be truncated

Parameter	Field	Description
	LongIDInSystem	Full-accuracy ID value converted by YD OMS
	IDFromExchange	Exchange's native ID value, max. length: 24 bytes
YDExchange		Exchange pointer

For investors using `ydExtendedApi`, the native ID numbers can also be directly queried through `getIDFromExchange`. For the meaning of the parameters to be entered for query, see the table above. The maximum length of the native ID number sent back is 24 bytes.

```
1 | virtual const char *getIDFromExchange(const YDExchange *pExchange, int idType, int idInSystem)
```

7.1.1.5 Extended order notification

If `YDExtendedListener` is used by investors, notifications can be received through `notifyExtendedOrder` under the following circumstances:

- In case of a `notifyOrder` callback, regardless of correct or incorrect orders, regardless of orders rejected by the OMS or an exchange
- In case of an attribute change of `YDExtendedOrder`
- In case of a successful order submission and sending through `checkAndInsertOrder`

```
1 | virtual void notifyExtendedOrder(const YDExtendedOrder *pOrder)
```

Since `YDExtendedTrade` comes from `YDTrade`, exclusive fields of `YDExtendedTrade` are:

Field	Description
<code>m_pInstrument</code>	Order instrument pointer
<code>m_pCombPositionDef</code>	The order type is <code>YD_YOF_CombPosition</code> , which is valid for combining or decombining and defines the pointer for traditional combined positions
<code>m_pAccount</code>	Pointer of investor to which orders belong
<code>m_pInstrument2</code>	The order type is <code>YD_YOF_OptionExecuteTogether</code> , which is valid in case of combined exercise and is the pointer of the second instrument for combined exercise

7.1.1.6 Trade notification

When a trade is generated, a trade notification will be sent back through `notifyTrade`. The `OrderRef`, `OrderLocalID`, and `OrderSysID` of the trade notification can be used for correlating to a specific order.

```
1 | virtual void notifyTrade(const YDTrade *pTrade, const YDInstrument *pInstrument, const YDAccount *pAccount) {}
```

The return values of the trade notification are as follows, those fields being the same as those of orders will not be described again:

Parameter	Field	Description
YDTrade	TradeID	The trade ID number sent back by an exchange If the exchange order ID number exceeds the maximum length of the <code>TradeID</code> , part of it will be truncated
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	LongTradeID	Full-accuracy trade ID number sent back from an exchange
	OrderSysID	For futures exchanges, it refers to a system order ID number sent back by an exchange For spot exchanges, it refers to a converted order ID number of a virtual exchange. To obtain the real exchange order ID numbers, see the following contents If the exchange order ID number exceeds the maximum length of <code>OrderSysID</code> , part of it will be truncated
	LongOrderSysID	For futures exchanges, it refers to a system order ID number sent back by an exchange For spot exchanges, it refers to a converted order ID number of a virtual exchange. To obtain the real exchange order ID numbers, see the following contents The exchange order ID number will not be truncated for the sake of a full-accuracy
	OrderRef	Investor's order reference No.
	OrderGroupID	Order group ID number to which an order belongs
	Price	Trade price
	Volume	Trade volume

Parameter	Field	Description
	Commission	Trade commission
	TradeTime	Second counts from the beginning (17:00) to the trade time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between the TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h
	TradeTimeStamp	Millisecond counts from the beginning (17:00) to the trade time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
YDInstrument		Trade instrument pointer
YDAccount		Current API login account

7.1.1.7 Extended trade notification

If an investor uses YDExtendedListener, the notifyExtendedTrade will be called back in case of a notifyTrade callback:

```
1 | virtual void notifyExtendedTrade(const YDExtendedTrade *pTrade)
```

Since YDExtendedTrade comes from YDTrade, exclusive fields of YDExtendedTrade are:

Field	Description
m_pInstrument	Trade instrument pointer
m_pAccount	Investor pointer to which a trade belongs

7.1.2 Order cancellation

Investors can call the following function to cancel orders under queuing (the OrderStatus is YD_OS_Queueing).

```
1 | virtual bool cancelOrder(YDCancelOrder *pCancelOrder, const YDExchange *pExchange, const YDAccount
   | *pAccount=NULL)
2 | virtual bool cancelMultiOrders(unsigned count, YDCancelOrder cancelOrders[], const YDExchange
   | *exchanges[], const YDAccount *pAccount=NULL)
```

7.1.2.1 Normal order cancellation

The following method can be used to cancel a single order. If a "False" regarding this function is sent back, it means that the network to an OMS is interrupted.

```
1 | virtual bool cancelOrder(YDCancelOrder *pCancelOrder, const YDExchange *pExchange, const YDAccount
   | *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDCancelOrder	YDOrderFlag	For providing the type information of orders to be cancelled
	OrderSysID	Exchange order ID No.
	LongOrderSysID	Full-accuracy exchange order ID No.
	OrderGroupID	For specifying the logical group to which an order and a quote belong. When it is used together with OrderRef, OrderRef orders can be cancelled
	OrderRef	Investor's order reference No. When it is used together with OrderGroupID, OrderRef orders can be cancelled
YDExchange		Exchange for order cancellation
YDAccount		The given "NULL" means that the current API login account is used

The OMSs support three order cancellation modes, including LongOrderSysID, OrderSysID, and OrderRef. The OrderRef order cancellation mode allows investors to cancel orders before receiving notifications. Currently, it is supported by CFFEX, SHFE, INE, DCE, SSE and SZSE.

GFEX must use the information returned by the exchange to cancel an order, so it cannot support order cancellation before receiving a return. If an investor sends an OrderRef cancellation request to the GFEX counter before the counter receives a return from the exchange, the counter will reject it and return an error of YD_ERROR_ExchangeConnectionSendError=80; if an investor sends an OrderRef cancellation request to the GFEX counter after the counter receives a return from the exchange, the counter will use the return information to initiate an order cancellation request to the exchange. Therefore, there is uncertainty in GFEX's OrderRef cancellation, and it is not recommended.

The logic of the cancellation mode is as follows:

- First, judge the OrderGroupID. If the OrderGroupID is 0, continue to evaluate the value of LongOrderSysID. Otherwise, proceed with the logic for non-zero OrderGroupID.
 - If LongOrderSysID is not 0, the LongOrderSysID can be used for cancelling orders
 - If LongOrderSysID is 0, OrderSysID can be used for cancelling orders
- If OrderGroupID is not 0, OrderRef can be used for cancelling orders

7.1.2.2 Multi-order cancellation

Multi-order cancellation can help to cancel up to 16 orders each time, its characteristics are similar to [Multi-Orders](#).

```
1 virtual bool cancelMultiOrders(unsigned count, YDCancelOrder cancelOrders[], const YDExchange
  *exchanges[], const YDAccount *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Description
unsigned count	Count of orders to be cancelled: 16 at most
YDCancelOrder cancelOrders[]	Order cancellation information arrays corresponding to exchange pointer arrays
YDExchange *exchanges[]	Exchange pointer array corresponding to each order cancellation information
YDAccount	The given "NULL" means that the current API login account is used

7.1.2.3 Order cancellation notification

If orders are cancelled successfully, the latest statuses of those cancelled orders will be sent back through notifyOrder, please refer to [Order Notification](#) for details.

If the order cancellation fails, a notification will be sent back through the following callback regardless of the cancellation caused by the OMS rejection or the exchange rejection. Just check the ErrorNo of the YDFailedCancelOrder to obtain the reason for the cancellation failure. Generally, this error is made in that the orders have been traded or cancelled. **Please note that YD cannot ensure a strict correspondence between a cancellation failure notification and an investors' cancellation instruction in terms of volume and content. It is strongly suggested that investors only use the cancellation failure notifications for logging. The trading strategy should not be dependent on cancellation failure notifications but should establish a timeout mechanism after cancellation instructions are sent. If no notification on the corresponding change of order status is received within a set period of time, a new cancellation instruction should be sent**

```
1 virtual void notifyFailedCancelOrder(const YDFailedCancelOrder *pFailedCancelOrder, const YDExchange
  *pExchange, const YDAccount *pAccount)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDCancelOrder	AccountRef	Account reference No.
	ExchangeRef	Exchange reference No.
	YDOrderFlag	Type of order corresponding to cancellation error
	OrderSysID	Exchange order ID No.
	LongOrderSysID	Full-accuracy exchange order ID No.
	OrderGroupID	Order logic group ID
	OrderRef	Investor's order reference No.
	ErrorNo	Error No.
	IsQuote	Is it a notification for quote cancellation
YDExchange		Exchange with order cancellation error
YDAccount		Account for order cancellation error

Generally, the information sent back due to order cancellation failure corresponds to the method used for cancellation, namely:

- Filling in OrderSysID/LongOrderSysID and OrderGroupID/OrderRef is meaningless in notifications regarding order cancellation through OrderSysID and LongOrderSysID.
- Filling in OrderGroupID/OrderRef and OrderSysID/LongOrderSysID is meaningless in notifications regarding order cancellation through OrderGroupID and OrderRef.

However, under the following circumstances, the notification on failure of order cancellation using OrderGroupID and OrderRef is different from that mentioned above:

- Notifications on cancellation failures sent by exchanges will not be forwarded to investors.
- When an order notification has been sent back and errors such as flow control or network disconnection occur during cancellation, the OrderSysID and LongOrderSysID will be filled out in the order cancellation failure notification.

Therefore, when receiving a cancellation failure notification, the notified information should be checked. If OrderGroupID is 0, the corresponding order can be found through OrderSysID or LongOrderSysID; If OrderGroupID is not 0, the corresponding order can be found through OrderGroupID and OrderRef.

Generally, investors use OrderRef to establish an order index. When receiving a cancellation error notification with filled out OrderSysID and LongOrderSysID, the following method can be used to obtain the corresponding OrderRef and OrderGroupID:

```
1 virtual const YDExtendedOrder *getOrder(int orderSysID, const YDExchange *pExchange, int
   YDOrderFlag=YD_YOF_Normal)
2 virtual const YDExtendedOrder *getOrder(long long longOrderSysID, const YDExchange *pExchange, int
   YDOrderFlag=YD_YOF_Normal)
```

7.2 Covered service

7.2.1 Spot freezing and unfreezing

Before covered stock option service and normal-to-covered conversion service of SSE, the spot positions should be locked first. When SZSE carries out a covered service, its trading system is subject to an internal freezing service, so the covered service can be carried out directly without freezing in advance.

The freezing and unfreezing services should be initiated to exchanges through the insertOrder instruction according to the following parameter description.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_FreezeUnderlying, indicating that the order is subject to a spot freezing service
	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	Direction	YD_D_Freeze: Freeze YD_D_Unfreeze: Unfreeze
	OrderVolume	For providing the volume to be frozen and unfrozen
	HedgeFlag	For filling in YD_HF_Covered
YDInstrument		For specifying a spot instrument pointer to be frozen or unfrozen
YDAccount		The given "NULL" means that the current API login account is used

7.2.2 Covered open and close

The open and close services should be initiated to exchanges through the "insertOrder" instruction according to the following parameter description.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Normal
	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	Direction	YD_D_Buy: Buy YD_D_Sell: Sell
	OffsetFlag	YD_OF_Open: Open YD_OF_Close: Close
	HedgeFlag	For filling in YD_HF_Covered
	OrderType	For providing the following values as needed: YD_ODT_Limit: Price-limited Order YD_ODT_Market: Market Order YD_ODT_FOK: FOK Order
YDInstrument	Price	For providing a price
	OrderVolume	Volume for open and close
		For specifying an option instrument pointer for covered position opening and closing
YDAccount		The given "NULL" means that the current API login account is used

7.2.3 Normal-to-covered conversion

The normal-to-covered or covered-to-normal conversion service should be initiated to exchanges through the insertOrder instruction according to the following parameter description. It should be noted that both SSE and the SZSE support normal-to-covered conversion, but only SZSE supports covered-to-normal conversion.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Cover
	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	Direction	YD_D_Normal2Covered: Normal-to-covered YD_D_Covered2Normal: Covered-to-normal, only supported by SZSE
	HedgeFlag	For filling in YD_HF_Covered
	OrderVolume	Volume to be converted
YDInstrument		For specifying an option instrument pointer to be converted
YDAccount		The given "NULL" means that the current API login account is used

7.3 Market maker service

7.3.1 RFQ

For SHFE, INE, CFFEX, DCE, CZCE and GFEX, the RFQ should be sent through insertOrder according to the following parameter description. The RFQ must be made to an instrument allowing the request, otherwise it will be meaningless.

The SHFE and INE allow RFQ to be involved in the message count calculation. In order to prevent investors from unexpectedly exceeding the message count limit and being charged a high commission, YD provided [Message Count Risk Control](#).

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_RequestForQuote
	Direction	For filling in YD_D_Buy
	OffsetFlag	For filling in YD_OF_Open
	HedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge DCE and CZCE do not use this flag, the YD OMS uniquely determines the trading code using the capital account and the hedge flag, therefore please specify the corresponding hedge flag.
	OrderType	For filling in YD_ODT_Limit
YDInstrument		For specifying an instrument pointer for RFQ
YDAccount		The given "NULL" means that the current API login account is used

Even if an OMS notification YD_AF_NotifyOrderAccept function has been set for the AccountFlag of YDAccount, no OMS notification regarding the RFQ instruction will be sent.

The RFQ notification notifyRequestForQuote is only sent by the market maker system (a market maker flag is included in the OMS license file) in order to prevent those unnecessary data, and ordinary systems cannot receive RFQ notifications.

```
1 | virtual void notifyRequestForQuote(const YDRequestForQuote *pRequestForQuote, const YDInstrument *pInstrument)
```

The following shows a description of the information sent back from notifyRequestForQuote.

Parameter	Field	Description
YDRequestForQuote	RequestTime	Second counts from the beginning (17:00) to the RFQ time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between the integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	RequestForQuoteID	RFQ ID number. If being too long, the number length sent back will be truncated
	LongRequestForQuoteID	Full-accuracy RFQ ID number
YDInstrument		RFQ instrument pointer

In ydExtendedApi, YD provides a method for actively querying RFQ ID numbers as shown below:

```
1 | virtual const YDExtendedRequestForQuote *getRequestForQuote(const YDInstrument *pInstrument)
```

The return value YDExtendedRequestForQuote of the above method is a subclass of YDRequestForQuote, so the RequestTime and RequestForQuoteID can be read directly. If no quote is found after a query, the operation of the above method will return NULL.

7.3.2 Quote

YD supports the market maker quote service of SHFE, INE, CFFEX, DCE, GFEX, CZCE, SSE and SZSE. The quote instructions can only be used through YD OMSs with the market maker function enabled (a MarketMaker flag is included in the OMS authorization file). The current OMS can be checked for supporting the quote function depending on the MarketMaker flag of SystemParam. Please refer to [System Parameter](#) for details.

7.3.2.1 Normal quote

Market makers can send single quotes according to the following method. If a "False" is sent back through this function, it means that the network to the OMS has been interrupted.

```
1 | virtual bool insertQuote(YDInputQuote *pInputQuote, const YDInstrument *pInstrument, const YDAccount *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDInputQuote	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	BidOffsetFlag	Bid offset flag
	BidHedgeFlag	Bid hedge flag CFFEX: Actually, the exchange interface only supports the same hedge flag on both sides, which will not be checked by YD, however when submitting orders to the exchange, the customer ID number corresponding to the buyer's hedge flag should be used. Other exchanges: The buyer's and seller's hedge flags can be different
	AskOffsetFlag	Ask offset flag
	AskHedgeFlag	Ask hedge flag
	BidPrice	Bid price. The buying price must be lower than the selling price
	AskPrice	Ask price
	BidVolume	Bid volume. Depending on YDExchange.QuoteVolumeRestriction, different trade volume limits are provided: 0 (DCE, GFEX, SSE, SZSE): At least one of BidVolume and AskVolume is higher than 0, and the other is higher than or equal to 0 1 (CFFEX): BidVolume and AskVolume must be higher than 0, however they can be different 2 (SHFE, INE, CZCE): BidVolume and AskVolume must be equal and higher than 0
	AskVolume	Ask volume
	OrderRef	Customer's order reference No.
	YDQuoteFlag	Quote flag. This field is a bitmap that can simultaneously set multiple flags. YD_YQF_ResponseOfRFQ: For automatically filling in the RFQ number to indicate the response price. It is supported by SHFE, INE, DCE, GFEX and CZCE. Other exchanges do not provide RFQ numbers. YD_YQF_ReplaceLastQuote: Whether or not to top the last sent quote is only effective for CFFEX.
	ExchangeQuoteAttribute	the quote attributes of the exchange, which shall be interpreted by the exchange to which the quote belongs. YD_EOA_DCE_GIS=1 : DCE GIS(Good in Session).
YDInstrument		For specifying an instrument pointer to be quoted
YDAccount		The given "NULL" means that the current API login account is used

After initiating a quote request, in order to unify the quote service of all exchanges, the YD OMS will generate corresponding orders for the quote. The YDOrderFlag of these orders is YD_YOF_QuoteDerived: If involving a two-sided quote, two derived orders will be generated; If involving a one-side quote, only one derived order will be generated. Derived orders only exist in YD and will not be sent to exchanges. The traded quotes and order statuses will be shown on the derived orders, while the quotes themselves are provided without any status information. The relationship between the derived order fields YDOrder and quote YDQuote fields is as follows:

- The directions of derived buy/sell orders correspond to YD_D_Buy and YD_D_Sell, respectively
- The OffsetFlag of derived buy/sell orders corresponds to BidOffsetFlag and AskOffsetFlag, respectively
- The HedgeFlag of derived buy/sell orders corresponds to BidHedgeFlag and AskHedgeFlag, respectively
- The prices of derived buy/sell orders correspond to BidPrice and AskPrice, respectively

- The OrderVolume of derived buy/sell orders corresponds to BidVolume and AskVolume, respectively
- The OrderRef of derived buy/sell orders refers to the OrderRef of quotes
- The YDOrderFlag of derived buy/sell orders refers to YD_YOF_QuoteDerived
- The OrderLocalID of derived buy/sell orders is numbered according to the normal order sequence. The OrderLocalID of derived sell orders must be the OrderLocalID+1 of buy orders
- The OrderSysID of derived buy/sell orders correspond to BidOrderSysID and AskOrderSysID, respectively

7.3.2.2 Multi-quote

The multi-quote service can involve no more than 12 quotes each time, and its characteristics are similar to [Multi-Orders](#).

```
1 virtual bool insertMultiQuotes(unsigned count, YDInputQuote inputQuotes[], const YDInstrument
  *instruments[], const YDAccount *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Description
unsigned count	Count of quotes: 12 at most
YDInputQuote inputQuotes[]	Order arrays corresponding to instrument pointer arrays
const YDInstrument *instruments[]	Exchange point array corresponding to each quote
YDAccount	The given "NULL" means that the current API login account is used

7.3.2.3 Quote notification

All correct or incorrect exchange quote notifications will be sent back through notifyQuote. After receiving a notification, if YDQuote.ErrorNo is 0, it will be process as a true one; If YDQuote.ErrorNo is not 0, it will be processed as false one.

If the quote is made successfully, a notification will be received through the following callback functions, where:

- notifyQuote: Each quote corresponds to only one notification, indicating that the exchange has received the quote request, and its position among notifyOrder callbacks is not ensured;
- notifyOrder: This callback will be received when the status of a derived order changes, and its status machine is the same as that for normal orders;
- notifyTrade: The quote is the same as that for a normal order. A trade notification will be generated when a quote is accepted successfully. It can be associated with a derived order through OrderLocalID, or with a quote through OrderSysID.

```
1 virtual void notifyQuote(const YDQuote *pQuote, const YDInstrument *pInstrument, const YDAccount *pAccount)
  {}
2 virtual void notifyOrder(const YDOrder *pOrder, const YDInstrument *pInstrument, const YDAccount *pAccount)
  {}
3 virtual void notifyTrade(const YDTrade *pTrade, const YDInstrument *pInstrument, const YDAccount *pAccount)
  {}
```

Similar to [Order Notification](#), the order of 'notifyQuote' and 'notifyOrder' depends on the specific behavior of the exchange. For example, after receiving a two-sided quote, CFFEX first sends the 'notifyQuote' for the quote and then sends the 'notifyOrder' for the two derivative orders. On the other hand, SHFE first sends the 'notifyOrder' for the two derivative orders and then sends the 'notifyQuote' for the quote. The exchange does not promise or guarantee any behavior regarding quote feedback, and reserves the right to change such behavior at any time. Therefore, when developing strategy programs, investors should not rely on the exchange's feedback behavior and instead ensure that they can handle it correctly regardless of the order in which they receive any type of feedback.

Among them, notifyOrder and notifyTrade are the same as normal orders. The YDQuote of notifyQuote is a subclass of YDInputQuote. In addition to the regular RealConnectionID and ErrorNo, the new information for YDQuote is as follows:

Parameter	Field	Description
YDQuote	QuoteSysID	If ErrorNo is YD_ERROR_InvalidGroupOrderRef, it means that the maximum OrderRef has been received by current OMS; Otherwise, it means a quote number for quote cancellation. If the return value of the exchange is too long, it will be truncated.
	BidOrderSysID	The OrderSysID of the bid quote, which is 0 when selling a one-side quote. If the return value of the exchange is too long, it will be truncated.
	AskOrderSysID	The OrderSysID of the ask quote, which is 0 when buying a one-side quote. If the return value of the exchange is too long, it will be truncated.
	RequestForQuoteID	When YDQuoteFlag is YD_YQF_ResponseOfRFQ, the order response RFQ number will be recorded, otherwise will be 0. If the return value of the exchange is too long, it will be truncated.
	LongQuoteSysID	Full-accuracy quote number for quote cancellation
	LongBidOrderSysID	The OrderSysID of a full-accuracy bid quote, which will be 0 when selling a one-side quote

Parameter	Field	Description
	LongAskOrderSysID	The OrderSysID of a full-accuracy ask quote, which will be 0 when buying a one-side quote
	LongRequestForQuoteID	Full-accuracy. When YDQuoteFlag is YD_YQF_ResponseOfRFQ, the full-accuracy order response RFQ number will be recorded, otherwise will be 0
YDInstrument		Quote instrument pointer
YDAccount		current API login account pointer

7.3.2.4 Extended quote notification

If YDExtendedListener is used by investors, notifications can be received through notifyExtendedQuote under the following circumstances:

- In case of a notifyOrder callback, regardless of correct or incorrect quotes and regardless of orders rejected by the OMS or an exchange
- In case of an attribute change of YDExtendedQuote
- In case of a successful order submission and sending through checkAndInsertQuote

```
1 | virtual void notifyExtendedQuote(const YDExtendedQuote *pQuote)
```

Since YDExtendedQuote comes from YDQuote, exclusive fields of YDExtendedQuote are:

Field	Description
BidOrderFinished	For determining whether a derived order corresponding to a buy leg has been finished When the derived order is cancelled, completed or rejected, it is considered finished If the quote bid volume is 0, it will be directly considered finished
AskOrderFinished	For determining whether a derived order corresponding to a sell leg has been finished When the derived order is cancelled, completed or rejected, it is considered finished If the quote ask volume is 0, it will be directly considered finished
m_pInstrument	Quote instrument pointer
m_pAccount	Investor pointer to which the quote belongs

7.3.3 Quote cancellation

Investors can call the following method to cancel unhandled quotes.

```
1 | virtual bool cancelQuote(YDCancelQuote *pCancelQuote, const YDExchange *pExchange, const YDAccount
   | *pAccount=NULL)
2 | virtual bool cancelMultiQuotes(unsigned count, YDCancelQuote cancelQuotes[], const YDExchange
   | *exchanges[], const YDAccount *pAccount=NULL)
```

Based on the current implementation of the DCE, if the one leg's derivative orders are all traded or cancelled and another leg's are in the pending status, the quote cancellation order will cause exchange to return cancellation error to the OMS at first. The OMS will return cancellation error through notifyFailedCancelQuote, and then will notify that leg which in that pending status's derivative orders are cancelled. The OMS will return derivative orders' order return through notifyOrder. Investors are advised to avoid this problem.

7.3.3.1 Normal quote cancellation

The following method can be used for cancelling single quotes. If a "False" is sent back through this function, it means that the network to an OMS has been interrupted.

```
1 | virtual bool cancelQuote(YDCancelQuote *pCancelQuote, const YDExchange *pExchange, const YDAccount
   | *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDCancelQuote	QuoteSysID	Exchange quote number
	LongQuoteSysID	Full-accuracy exchange quote number
	OrderGroupID	For specifying the logical group to which the quote belongs, which can be used together with OrderRef for quote cancellation through OrderRef
	OrderRef	For using together with OrderGroupID for quote cancellation through OrderRef
YDExchange		Exchange pointer for quotes to be cancelled
YDAccount		The given "NULL" means that the current API login account is used

The OMSs support three quote cancellation modes, including LongOrderSysID, OrderSysID, and OrderRef. The OrderRef order cancellation mode allows investors to cancel orders before receiving notifications. Currently, it is supported by CFFEX, SHFE, INE, DCE, SSE and SZSE.

The cancellation of quotes by GFEX (including single-sided quotes and double-sided quotes) and the cancellation of double-sided quotes by DCE must use the information returned by the exchange, so it is not possible to support the cancellation of quotes before receiving returns. If an investor sends an OrderRef cancellation request to the GFEX or DCE counter before the counter receives the return from the exchange, it will be rejected by the counter and return the error YD_ERROR_ExchangeConnectionSendError=80; if an investor sends an OrderRef cancellation request to the GFEX or DCE counter after the counter receives the return from the exchange, the counter will use the return information to initiate a cancellation request to the exchange. Therefore, there is uncertainty in the OrderRef cancellation of quotes by GFEX (including single-sided quotes and double-sided quotes) and the cancellation of double-sided quotes by DCE, and it is not recommended.

The logic of the cancellation mode is as follows:

- First, judge the value of OrderGroupID. If OrderGroupID is 0, then continue to evaluate the value of LongQuoteSysID. Otherwise, proceed with the logic for non-zero OrderGroupID.
 - If LongQuoteSysID is not 0, the LongQuoteSysID can be used for cancelling quotes
 - If LongQuoteSysID is 0, OuoteSysID can be used for cancelling quotes
- If OrderGroupID is not 0, OrderRef can be used for cancelling quotes

7.3.3.2 Multi-quote cancellation

Multi-quote cancellation can help to cancel up to 16 orders each time, its characteristics are similar to [Multi-Orders](#).

```
1 | virtual bool cancelMultiQuotes(unsigned count, YDCancelQuote cancelQuotes[], const YDExchange
    | *exchanges[], const YDAccount *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Description
unsigned count	Count of quotes to be cancelled: 16 at most
YDCancelQuote cancelQuotes[]	Quote cancellation information arrays corresponding to exchange pointer arrays
YDExchange *exchanges[]	Exchange pointer array corresponding to each quote cancellation information
YDAccount	The given "NULL" means that the current API login account is used

7.3.3.3 Cancel Derivative Quote

DCE and GFEX support the cancellation of one side of the double-sided quote by cancelling derivative orders, but CFFEX, SHFE, INE, CZCE, SSE and SZSE do not support it. The method of cancelling derivative orders is the same as that of ordinary orders. Please refer to the corresponding content of [order cancellation](#). Similar to cancelling quotes, when cancelling one-leg derivative orders, there will only be an order callback notifyOrder, but no quote callback notifyQuote. Only when both legs of derivative orders are cancelled will the quote callback be returned.

7.3.3.4 Cancel Instrument Quote

Unlike futures exchanges, SSE and SZSE need to use the following method to cancel quotes by contract. If the function returns false, it means that the network to the YD OMS is interrupted.

```
1 | virtual bool cancelQuoteByInstrument(YDCancelQuote *pCancelQuote, const YDInstrument *pInstrument, int
    | cancelQuoteByInstrumentType, const YDAccount *pAccount=NULL)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDCancelQuote		Used to specify the order connection. Please do not fill in other business fields related to quote cancellation in the structure.
YDInstrument		The instrument pointer of instrument to cancel quote
cancelQuoteByInstrumentType		The method of cancelling quotes. It is only valid for SSE. SZSE can only cancel quotes from both sides at the same time.: YD_CQIT_Buy=1: Withdrawl buy side YD_CQIT_Sell=2: Withdrawl sell side YD_CQIT_Both=3: Withdrawl both side
YDAccount		Fill in NULL to use the current API login account

7.3.3.5 Quote cancellation notification

If quotes are cancelled successfully, the status of each derived order will be sent back through the notifyOrder callback. Please note that **there will not** be a notifyQuote callback when a quote is cancelled.

```
1 | virtual void notifyOrder(const YDOrder *porder, const YDInstrument *pInstrument, const YDAccount *pAccount)
    | {}
```

If the quote cancellation fails, the error information will be sent back through `notifyFailedCancelOrder` callback. At this time, the `ErrorNo` of `YDQuote` should be checked for error reasons. **Please note that YD cannot ensure a strict correspondence between a quote cancellation failure notification and an investors' cancellation instruction in terms of volume and content. It is strongly suggested that investors only use the quote cancellation failure notifications for logging. The trading strategy should not be dependent on quote cancellation failure notifications but should establish a timeout mechanism after cancellation instructions are sent. If no notification on the corresponding change of quote status is received within a set period, a new quote cancellation instruction should be sent**

```
1 virtual void notifyFailedCancelQuote(const YDFailedCancelQuote *pFailedCancelQuote, const YDExchange
  *pExchange, const YDAccount *pAccount)
```

The parameters of the above method are described as follows:

Parameter	Field	Description
YDCancelQuote	AccountRef	Account reference No.
	ExchangeRef	Exchange reference No.
	QuoteSysID	Exchange quote number
	LongQuoteSysID	Full-accuracy exchange quote number
	OrderGroupID	Order logic group ID
	OrderRef	Investor's order reference No.
	ErrorNo	Error No.
	IsQuote	Is it a notification for quote cancellation
YDExchange		Exchange with order cancellation error
YDAccount		Account for order cancellation error

Generally, the information sent back due to quote cancellation failure corresponds to the method used for quote cancellation, namely:

- Filling in `QuoteSysID/LongQuoteSysID` and `OrderGroupID/OrderRef` is meaningless in notifications regarding quote cancellation through `OrderSysID` and `LongOrderSysID`
- Filling in `OrderGroupID/OrderRef` and `QuoteSysID/LongQuoteSysID` is meaningless in notifications regarding quote cancellation through `OrderGroupID` and `OrderRef`

However, under the following circumstances, the notification on failure of quote cancellation using `OrderGroupID` and `OrderRef` is different from that mentioned above:

- Notifications on quote cancellation failures sent by exchanges will not be forwarded to investors
- When an order notification has been sent back and errors such as flow control or network disconnection occur during quote cancellation, the `QuoteSysID` and `LongQuoteSysID` will be filled out in the quote cancellation failure notification

Therefore, when receiving a quote cancellation failure notification, the notified information should be checked. If `OrderGroupID` is 0, the corresponding order can be found through `QuoteSysID` or `LongQuoteSysID`; If `OrderGroupID` is not 0, the corresponding order can be found through `OrderGroupID` and `OrderRef`.

Generally, investors use `OrderRef` to establish an order index. When receiving a quote cancellation error notification with filled out `QuoteSysID` and `LongQuoteSysID`, the following method can be used to obtain the corresponding `OrderRef` and `OrderGroupID`:

```
1 virtual const YDExtendedQuote *getQuote(int quoteSysID, const YDExchange *pExchange);
2 virtual const YDExtendedQuote *getQuote(long long LongQuoteSysID, const YDExchange *pExchange);
```

7.3.4 Quote modification

SHFE, INE, DCE, CZCE, CFFEX, SSE and SZSE support quote modification services (quote to cancel, QTC), namely a new quote under the same instrument can be made for replacing the previous one in order to reduce cancellations.

In CTP, SHFE and INE do not perform position verification when closing out a position. For example, a market maker has a position of 10 lots in a particular instrument. In the scenario where there is already an outstanding bid order to close 8 lots, the market maker can directly place a bid order to close 8 lots, achieving the effect of replacing the previous bid order. However, due to the lack of support for position verification exemption in closing out orders on other exchanges within CTP, YD does not provide special handling for position verification exemption in closing out orders on SHFE and INE. This is not conducive to market maker's unified order code. Therefore, YD does not support the excess position replacing order functionality on all exchanges. Please note that when the sum of the quantity in the outstanding quote orders and the quantity in the replacing orders is less than the total position, the submission of replacing orders is always allowed. For example, a market maker has a position of 10 lots in a particular instrument. In the scenario where there is already an outstanding bid order to close 4 lots, the market maker can directly place a bid order to close 4 lots, achieving the effect of replacing the previous bid order.

When selecting the quote modification service, the investors can directly submit a new quote according to the common quote method. The callback means the notification generated for one order cancellation and one quote. Therefore, refer to [Quote](#) and [Quote cancellation](#) for the relevant details.

CFFEX supports multi-level quoting, which means participants can submit quotes at multiple price levels. Therefore, CFFEX supports three options for replacing orders: not replacing, replacing with the last order, and replacing with a specified order. Given that other exchanges have not implemented multi-level quoting and specified replacing order functionality, the YD system offers only two choices: no replacement and replacement with a new order. When quoting, if the 'YDQuote.YDQuoteFlag' is set with the 'YD_YQF_ReplaceLastQuote' flag, it indicates replacing the last order; if not set, it indicates no replacement, which means multiple-level quoting will be generated. Previous quote orders can only be canceled by specifying cancellation, and cannot be replaced through the submission of a new order.

SSE and SZSE support two types of quote modification: double-side and single-side. In the case of double-side quote modification, the buy and sell quantities in the quote modification order must be greater than 0; in the case of single-side quote modification, the quantity of the side to be modified must be greater than 0, and the quantity of the other side must be equal to 0. In single-side quote modification, the exchange will traverse all quotes of the same contract in the pending state, and determine the impact on the quote modification order by its order type and direction gradually:

- If the quote being modified is a single-side quote and the direction is same as the quote modification order, the quote being modified will be cancelled.
- If the quote being modified is a single-side quote and the direction is opposite to the quote modification order:
 - If the prices cross, the quote being modified will be cancelled.
 - If the prices do not cross, the quote being modified will remain unchanged.
- If the quote being modified is a double-side quote:
 - If the side with the same direction as the quote modification order has been traded, it will remain unchanged.
 - If the side with the same direction as the quote modification order has not been traded, it will be cancelled.
 - If the side is opposite to the quote modification order has been traded, it will remain unchanged.
 - If the side is opposite to the quote modification order has not been traded:
 - If the prices cross, the quote being modified will remain unchanged.
 - If the prices do not cross, the quote being modified will remain unchanged.

7.4 Combination and decombination position service

DCE, GFEX, SSE and SZSE do not have provide large-side margin services and only provide deducted margin services relying on traditional combined positions. CZCE provides both one-way large-side margin services (under the same instrument) and deducted margin services relying on traditional combined positions.

For DCE and GFEX, positions can be combined automatically and traditionally for settlement, so they are combined before opening. If there are new positions that meet the definition of traditional combined positions and need to be combined during trading, a combination instruction can be used for combination. When closing positions, the DCE and GFEX can perform decombination automatically, and therefore no advanced decombination is needed.

For CZCE, positions can be combined automatically for covered call service before settlement, and other service types are allowed depending on a successful application. Therefore, positions that have been covered and successfully applied for before opening will be combined and the margin will be automatically deducted according to the offset (larger side under the same instrument) during trading. When closing, CZCE supports automatic decombination, and therefore no advanced decombination is needed.

For SSE and SZSE, automatic combination is not allowed for settlement unless combination instructions are used during trading. In SSE and SZSE, positions must be decombined before closing. Combined positions are not allowed for closing.

YD provides three different methods for sending combination instructions, which are:

- Native instruction: ydApi and ydExtendedApi users can call insertCombPositionOrder to send native combination and decombination instructions, while ydExtendedApi users can call checkAndInsertCombPositionOrder to send native combination and decombination instructions;
- Auto instruction: ydExtendedApi users can call autoCreateCombPosition to send an automatic combination instruction;
- Auto tool: For using the auto combination tool ydAutoCP to regularly send a combination instruction, which can be downloaded from [Here](#) or obtained from a broker.

7.4.1 Native instruction

ydApi and ydExtendedApi users can call insertCombPositionOrder to send native combination and decombination instructions, while ydExtendedApi users can call checkAndInsertCombPositionOrder to send native combination and decombination instructions. Compared to other combination methods, the native instruction is the freest, because it allows investors to choose instructions for combination without priority constraints. Native instruction is also the only one that supports decombination. At the same time, the auto instruction and auto tool are developed based on the native instruction.

```
1 virtual bool insertCombPositionOrder(YDInputOrder *pInputOrder, const YDCombPositionDef
   *pCombPositionDef, const YDAccount *pAccount=NULL)
2 virtual bool checkAndInsertCombPositionOrder(YDInputOrder *pInputOrder, const YDCombPositionDef
   *pCombPositionDef, const YDAccount *pAccount=NULL)
```

The parameters of checkAndInsertCombPositionOrder and insertCombPositionOrder are similar, the difference is that checkAndInsertCombPositionOrder should be subject to a validity check for input parameters at the API end compared to insertCombPositionOrder. If the validity check fails, a "False" will be directly sent back through checkAndInsertCombPositionOrder. The error reasons can be obtained via ErrorNo of YDInputOrder instead of the OMS's check, these validity checks are:

- Checking the validity of trading direction, hedge flag and combination quantity values
- Checking if there are sufficient positions for combination and if there are sufficient traditional combined positions for decombination

API local validity checks can help to save time for sending notifications due to invalid fields, but an overhead to each combination instruction will be added. However, combination instructions are not sensitive to performance, so it is suggested that all ydExtendedApi users who want to use the native instruction send combination instructions through checkAndInsertCombPositionOrder.

The calling method for insertCombPositionOrder and checkAndInsertCombPositionOrder is shown below.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_CombPosition
	OrderRef	Customer order reference No., refer to Normal Order for more details about the description of OrderRef.
	Direction	YD_D_Make: Combine YD_D_Split: Decombine
	OrderType	0
	HedgeFlag	For filling in YD_HF_Speculation
	OrderVolume	Volume required for combination or decombination. When combining, the volume for the two legs' traditional uncombined positions should be higher than that for combination. When decombinig, the volume for traditional combined positions should be higher than that for decombination.
	CombPositionDetailID	Combination details ID. Only required for decombination in SSE and SZSE. For other circumstances, it should be 0.
YDCombPositionDef		Traditional combined position definition pointer for pending combinations, refer to Traditional combined Position Definition for details.
YDAccount		The given "NULL" means that the current API login account is used

After a combination is made or fails, the investors will be notified through the following callback interface.

```
1 | virtual void notifyCombPositionOrder(const YDOrder *pOrder, const YDCombPositionDef *pCombPositionDef, const YDAccount *pAccount) {}
```

7.4.1.1 Extended combination notification

If using YDExtendedListener, investors can receive detailed combination notifications through notifyExchangeCombPositionDetail under the following circumstances:

- When receiving detailed preday combination information, refer to [traditional combined preday position](#) for more details
- When a detailed combination changes due to a successful combination or unfreezing instruction service

```
1 | virtual void notifyExchangeCombPositionDetail(const YDExtendedCombPositionDetail *pCombPositionDetail)
```

For the structure of YDExtendedCombPositionDetail, refer to [traditional combined preday position](#).

7.4.2 Auto instruction

For ydExtendedApi users, in most cases, the function of autoCreateCombPosition can be called to automatically complete combinations. Currently, the auto instruction is only supported by DCE and GFEX. For SSE and SZSE, decombination is required first for position closing, and the native instruction must be used for decombination. Therefore, for the combination and decombination in SSE and SZSE, the native instruction should be used directly.

```
1 | virtual const YDCombPositionDef *autoCreateCombPosition(const int *combTypes)
```

When the autoCreateCombPosition is called each time, the positions that can be combined can be checked one by one according to the combination type sequence specified by combTypes, and under the same combination type, to the priority sequence specified by YDCombPositionDef.Priority (the lower the value, the higher the priority). Once the positions that can be combined are found, a combination instruction will be sent to the OMS. Please note at most one combination is allowed each time when the autoCreateCombPosition is called. In production, a separate thread can be used to continuously call this function at a certain gap. If the positions to be combined are found and sent successfully, the corresponding traditional combined position definition will be sent back through this function. If the positions to be combined are not found, a "NULL" will be sent back through the function.

The calling method for autoCreateCombPosition is shown below.

Parameter	Description
const int* combTypes	The type array to be involved in a combination, ended with -1. The combination should be conducted according to the array sequence and types one by one. When it is NULL, the sequence of all traditional combined position types of DCE is adopted by default, namely, from YD_CPT_DCE_FuturesOffset to YD_CPT_DCE_SellOptionsCovered

After a combination is made or fails, the investors will be notified through the following callback interface.


```
1 virtual void notifyCombPositionOrder(const YDOrder *pOrder,const YDCombPositionDef *pCombPositionDef,const YDAccount *pAccount) {}
```

7.4.3 Auto tool

YD provided an independent auto combination tool namely ydAutoCP for investors early on. This tool can be started up through command lines, and after startup, it, at a certain gap, can be used for searching for positions that can be combined and send combination instructions. Due to many restrictions and inconvenience in using ydAutoCP, it is suggested that investors use APIs to send combination instructions. For the same reason as that for automatic instructions, ydAutoCP only supports the automatic combination of DCE.

The example configuration file ydACP.ini for ydAutoCP is as follows:

```
1 AppID=yd_dev_1.0
2 AuthCode=ecbf8f06469eba63956b79705d95a603
3
4 AccountID=001
5 Password=001
6
7 # time ranges for auto combine position , can be multiple, no such setting indicates selecting at any
  time. Default is no setting
8 TimeRange=21:00:05-23:29:55
9 TimeRange=09:00:05-10:14:55
10 TimeRange=10:30:05-11:29:55
11 TimeRange=13:30:05-15:29:55
12
13 # refresh period, in seconds, default is 10, must be greater than 1
14 RefreshPeriod=10
15
16 # directory to hold log files, default is "ACPLog"
17 LogDir=ACPLog
18
19 # Comb positions types to be processed. If not given, default is all types. Optional values are as
  follow:
20 # YD_CPT_DCE_FuturesOffset=0
21 # YD_CPT_DCE_OptionsOffset=1
22 # YD_CPT_DCE_FuturesCalendarSpread=2
23 # YD_CPT_DCE_FuturesProductSpread=3
24 # YD_CPT_DCE_BuyOptionsVerticalSpread=4
25 # YD_CPT_DCE_SellOptionsVerticalSpread=5
26 # YD_CPT_DCE_OptionsStraddle=7
27 # YD_CPT_DCE_OptionsStrangle=8
28 # YD_CPT_DCE_BuyOptionsCovered=9
29 # YD_CPT_DCE_SellOptionsCovered=10
30 CombPositionType=0,1,2,3,4,5,7,8,9,10
```

Except a few obvious parameters, the core parameters are described as follows:

- The TimeRange can help to specify the system running time. Because ydAutoCP will disable traditional combination position definitions from being involved in the subsequent combinations in case of a combination error and sending traditional combination orders during non-trading hours will result in that all combination position definitions be disabled during this period of time, when configuring the system, the non-trading hours of exchanges must be avoided. At the same time, the operating system running ydAutoCP must be properly timed.
- The meaning of the parameter combTypes for CombPositionType and autoCreateCombPosition is the same. Please refer to [Auto Instruction](#) for the relevant details.

The startup method of ydAutoCP is the following, where config.txt is the configuration file of YD API, and the configuration file ydACP.ini of ydAutoCP is loaded by default, so the file name cannot be modified:

```
1 | ./ydAutoCP config.txt
```

Once a traditional combination position definition is disabled in the system, the ydAutoCP must be restarted if a revalidation of this definition is expected. The disabled traditional combination position definition can be checked in the log. The following shows an example about error messages:

```
1 | error in handling (pg2302&-eg2303,1) for account 12345678
```

7.5 Exercise and performance

The following text will provide a detailed introduction to the exercise and performance business of various exchanges. Although this article attempts to comprehensively explain the specific business logic, there may be omissions, and as the business of the exchanges develops, the following business logic may no longer be in line with the actual exercise and performance business of the exchanges. Therefore, the following content cannot be used as the basis for investors to implement exchange exercise and performance business, all subject to the announcement of the exchange.

To avoid losses during the exercise and performance process, it is **essential** to test and verify the exercise and performance behavior in the exchange's simulation environment before using the exercise and performance API interface of the YD OMS for business services in the production environment. YD is not responsible for any losses resulting from misunderstood or untested exercise and performance behavior.

7.5.1 Exercise and performance of SHFE and INE

According to the exercise and performance rules of SHFE and INE, investors can settle their options positions before exercise through offsetting or holding offsetting positions in two-way options; on the expiration day, the exchange will automatically exercise options for the option buyer according to the logic of exercising for in-the-money options and not exercising for out-of-the-money options. However, the buyer of an in-the-money option can choose to abandon the exercise, and the buyer of an out-of-the-money option can request exercise to override the exchange's automatic exercise logic. On the expiration day, both the option buyer and seller can apply for exercise and performance to offset their two-way futures positions. If an investor's position involves the above-mentioned transactions at the same time, the exchange's processing logic is as follows: first, the offsetting of the two-way option positions is processed, followed by the handling of the futures opening and exercise abandonment related to the option exercise application. Then, the remaining positions are automatically exercised or abandoned, and finally, the offsetting of the two-way futures options after exercise and performance is processed. The above-mentioned transactions are all application-based transactions, which means that only capital and position freezing processing is done during trading hours, and the unified calculation and processing are carried out by the post-trading settlement system.

- Hedging of bi-directional options positions involves applying for offsetting closing or automatic offsetting closing of bi-directional options positions for the same trade code and contract. For regular investors, hedging of bi-directional options positions is not done by default and requires manual application by the investor. For market makers, hedging of bi-directional options positions is done by default, and if a market maker does not wish for automatic hedging of bi-directional options positions, they can apply for exemption from automatic hedging. The hedging of bi-directional options positions results in a deduction from the current options position, incurring trading fees, and an adjustment to the corresponding position volume. The application for hedging and the application for waiving the hedging of European options must be submitted before 15:30 on the expiration date. The application for hedging and the application for waiving the hedging of American options must be submitted during trading hours on any trading day before the expiration date and before 15:30 on the expiration date. The application applies to specified instruments and quantities under the designated trading code, and the application is only valid for the current day.
- Exercise and waiver of exercise refer to the ability of the buyer of European options to submit an exercise or waiver of exercise request before 15:30 on the expiration date. The buyer of American options can submit an exercise request during any trading session on any trading day before the expiration date, and on the expiration date, they can submit an exercise or waiver of exercise request before 15:30. The application applies to specified instruments and quantities under the designated trading code, and the application is only valid for the current day.
- Automatic exercise on expiration date refers to the automatic exercise of long option positions for investors who have not submitted an exercise or waiver of exercise request within the specified time before the expiration date, and whose options have intrinsic value relative to the settlement price of the underlying futures instrument. Other option positions will be automatically abandoned.
- After exercising, the futures position can be hedged, which means that the option buyer(seller) can apply to hedge and close out the futures position obtained through the exercise under the same trading code, or can apply to hedge and close out the futures position obtained through exercise under the same trading code with the original futures position in the futures market. The hedging quantity shall not exceed the futures position obtained through exercise. The hedging result shall be deducted from the current futures position. The application time for European options is before 15:30 on the expiration date; the application time for American options is any trading day before the expiration date and before 15:30 on the expiration date. The application is for a specified instrument and quantity under a designated trading code, and is only valid for the current day. In the exchange interface, the hedging flag for the bidirectional futures position after exercise is specified in the exercise instruction. In order to simplify the structure of the YD API, the YD OMS sets the hedging flag for the bidirectional futures position after exercise, which investors cannot modify. Therefore, as long as the option buyer sends an exercise request to the exchange, it indicates the need for hedging of the bidirectional futures position after exercise. The hedging flag for the bidirectional futures position after performance is sent through proprietary instructions.

7.5.1.1 Hedging of bidirectional options positions on SHFE and INE

The "insertOrder" instruction is used to initiate an application for hedging or abandoning the hedging of bidirectional options positions at the exchange. The request parameters are as follows. For the same investor, instrument, OrderType and HedgeFlag, only the most recent application record will be retained.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionSelfClose
	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	OrderType	YD_ODT_CloseSelfOptionPosition: Requesting the hedging of bidirectional options positions for ordinary investors. YD_ODT_ReserveOptionPosition: Abandoning the hedging of bidirectional options positions for market makers.
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use the hedge flag. The YD system uses the fund account and the hedge flag to uniquely identify the trading code, so please specify the corresponding hedge flag.
	OrderVolume	
YDInstrument		For specifying an option instrument pointer for hedging of pending bi-directional options position.
YDAccount		The given "NULL" means that the current API login account is used

The success or failure of the options hedging application will be returned through "notifyOrder". Regardless of success or failure, the corresponding order will enter the final state and cannot continue to apply for orders. The values of the "YDOrder" in the return are consistent with the "YDInputOrder" in the application, so the table below ignores the same fields as the YDInputOrder and meaningless fields. Investors mainly focus on the "OrderStatus" and "ErrorNo" in the return to understand whether the application is successful.

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed as YD_YOF_OptionSelfClose.
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	Exchange order ID No. If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated.
	LongOrderSysID	Full-accuracy exchange order ID No. The exchange order ID number will not be truncated for the sake of a full-accuracy
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	OrderStatus	If the application is successful, it will be "YD_OS_AllTraded"; if the application is unsuccessful, it will be "YD_OS_Rejected".
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		For specifying an option instrument pointer for hedging of pending bi-directional options position.
YDAccount		Current API login account.

Due to the fact that SHFE and INE retain only the most recent application for the same investor, instrument, order type and hedge flag, investors shall withdraw the hedging application for the bi-directional options position by placing an order for the same instrument, order flag and hedge flag as the specified bi-directional options position hedging application. In order to comply with the characteristics of the exchange's instructions, YD cancels the hedging of the bi-directional option position using an independent order from the hedging application for the bi-directional options position. Therefore, the "OrderRef" in the cancellation instruction should not be filled with the "OrderRef" in the application instruction to avoid being subject to [Monotonic increase check of order reference number](#) and [Order group](#), which may result in the failure of the cancellation instruction.

Using the insertOrder command to cancel the application for hedging the bi-directional options position with the exchange, the request parameters are as shown in the following table.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionSelfClose
	OrderRef	Customer order reference No., see the description of OrderRef in Normal Order for details
	OrderType	Please fill in the corresponding "OrderType" for the hedging application to be cancelled.
	HedgeFlag	Please fill in the corresponding "HedgeFlag" for the hedging application to be cancelled.
	OrderVolume	Fill in 0 to indicate cancellation of the options hedging application.
YDInstrument		Designate the pointer to the option instrument for which the pending bi-directional option position hedging is to be cancelled.
YDAccount		The given "NULL" means that the current API login account is used

The success or failure of canceling the bi-directional option position hedging will be returned through "notifyOrder". Once the corresponding order enters the final state, it cannot be further canceled, regardless of success or failure. The values of the "YDOrder" in the return should be consistent with the "YDInputOrder" in the application, so the table below ignores the same fields as YDInputOrder and meaningless fields. Investors mainly focus on the "OrderStatus" and "ErrorNo" in the return to understand whether the application is successful.

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed as YD_YOF_OptionSelfClose.
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	Exchange order ID No. If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated.
	LongOrderSysID	Full-accuracy exchange order ID No. The exchange order ID number will not be truncated for the sake of a full-accuracy.
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	OrderStatus	If the application is successful, it will be "YD_OS_AllTraded"; if the application is unsuccessful, it will be "YD_OS_Rejected".
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		For specifying an option instrument pointer for hedging of pending bi-directional options position.
YDAccount		Current API login account.

7.5.1.2 Option exercise of SHFE and INE

For the sake of easy programming, YD provides parameterized representations for supporting option exercise or not. The YDExchange.OptionExecutionSupport is used for supporting option exercise at different levels:

- 0: Not supported. It is applicable to CFFEX now.
- 1: Supported, excluding risk control. The so-called "Excluding Risk Control" means excluding money position check and freezing. It is unapplicable to any exchange now.
- 2: Supported, including risk control. The so-called "Risk Control" means money position check and freezing. It is applicable to SHFE, INE, DCE, CZCE, GFEX, SSE and SZSE now.

Use the "insertOrder" command to initiate an exercise request to the exchange. The parameter filling method is as shown in the table below.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionExecute
	OrderRef	Customer order reference No., refer to Normal Order for more details about the description of OrderRef
	Direction	For filling in YD_D_Sell
	OffsetFlag	For filling in the OffsetFlag of the position corresponding to the exercise For SHFE and INE, for today's position, fill in "YD_OF_CloseToday", and for yesterday's position, fill in "YD_OF_CloseYesterday". For DCE, CZCE, and GFEX, just fill in YD_OF_Close
	OrderType	For filling in YD_ODT_Limit
	HedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge.
	OrderVolume	For specifying the volume for exercise
YDInstrument		For specifying the instrument pointer for options to be exercised
YDAccount		The given "NULL" means that the current API login account is used

The option exercise notification will be returned through the "notifyOrder". If the option exercise request is successful, the notification will show as "pending order"; if the option exercise request fails, the notification will show as "failed order". The values of the "YDOrder" in the return are consistent with the "YDInputOrder" in the application, so the table ignores the same fields as the YDInputOrder and meaningless fields. Investors are mainly concerned with the "OrderStatus" and "ErrorNo" in the return to understand whether the application is successful.

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed as YD_YOF_OptionExecute.
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	Exchange order ID No. If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated.
	LongOrderSysID	Full-accuracy exchange order ID No. The exchange order ID number will not be truncated for the sake of a full-accuracy.
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	OrderStatus	If the application is successful, it will be "YD_OS_Queueing"; if the application is unsuccessful, it will be "YD_OS_Rejected".
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		Pointer to the option instrument pending exercise.
YDAccount		Current API login account.

As the exercise request for the option is in pending status after a successful application, it can be canceled using the "cancelOrder" function. The usage and notification for canceling exercise requests and cancelling ordinary orders are the same. For details, please refer to [Order cancellation](#)

7.5.1.3 Abandonment of option exercise of SHFE and INE

For the sake of easy programming, YD provides parameterized representations for supporting option exercise abandonment or not. The YDExchange.OptionAbandonExecutionSupport is used for supporting option exercise at different levels:

- 0: Not supported. It is applicable to CFFEX, SSE, SZSE, DCE and GFEX now. CFFEX, SSE and SZSE do not support option exercise abandonment instruction. DCE and GFEX support abandoning option exercises through YD_YOF_Mark rather than YD_YOF_OptionAbandonExecute, refer to the following content for details.
- 1: Supported, excluding risk control. The so-called "Excluding Risk Control" means excluding money position check and freezing. It is unapplicable to any exchange now.
- 2: Supported, including risk control. The so-called "Risk Control" means money position check and freezing. It is applicable to SHFE, INE and CZCE now.

Use the "insertOrder" command to initiate the abandonment of the option exercise request to the exchange. The parameter filling method is as shown in the following table.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionAbandonExecute
	OrderRef	Customer order reference No., refer to Normal Order for details about the description of OrderRef
	Direction	For filling in YD_D_Sell
	OffsetFlag	For filling in the OffsetFlag of the position corresponding to the exercise For SHFE and INE, for today's position, fill in "YD_OF_CloseToday", and for yesterday's position, fill in "YD_OF_CloseYesterday". For other exchanges, just fill in YD_OF_Close
	OrderType	For filling in YD_ODT_Limit

Parameter	Field	Description
	HedgeFlag	Hedge flag. The definition is different for futures exchanges and stock option exchanges. For futures exchanges: YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge For stock option exchanges: YD_HF_Normal=1: Normal YD_HF_Covered=3: Covered
	OrderVolume	For specifying the volume for exercise
YDInstrument		For specifying the instrument pointer for options to be abandoned.
YDAccount		The given "NULL" means that the current API login account is used

The abandonment of the option exercise will be returned through "notifyOrder". If the abandonment of the option exercise request is successful, the return will be displayed as pending order; if the abandonment of the option exercise request is unsuccessful, the return will be displayed as rejected order. The values of the YDOrder in the return are consistent with those of the YDInputOrder in the application, so the table ignores the same fields as the YDInputOrder and meaningless fields. Investors mainly focus on the "OrderStatus" and "ErrorNo" in the return to understand whether the application is successful.

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed as YD_YOF_OptionAbandonExecute.
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	Exchange order ID No. If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated.
	LongOrderSysID	Full-accuracy exchange order ID No. The exchange order ID number will not be truncated for the sake of a full-accuracy.
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
	OrderStatus	If the application is successful, it will be "YD_OS_Queueing"; if the application is unsuccessful, it will be "YD_OS_Rejected".
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		For specifying the instrument pointer for options to be abandoned.
YDAccount		Current API login account.

After the successful abandonment of the option exercise, the application order is in a pending status. Therefore, if you want to cancel the abandonment application, you can use "cancelOrder" to withdraw it. The usage and return of canceling the abandonment application and canceling a normal order are the same. Please refer to [Order cancellation](#).

7.5.1.4 Hedge performance for bi-directional futures positions after fulfillment for SHFE and INE

Use the "insertOrder" command to initiate a request for hedging bi-directional futures positions after fulfillment on the exchange, with the following parameters. For the same investor, instrument order type and hedge flag, only the most recent application record is retained.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionSelfClose.
	OrderRef	Customer order reference No., refer to Normal Order for details about the description of OrderRef
	OrderType	YD_ODT_SellCloseSelfFuturesPosition: Hedge performance for bi-directional futures positions after fulfillment.

Parameter	Field	Description
	HedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use hedging flags. YD OMS uses the fund account and hedging flag to uniquely determine the trading code, so please specify the corresponding hedging flag.
	OrderVolume	The volume for hedging bi-directional futures positions.
YDInstrument		Designate the pointer to the option instrument for hedging bi-directional futures positions after fulfillment.
YDAccount		The given "NULL" means that the current API login account is used

The notification on hedging bi-directional futures positions after fulfillment, the notification on the cancellation of the application for hedging bi-directional futures positions after fulfillment and the notification on the cancellation of the application for hedging bi-directional futures positions after fulfillment are similar to the [Hedging of bidirectional options positions on SHFE and INE](#), please refer to the corresponding content.

7.5.2 Exercise and performance of DCE and GFEX

According to the exercise and performance rules of DCE and GFEX, investors can settle their options positions before exercise through closing out or hedging with bi-directional options positions. On the expiration day, the exchange will automatically exercise options for the option buyers based on the logic that in-the-money options are exercised by default, while out-of-the money options are not exercised by default. However, in-the-money option buyers can choose to waive their right to exercise, and out-of-the-money option buyers can request exercise to override the exchange's automatic exercise logic. On the expiration day, the option buyer can apply to exercise and hedge the bi-directional futures position. On any trading day, investors can apply to perform and hedge the bi-directional futures position, and once the application is successful, it will be permanently effective. If an investor's position simultaneously utilizes the above-mentioned services, the exchange's processing logic is as follows: first, the hedge of the bi-directional option position is processed, then the exercise of the option and the opening of the futures position are handled, followed by the hedge of the bi-directional futures position after exercise, and finally the hedge of the bi-directional futures position after performance. Both the hedge of the bi-directional option position and the abandonment of exercise are special application services, and no funds or position freezes are carried out during trading hours; exercise application is a general application service, and funds and position freezes are carried out during trading hours.

- Hedging of bi-directional option positions refers to the application for hedging and closing out of the bi-directional option positions under the same trading code and instrument. It follows the principle of maximum hedging volume (taking the smaller of the buying position and selling position) and prioritizes closing speculative positions before closing hedged positions. The closing price for the hedging position is the settlement price of the current day. The closing sequence is based on the principle of "first in, first out". The transaction is included in the trading volume and trading amount, and transaction fees are charged. The position volume is adjusted accordingly. The application can be made during the trading hours of the trading day and from 15:00 to 15:30 on the expiration day. The application is only valid for the specified instrument under the designated trading code on the same day.
- Automatic exercise on expiration date refers to the automatic exercise of a call (put) option position if the exercise price is lower (higher) than the settlement price of the underlying futures instrument after the market closes on the expiration date. All other options will be automatically abandoned. For positions with automatic exercise application by the exchange, investors have the option to waive the automatic exercise application. If the investor does not waive the automatic exercise, the exchange will submit the exercise application for the entire position (without deducting the exercise application volume already submitted by the buyer). If both the exercise application has been submitted and the automatic exercise is not waived, the exchange will first process the exercise application submitted by the buyer, and then process the exchange's automatic exercise application until the buyer's entire position is exercised. If the automatic exercise is waived, the exchange will not submit an exercise application for the instrument on behalf of the buyer. If partial exercise of the position is required, an exercise application for the corresponding volume must be submitted and the automatic exercise application canceled (the order sequence of submission does not affect the process). For positions with automatic exercise waiver by the exchange, investors can apply for manual exercise to override the automatic exercise waiver. The application can be made during the trading hours and from 15:00 to 15:30 on the expiration date. The manual exercise application applies to specified instrument and volume under a designated trading code and is only valid for the current day. The manual waiver of exercise applies to specified instrument under a designated trading code and is also only valid for the current day. Please note that manual exercise and manual waiver of exercise can coexist and are not mutually exclusive. Assuming an investor holds 10 instruments, the final exercise results under different conditions are shown in the table below. For simplicity, the exercise results do not consider situations where exercise is not possible due to insufficient funds, etc.

Exchange automatic logic	Manual exercise of options	Manual abandonment of options	The exercise result
Exercise 10 lots	Apply for 5 lots	Yes	Exercise 5 lots
Exercise 10 lots	Apply for 5 lots	No	Exercise 10 lots
Exercise 10 lots	Do not apply	Yes	Not exercise
Exercise 10 lots	Do not apply	No	Exercise 10 lots
Waive exercise	Apply for 5 lots	Yes	Exercise 5 lots
Waive exercise	Apply for 5 lots	No	Exercise 5 lots
Waive exercise	Do not apply	Yes	Not exercise
Waive exercise	Do not apply	No	Not exercise

- The hedging of the bi-directional futures position after exercise refers to the option buyer's ability to apply for hedging and closing out the bi-directional futures position under the same trading code after exercising the option. This process must adhere to the principle of not exceeding the hedging volume of the futures position obtained from exercise (taking the smaller of the exercise opening position and the reverse position), and the speculative position should be closed before the hedging position. The closing price for the hedging position is the settlement price of the current day, and the closing sequence follows the "first in, first out" principle. Transaction fees are charged, and the position volume is adjusted accordingly. The application can be made during the trading hours of the trading day and from 15:00 to 15:30 on the expiration date. The application applies to the specified instrument under the designated trading code and is only valid for the current day. In the exchange interface, the hedging flag for the bi-directional futures position after exercise is specified in the exercise instruction. In order to simplify the structure of the YD API, the YD OMS has fixed the hedging flag for the bi-directional futures position after exercise, and investors cannot modify it.
- The hedging of the bi-directional futures position after performance refers to the option seller's ability to apply for hedging and closing the bi-directional futures position after the performance under the same trading code. The hedging logic of the bi-directional futures position after the performance is the same as the hedging logic of the bi-directional futures position after the exercise. The application time is during the trading hours of the trading day and from 15:00 to 15:30 on the expiration day. The application is for the specified trading code and, once successful, it remains permanently valid.

7.5.2.1 Hedging of bidirectional options positions on DCE and GFEX

Use the "insertOrder" command to initiate a request for hedging the bi-directional options position on the exchange, with the following parameters as shown in the table. Please note that for the hedging instructions of bi-directional options positions on DCE and GFEX, it is not possible to select a specific seat for order placement. The YD OMS will uniformly designate the order to be sent from the lead seat. This is because such orders do not have a system order number or a local order number, and therefore, it is not possible to implement any mechanism to identify duplicate receipts. If the all management seat mode is used to receive the full flow, all seats will receive the notification, which will cause the YD OMS to generate unreliable order notifications. Therefore, the current implementation is to uniformly send and receive using the lead seat. However, this also brings a problem, if the lead seat is not a public seat, then for investors who cannot use the lead seat, they will not be able to issue this instruction. Therefore, for Dalian Commodity Exchange and Zhengzhou Commodity Exchange, the lead seat cannot be configured as a dedicated seat.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark
	OrderRef	Always fill in 0 The OrderRef in the notification is always -1 and cannot be matched with the OrderRef in the request.
	Direction	YD_D_Buy: Initiate an application for hedging of the bi-directional option position.
	OrderType	YD_ODT_PositionOffsetMark: Hedging a bi-directional option position.
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use a hedge flag, the YD OMS uses the fund account and the hedge flag to uniquely determine the trading code, so please specify the corresponding hedge flag.
YDInstrument		The specified pointer for hedging the pending bi-directional option position.
YDAccount		The given "NULL" means that the current API login account is used

Due to the special nature of the hedging instructions for the bi-directional option positions of DCE and GFEX, YD cannot associate the requests and notifications sent to the exchange. After receiving the notifications from the exchange, YD will generate special notifications on its own, as shown in the table below. Therefore, after sending the hedging instructions for the bi-directional option positions of DCE and GFEX, investors should not keep the order in the client terminal or wait for a matching notification. Instead, after receiving the order notification with YDOrderFlag as YD_YOF_Mark, they should differentiate the specific business and whether the instruction was successful or not based on the instrument, exchange, OrderType, and OrderStatus. For the same reason as mentioned above, even if the AccountFlag in YDAccount is set as YD_AF_NotifyOrderAccept to receive notifications from the counter, the hedging instructions for DCE and GFEX will not send notifications from the counter.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark.
	Direction	YD_D_Buy: Initiate a request for hedging a bi-directional option position.
	OrderType	YD_ODT_PositionOffsetMark: Hedging a bi-directional option position.
	HedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge.
	OrderRef	-1
	OrderLocalID	-1
	OrderSysID	0
	InsertTime	-1

Parameter	Field	Description
	OrderStatus	success: YD_OS_AllTraded failure: YD_OS_Rejected
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		The hedge instrument pointer for the bi-directional option position.
YDAccount		Current API login account.

Use the "insertOrder" command to cancel the bi-directional option hedge position request at the exchange, with the request parameters as shown in the table below.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark
	OrderRef	Always fill in 0 The OrderRef in the notification is always -1 and cannot be matched with the OrderRef in the request.
	Direction	YD_D_Sell: cancel the bi-directional option hedge position request.
	OrderType	YD_ODT_PositionOffsetMark: Hedging a bi-directional option position.
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use a hedge flag, the YD OMS uses the fund account and the hedge flag to uniquely determine the trading code, so please specify the corresponding hedge flag.
YDInstrument		The specified pointer for hedging the pending bi-directional option position.
YDAccount		The given "NULL" means that the current API login account is used

The cancellation of the bi-directional option hedge position request will result in a success or failure notification through the "notifyOrder" command. Once the request is in its final state, it cannot be further cancelled, regardless of whether it was successful or not. The values of the corresponding fields in the YDOrder from the notification and the YDInputOrder from the application remain consistent. Therefore, the table below omits the same fields as YDInputOrder and meaningless fields. Investors should focus on the OrderStatus and ErrorNo in the notification to understand whether the application was successful.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark.
	Direction	YD_D_Sell: Cancel the application for hedging of the bi-directional option position.
	OrderType	YD_ODT_PositionOffsetMark: Hedging a bi-directional option position.
	HedgeFlag	Hedge flag. YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge.
	OrderRef	-1
	OrderLocalID	-1
	OrderSysID	0
	InsertTime	-1
	OrderStatus	success: YD_OS_AllTraded failure: YD_OS_Rejected
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		The hedge instrument pointer for the bi-directional option position.
YDAccount		Current API login account.

7.5.2.2 Option exercise of DCE and GFEX

The exercise reporting method for DCE and GFEX is the same as that for SHFE and INE. For details, please refer to the [option exercise of shfe and ine](#).

7.5.2.3 Abandonment of option exercise of DCE and GFEX

Please refer to the following instructions for using insertOrder to initiate an exercise abandonment request to the exchange.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark
	OrderRef	Customer order reference number Due to the lack of order-related fields in the upward and downward interfaces for exercise abandonment in DCE and GFEX, YD is unable to associate the exchange feedback of exercise abandonment with the corresponding investor order. As a result, the OrderRef sent by the OMS is always -1, which cannot correspond to the OrderRef in the request.
	Direction	YD_D_Buy: Abandon Exercise
	OrderType	YD_ODT_OptionAbandonExecuteMark: Abandon Exercise
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use a hedge flag, the YD OMS uses the fund account and the hedge flag to uniquely determine the trading code, so please specify the corresponding hedge flag.
YDInstrument		Specify the pointer to the options instrument for which automatic exercise is to be abandoned.
YDAccount		The given "NULL" means that the current API login account is used

Due to the special nature of the cancellation of the abandonment of option exercise instructions of DCE and GFEX, YD cannot associate the requests and notifications sent to the exchange. After receiving the notifications from the exchange, YD will generate special notifications on its own, as shown in the table below. Therefore, after sending the cancellation of the abandonment of option exercise instructions of DCE and GFEX, investors should not keep the order in the client terminal or wait for a matching notification. Instead, after receiving the order notification with YDOrderFlag as YD_YOF_Mark, they should differentiate the specific business and whether the instruction was successful or not based on the instrument, exchange, OrderType, and OrderStatus. For the same reason as mentioned above, even if the AccountFlag in YDAccount is set as YD_AF_NotifyOrderAccept to receive notifications from the counter, the cancellation of the abandonment of option exercise instructions of DCE and GFEX will not send notifications from the counter.

The cancellation of the abandonment of option exercise instructions will result in a success or failure notification through the "notifyOrder" command. Once the request is in its final state, it cannot be further cancelled, regardless of whether it was successful or not. The values of the corresponding fields in the YDOrder from the notification and the YDInputOrder from the application remain consistent. Therefore, the table below omits the same fields as YDInputOrder and meaningless fields. Investors should focus on the OrderStatus and ErrorNo in the notification to understand whether the application was successful.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark
	Direction	YD_D_Sell: cancel the abandonment of option exercise.
	OrderType	YD_ODT_OptionAbandonExecuteMark: abandon option exercise.
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use a hedge flag, the YD OMS uses the fund account and the hedge flag to uniquely determine the trading code, so please specify the corresponding hedge flag.
	OrderStatus	success: YD_OS_AllTraded failure: YD_OS_Rejected
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument		Specify the pointer to the options instrument for which automatic exercise is to be abandoned.
YDAccount		Current API login account.

7.5.2.4 Hedge performance for bi-directional futures positions after fulfillment for DCE and GFEX

For the sake of simplifying the API, when initiating an instruction for hedge performance, please specify any instrument belonging to DCE or GFEX, so that YD can identify the exchange through the instrument.

Please refer to the following method to use the insertOrder command to initiate a bi-directional futures position hedging application after fulfillment to the exchange.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_Mark
	OrderRef	Always fill in 0 The OrderRef in the notification is always -1 and cannot be matched with the OrderRef in the request.

Parameter	Field	Description
	Direction	YD_D_Buy: Hedging request application
	OrderType	YD_ODT_CloseFuturesPositionMark: Hedging of bidirectional futures positions after fulfillment.
	HedgeFlag	Hedge flag YD_HF_Speculation=1: Speculation YD_HF_Hedge=3: Hedge The exchange does not use a hedge flag, the YD OMS uses the fund account and the hedge flag to uniquely determine the trading code, so please specify the corresponding hedge flag.
YDInstrument		A designated pointer to any instrument on the corresponding exchange.
YDAccount		The given "NULL" means that the current API login account is used.

In the notification on hedging bi-directional futures positions after fulfillment, the exchange does not include specific instruments in the notification. Therefore, YD will return the first instrument of the corresponding exchange along with the notification, and investors can obtain the exchange of this instrument. The notification on hedging bi-directional futures positions after fulfillment, the notification on the cancellation of the application for hedging bi-directional futures positions after fulfillment and the notification on the cancellation of the application for hedging bi-directional futures positions after fulfillment are similar to the [hedging of bidirectional options positions on dce and gfex](#), please refer to the corresponding content.

7.5.3 Exercise and performance of CZCE

According to the exercise and performance rules of CZCE, investors can settle their options positions before exercise through closing out bi-directional options positions. On the expiration day, the exchange will automatically exercise options for the option buyers based on the logic that in-the-money options are exercised by default, while out-of-the money options are not exercised by default. However, in-the-money option buyers can choose to waive their right to exercise, and out-of-the-money option buyers can request exercise to override the exchange's automatic exercise logic. The exchange's processing logic is as follows: first, process the futures opening of the options exercise application and the abandonment of the options exercise application. The above-mentioned transactions are all application-based transactions, which means that only funds and positions are frozen during trading hours, and they are uniformly calculated and processed by the post-trading settlement system.

- Exercise and abandonment of exercise refer to the option buyer's ability to submit exercise or abandonment requests at any time during the trading period before the expiration date and on the expiration date. The requests are for the specified instrument and volume under a designated trading code, and are only valid for the current day.
- Automatic exercise on expiration date means that for option positions that have not had exercise or abandonment requests submitted within the specified timeframe at expiration date settlement, the options will be automatically exercised at their intrinsic value (futures settlement price) for in-the-money options, and automatically abandoned for out-of-the-money options. The specific handling is as follows: for call options (long positions) with a strike price lower than the underlying asset's settlement price on the expiration date, they will be automatically exercised; for put options (short positions) with a strike price higher than the underlying asset's settlement price on the expiration date, they will be automatically exercised; all other option positions will be automatically abandoned. Please note that, due to differences between the closing price and settlement price of the underlying futures, variations in transaction costs, and investor judgments on future market trends of the slightly in-the-money options and the slightly out-of-the-money options, it may be necessary to either abandon slightly in-the-money options or exercise slightly out-of-the-money options.

7.5.3.1 Option exercise of CZCE

The exercise reporting method for CZCE is the same as that of SHFE and INE. For details, please refer to [option-exercise-of-shfe-and-ine](#).

7.5.3.2 Abandonment of option exercise of CZCE

The exercise abandonment reporting method for CZCE is the same as that of SHFE and INE. For details, please refer to [abandonment-of-option-exercise-of-shfe-and-ine](#).

7.5.4 Exercise and performance of CFFEX

According to the exercise and performance rules of the CFFEX, the exchange does not support exercise instructions. On the last trading day, CFFEX will automatically exercise all in-the-money options. If you do not wish to exercise, please close out your position before the last trading day.

7.5.5 Exercise and performance of SSE and SZSE

According to the exercise and performance rules of the SSE and SZSE, investors can close out their options positions before exercise. On the exercise date, they can apply for exercise and combined exercise, i.e., submit combined exercise instructions. If exercise and combined exercise are applied for simultaneously, the processing logic of ChinaClear is as follows: the combined exercise instructions are processed first, followed by the non-combined exercise instructions. The above-mentioned transaction are all application-based transactions, meaning that only fund and position freezing processing is done during trading hours, and the unified calculation and processing are carried out by the post-trading settlement system.

- Exercise application refers to the option buyer's ability to submit an exercise application during the period of 15:00-15:30 on the exercise date. The application is for the specified instrument and volume under a designated trading code, and it is only valid for the same day. The exercise volume for investors should be the number of uncovered long positions(including the volume that has been applied for but not yet executed). If exercise is not applied for, it is considered as waiving the right to exercise.

- Combined exercise (exercise instruction consolidation application) refers to the ability to simultaneously submit exercise applications for uncovered long positions of call and put options on the same underlying security that expire on the same day, achieving synchronized exercise of call and put options at expiration, improving the efficiency of investors' capital utilization, and avoiding overnight risks in the spot market during settlement. Investors can submit consolidated exercise applications multiple times, and the cumulative exercise volume should not exceed the net position of the rights held (i.e. the position after the corresponding derivative instrument account's expiration combination strategy is released and the rights and obligations positions are hedged). If the volume in a particular application exceeds the current net position, only the net position of the rights held in that application is valid. The time for submitting consolidated exercise applications is from 9:15 to 9:25, 9:30 to 11:30, and 13:00 to 15:30 on the exercise day. The option instrument for consolidated exercise must meet the following conditions:
 - The underlying securities of option instruments should be the same, for example, an SSE - SZSE 300ETF option instrument and an SSE 50ETF option instrument cannot be submitted together
 - The exercise price of a put option on a trading day must be higher than the call option and must be subject to an option instrument expiring on the very day
 - The instrument units must be the same. The underlying dividends, etc., may lead to adjustments in the instrument for ETF options, which may result in different instrument units for standard instruments and non-standard instruments (instruments with non-M codes). However, such non-standard instruments and standard instruments cannot be submitted together.
 - The volume of call/put options to be exercised shall not exceed the net volume of the corresponding instrument-based long positions held by investors. The volume of net positions of a trading day should be determined after the termination of the combination strategy at the very day. If exceeding the currently available exercise limit, the submitted volume will be invalid.

7.5.5.1 Option exercise of SSE and SZSE

The exercise reporting method for SSE and SZSE is the same as that for SHFE and INE. For more details, please refer to [option exercise of shfe and ine](#)

7.5.5.2 The combined exercise of SSE and SZSE

For SSE and SZSE, please refer to the following methods to use the insertOptionExecTogetherOrder and checkAndInsertOptionExecTogetherOrder commands to initiate a request for combined exercise application to the exchange, and user the cancelOrder command to cancel the combined exercise request.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	For filling in YD_YOF_OptionExecuteTogether
	OrderRef	Customer order reference No., refer to Normal Order for more details about the description of OrderRef
	OrderVolume	For specifying the volume for combined exercise.
YDInstrument	plnstrument	For specifying the pointer of option instrument for combined exercise
	plnstrument2	For specifying the pointer of option instrument for combined exercise
YDAccount	pAccount	The given "NULL" means that the current API login account is used

The notification for combined exercise is returned through notifyOptionExecTogetherOrder. If the combined exercise application is successful, the notification will show as pending order. If the exercise application fails, the notification will show as failed order. The values of the "YDOrder" and the "YDInputOrder" for the same fields in the response are consistent. Therefore, the table ignores the same fields as the YDInputOrder and meaningless fields. Investors mainly focus on the OrderStatus and ErrorNo in the notification to understand if the application is successful.

Parameter	Field	Description
YDOrder	YDOrderFlag	Fixed as YD_YOF_OptionExecuteTogether.
	OrderLocalID	Local OMS ID. For internal use by the YD OMS, its positive, negative, unique, incremental and other characteristics may change due to the version of the exchange, members, and the YD OMS. Please do not use this field as the identification of the order.
	OrderSysID	Exchange order ID No. If the exchange order ID number exceeds the maximum length of OrderSysID, part of it will be truncated.
	LongOrderSysID	Full-accuracy exchange order ID No.The exchange order ID number will not be truncated for the sake of a full-accuracy.
	InsertTime	Second counts from the beginning (17:00) to the order submission time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between YD's integral time and reference time, see string2TimeID and timeID2String of ydUtil.h

Parameter	Field	Description
	InsertTimeStamp	Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ For the conversion between YD's timestamp time and reference time, see <code>string2TimeStamp</code> and <code>timestamp2String</code> of <code>ydUtil.h</code> .
	OrderStatus	If the application is successful, it will be "YD_OS_Queueing"; if the application is unsuccessful, it will be "YD_OS_Rejected".
	ErrorNo	It will be 0 when the trade is made successfully, and the exchange error number when an error occurs.
YDInstrument	plInstrument	Pointer to the option instrument pending combined exercise.
YDInstrument	plInstrument2	Pointer to the option instrument pending combined exercise.
YDAccount	pAccount	Current API login account.

After the successful application for combined exercise of options, the application order is in a pending state. Therefore, if you want to cancel the exercise application, you can use the "cancelOrder" command to cancel it. The usage and notification for canceling the application for combined exercise of options is the same as canceling a normal order. For more details, please refer to [order cancellation](#).

7.6 Custody Transfer

YD supports the SZSE's custody transfer business, transferring all investors' holdings to other PBUs. It is usually used when investors moving out from the YD OMS and can only be used by administrator.

The custody transfer is initiated by `insertOrder`. The parameter description is shown in the following table.

Parameter	Field	Description
YDInputOrder	YDOrderFlag	Use YD_YOF_Designation
	HedgeFlag	Use YD_HF_Speculation
	TransfereePBUID	Target PBU of custody Transfer
YDInstrument	plInstrument	Any SZSE Stock pointer
YDAccount	pAccount	Pointer to the investor of custody transfer

7.6.1 Unsupported services

YD aims to support the services needed by investors as much as possible while maintaining the performance. Considering the performance and the actual needs of investors using YD products, YD OMSs have not yet supported the following trading business.

Exchange	Unsupported service
SHFE and INE	Settlement price trade TAS instruction

If you have actual needs for the above services, just contact YD through a broker. If the needs are indeed universal, YD will try its best to help you.

7.7 Trading restrictions

7.7.1 Trading right

Considering the appropriate requirements of risk management and investors, brokers should set trading rights at the investor, exchange, product and instrument levels; Investors also have a risk control need to proactively set some instrument trading rights. Therefore, YD provides four trading right setting sources to meet different risk control setting requirements:

- Permanent rights set by administrators: Permanently valid after setting. Only administrators rather than investors are allowed for this source setting.
- Permanent rights set by investors: Permanently valid after setting. Both administrators and investors are allowed for the setting.
- Temporary rights set by administrators: Valid for the current trading day after setting. Only administrators rather than investors are allowed for this source setting. Currently, they are set after the after-trade risk control rules are triggered to ensure that the after-trade risk control rules are only valid on the current trading day.
- Temporary rights set by investors: Valid for the current trading day after setting. Both administrators and investors are allowed for the setting.

Order cancellation is always allowed regardless of the trading rights of instruments.

Although YD allows the separate setting of trading right sources, an investor's rights on an instrument is the result after summarizing the above four sources. The method for investors to set trading rights is as follows:

```

1 virtual bool setTradingRight(const YDAccount *pAccount, const YDInstrument *pInstrument, const YDProduct
  *pProduct, const YDExchange *pExchange, int tradingRight, int requestID=0, int
  tradingRightSource=YD_TRS_AdminPermanent)

```

The parameters involved in the above method are described as follows:

Parameter	Description
pAccount	Account pointer. The account pointer of the current investor can be obtained through getMyAccount.
pInstrument	Instrument pointer, indicating the right to set an instrument. Please set pProduct and pExchange to NULL at this time
pProduct	Product pointer, indicating the right to set a product. Please set pInstrument and pExchange to NULL at this time
pExchange	Exchange pointer, indicating the right to set an exchange. Please set pProduct and pInstrument to NULL at this time
tradingRight	Rights that should be set, including: YD_TR_Allow=0: Allowing all trades, which is the most relaxed setting. YD_TR_CloseOnly=1: Only position closing is allowed. YD_TR_Forbidden=2: No trades are allowed, which is the most stringent setting.
requestID	For distinguishing between different setting requests, which is usually set to 0. The requestID of a request can be received through notifyResponse.
tradingRightSource	Trading right setting sources YD_TRS_AdminPermanent=0: Permanent rights set by an administrator YD_TRS_UserPermanent=1: Permanent rights set by an investors YD_TRS_AdminTemp=2: Temporary rights set by an administrator YD_TRS_UserTemp=3: Temporary rights set by an investor

The ultimate rights of an investor in a specific instrument depends on the exchange to which the instrument belongs, the product to which the instrument belongs, and the most stringent settings of the instrument. The pseudocodes are expressed as follows:

```

1 int getInstrumentTradingRight(YDApi *api, YDInstrument *instrument)
2 {
3     accountInfo=api->getMyAccount();
4     accountExchangeInfo=api->getAccountExchangeInfo(instrument->m_pExchange);
5     accountProductInfo=api->getAccountProductInfo(instrument->m_pProduct);
6     accountInstrumentInfo=api->getAccountInstrumentInfo(instrument);
7
8     accountInfo->TradingRight = max(
9         accountInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
10        accountInfo->TradingRightFromSource[YD_TRS_UserPermanent],
11        accountInfo->TradingRightFromSource[YD_TRS_AdminTemp],
12        accountInfo->TradingRightFromSource[YD_TRS_UserTemp]
13    );
14
15    accountExchangeInfo->TradingRight = max(
16        accountExchangeInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
17        accountExchangeInfo->TradingRightFromSource[YD_TRS_UserPermanent],
18        accountExchangeInfo->TradingRightFromSource[YD_TRS_AdminTemp],
19        accountExchangeInfo->TradingRightFromSource[YD_TRS_UserTemp]
20    );
21
22    accountProductInfo->TradingRight = max(
23        accountProductInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
24        accountProductInfo->TradingRightFromSource[YD_TRS_UserPermanent],
25        accountProductInfo->TradingRightFromSource[YD_TRS_AdminTemp],
26        accountProductInfo->TradingRightFromSource[YD_TRS_UserTemp]
27    );
28
29    accountInstrumentInfo->TradingRight = max(
30        accountInstrumentInfo->TradingRightFromSource[YD_TRS_AdminPermanent],
31        accountInstrumentInfo->TradingRightFromSource[YD_TRS_UserPermanent],
32        accountInstrumentInfo->TradingRightFromSource[YD_TRS_AdminTemp],
33        accountInstrumentInfo->TradingRightFromSource[YD_TRS_UserTemp]
34    );
35
36    return max(
37        accountInfo->TradingRight,
38        accountExchangeInfo->TradingRight,
39        accountProductInfo->TradingRight,
40        accountInstrumentInfo->TradingRight
41    );
42 }

```

If a trading right changes during trading, it is usually caused by the administrator's active setting or an after-trade risk control trigger. The right change notification can be obtained through the following callback functions. When receiving any of the following three callbacks, the pseudocode function `getInstrumentTradingRight` above should be used to obtain the trading rights for the involved instrument:

```
1 virtual void notifyAccount(const YDAccount *pAccount)
2 virtual void notifyAccountExchangeInfo(const YDAccountExchangeInfo *pAccountExchangeInfo)
3 virtual void notifyAccountProductInfo(const YDAccountProductInfo *pAccountProductInfo)
4 virtual void notifyAccountInstrumentInfo(const YDAccountInstrumentInfo *pAccountInstrumentInfo)
```

If closing positions based on an instrument that does not have trading rights, a failed order notification `notifyOrder` or an error quote notification `notifyQuote` will be received, their `ErrorNo` is `YD_ERROR_NoTradingRight=10`.

The trading rights of a combined instrument does not mean using the rights set in the combined instrument, but separately determining whether the rights set in the two-leg instrument of the combined instrument meet the trading requirements. Assuming that the left and right legs of a combined instrument are Instrument A and B, depending on the rights set by investors in the two instruments, the trades under different combined instruments are as follows:

- If Instrument A and B allow trades, all trades under the combined instrument will be allowed.
- If Instrument A allows trades, and Instrument B allows only position closing, the trades can only be allowed when the two legs of the combined instrument are closed at the same time, or the left leg is opened and the right leg is closed.
- If Instrument A and B only allow position closing, then the trades can only be allowed when the two legs of the combined instrument are opened at the same time.
- If Instrument A allows trades, and Instrument B does not, all trades under this combined instrument will be not allowed.

7.7.2 Trading Constraint

In spot business, securities companies will restrict investors' trading and non-trading behaviors, such as prohibiting buying, prohibiting selling, prohibiting transfer of custody, etc. In terms of trading, trading constraints can only prohibit selling and allow buying, but [Trading Right](#) cannot be expressed; in terms of non-trading, the [Trading Right](#) mechanism is completely inapplicable. Therefore, YD has added a trading constraint function to meet the business needs of securities companies to set trading constraints and modify them during trading.

After calling `adjustCashTradingConstraint`, `notifyResponse` will be returned to indicate whether it is successful. If successful, `notifyAdjustCashTradingConstraint` callback will be received. If failed, it will not be received. This setting method is only valid for spot counters, and can only be called and set by administrators with trading permissions. After setting, it is valid for the current trading day.

```
1 // Trading Constraint
2 virtual bool adjustCashTradingConstraint(const YDAdjustCashTradingConstraint
   *pAdjustCashTradingConstraint, int requestID=0)
3
4 // Adjust Cash Trading Constraint
5 virtual void notifyAdjustCashTradingConstraint(const YDAdjustCashTradingConstraint
   *pAdjustCashTradingConstraint)
```

The structure of `YDAdjustCashTradingConstraint` is as follows:

Field	Description
AccountRef	Set the fund account id of the transaction constraint, which must be set
ExchangeRef	Exchange id. If not set, it means there is no restriction on the exchange.
ProductRef	Product id. If not set, it means there is no restriction on the product
InstrumentRef	Instrument id. If not set, it means there is no restriction on the product
IsSetting	0 means set trading constraint, 0 means cancel trading constraint
CashTradingConstraint	Spot trading constraint bitmap: 0: prohibit buying 1: prohibit selling

`ExchangeRef`, `ProductRef`, and `InstrumentRef` define the scope of contracts that are subject to trading constraints. For example:

- If only `ExchangeRef` is set, all contracts belonging to the exchange are subject to trading constraints
- If only `ProductRef` is set, all contracts belonging to the product are subject to trading constraints
- If only `InstrumentRef` is set, only this contract is subject to trading constraints
- If none of `ExchangeRef`, `ProductRef`, and `InstrumentRef` are set, all contracts are subject to trading constraints

Generally speaking, after setting `InstrumentRef`, it is not necessary to set `ExchangeRef` and `ProductRef`. After setting `ProductRef`, it is not necessary to set `ExchangeRef`. If a fine-grained scope is set and then a coarse-grained scope is set, it is necessary to ensure that they are logically consistent, otherwise the setting will fail. For example, if `ExchangeRef` is set to the Shanghai Stock Exchange, but `InstrumentRef` is set to the Shenzhen Stock Exchange, the setting will fail.

After defining the setting scope, the setting method for each contract depends on the value of `IsSetting`. Assume that an investor's current trading constraint on a contract is to prohibit selling but allow buying (the investor's current trading constraint setting can be obtained from `YDExtendedAccountInstrumentInfo.CashTradingConstraint`), that is, `0b10`. The following demonstrates the effects of different setting methods:

- IsSetting=1, CashTradingConstraint=0b1 or 0b11, if the setting is successful, the investor's trading constraint is 0b11
- IsSetting=0, CashTradingConstraint=0b10 or 0b11, if the setting is successful, the investor's trading constraint is 0b0

7.7.3 Order count and cancellation count limitation

By default, YD does not control the number of orders and cancellations by investors. However, considering the upper limit of the YD OMS's in-memory database, one of the in-memory database's table reaches the upper limit, the entire YD OMS will be unavailable. In order to avoid a sudden increase of the trade volume caused by abnormal trades of some investors from affecting other customers at the OMS, brokers can set a limit on the order count for each investor.

The order count is controlled by the fund account. If an investor establishes more than one connection, the trade volume of all connections can be calculated in summary. Normal orders sent to the exchange and receiving a notification from the exchange will increase the investor's order volume. Quotes, cancellations, derived orders, and failed orders returned by the YD OMS will not be counted in the order volume.

For the current version, The upper limit for database orders and trades is 16.77 million, and the upper limit for quotes is 8.38 million. If there are changes in the business and normal operations require more space, YD can expand at any time.

The order count limit can be found through `YDAccount.MaxOrderCount`. The cancellation limit can be found in `YDAccount.MaxCancelCount`. If the value is -1, it means no limit, which is made by default.

For investors using `ydExtendedApi`, the total order count for the current account can be queried through `YDExtendedAccount.UsedOrderCount`. If an investor has established more than one connection through the fund account, the `YDExtendedAccount.UserOrderCount` will contain the number of reports for all connections. Investors' total cancellation count is only recorded on the counter side and cannot be viewed through the API. Additionally, the cancellation table is cleared after the counter restarts, so the total cancellation count for investors will also be reset to zero due to the restart.

After the investor makes the order count reach the upper limit of the account, for the next order, a `YD_ERROR_TooManyOrders=36` error will be sent back through `notifyOrder`. After an investor reaches the cancellation limit for their account, the next order will receive a `YD_ERROR_TooManyCancels=89` error in the `notifyFailedCancelOrder` notification. In order to prevent the OMS space from being occupied by excessive and abnormal orders, if the investor keeps conducting the order service, the orders will be directly discarded by the OMS without sending back any error notification. The `YD_ERROR_TooManyOrders=36` and `YD_ERROR_TooManyCancels=89` error must be checked, and the order logic must be stopped or adjusted after receiving the error notification. If you have any questions, just contact the broker.

If the OMS memory database resources are exhausted, investors will receive a `YD_ERROR_MemoryExceed=7` error for any order through `notifyOrder`.

7.7.4 Login count limit

By default, the count of login connections for a fund account is not limited by YD OMSs, and investors can log in to YD OMSs unlimitedly with one account. However, excessive connections may cause the TCP down-bound thread to always be busy issuing the down-bound flow, which is unfavorable to the efficient service of YD OMSs. In order to avoid affecting the performance of the OMSs due to excessive connections, brokers can set a login count limit for each investor.

The total login count can be shown through `YDAccount.MaxLoginCount`. If this value is -1, it means no limit, which is made by default. The current login count can be shown through `YDAccount.LoginCount`.

After an investor makes the order count reach the upper limit for his account, a `YD_ERROR_TooManyRequests=20` error will be sent back through `notifyLogin` when logging in to the account next time.

For more information about multiple connections, refer to [Multiple Connections](#) for more details.

7.7.5 Self-trade check

In order to avoid being restricted from trading by exchanges due to investors' self-trades, YD OMSs are provide with a self-trade check function. YD's self-trade check function has been optimized, so it is unnecessary to disable the trade check function due to performance concerns.

This function is enabled by default. If an investor has already performed a check in the strategy program, he/she can ask a broker to help disable the self-trade check function under his account. The investor can check the `YDAccount.AccountFlag` for a set `YD_AF_DisableSelfTradeCheck` flag. If yes, it means that the self-trade check function under this account has been disabled.

The exchanges specify that a self-trade can be only made when the price of a price-limited order overlaps with that of a pending price-limited order. Therefore, market orders, FAK orders and FOK orders are not involved in YD's self-trade check logic. If the trading prices of a price-limited order and the existing pending order do not overlap, they will not be considered as self-traded. For example, under an instrument, if a sell order price is CNY 10, and the quote of a buy order is CNY 11 at this time, it will be considered as a self-trade by the system, while if the quote of the buy order is CNY 9, it will not be considered as a self-trade.

SHFE, INE, DCE, CZCE, and CZEX are exempt from self-matching during the call auction. As a result, orders placed by the aforementioned exchanges during the call auction will not be identified as self-trades by the YD OMSs.

If the error of "1138 order triggered self-trade" is encountered during the trading in CFFEX, it means that the broker has enabled the "Self-trade prevention function" applied for to CFFEX. This function can be used to check whether there is a price overlap between a FAK/FOK order and a price-limited order under the same trading code. If there is an overlap, it will be intercepted by the exchange. The self-trade prevention function is provided by CFFEX to brokers. After being enabled, it can be used for all trading codes of brokers and cannot be set separately for investors. Although the self-trade prevention behavior of FAK/FOK orders can be intercepted through the self-trade prevention function, it does not mean that the self-trades caused by FAK/FOK orders can be determined as the self-trade behavior by the regulatory department of CFFEX. For the Measures for Administration of Abnormal Trades in Financial Futures Exchanges of China, it is clearly stated in the relevant regulatory standard and processing program for stock index options that when counting customer's self-trades, frequent order submissions / cancellations and large-volume order submissions / cancellations, the self-trade behavior caused by all real-time trades or price-limited cancellation instructions, real-time remaining price-limited cancellation instructions and market orders are excluded in the self-trade count.

Although some exchanges grant investors a certain amount of self-trade exemption, considering that the self-trade check is conducted based on investors' trading codes, if the trading code of an investor causes a self-trade due to trading via different brokers, it will also be included in a self-trade by exchanges, and the control of such self-trade is relatively difficult. Considering protecting investors' interests, in order to leave the exemption for uncontrollable self-trades under the same trading code during trading via different brokers, YD OMSs do not allow any potential self-trade orders, and all behaviors detected and considered by the system as potential self-trades will be directly rejected by OMSs.

If any order is considered as involved in a self-trade, a YD_ERROR_PossibleSelfTrade=25 error will be sent back through notifyOrder.

7.7.5.1 Self-trade check of CFFEX quote

Because CFFEX supports multi-level quotes, self-trade check can be very complicated. To ensure no self-trade occurs: When modifying the specific quote, the self-trade check will skip on the quote being modified while other quotes and specific orders will still be checked if they might be self-traded with the new quote; When modifying the last quote, the self-trade check will check all quotes and specific orders if they might be self-traded with the new quote since there is no way for the OMS to know what is the last quote that the exchange received.

There are several reasons that YD OMS can not know the last quote that the exchange received, including but not limited to:

- Investors send quotes in two OMS at the same time.
- Investors send quotes in one OMS continuously, but the sequence that the OMS receives is not the same as the sequence that is sent to the exchange. For example, send three orders one after another: Normal quote A, modification last quote B, and normal quote C. Suppose quote A and quote B will not pass the self-trade check. If using the self-trade check rule of the specific quote, quote A will be skipped because it is being modified; thus, quote B can be sent, quote C is also sent because it will pass the self-trade check with both quote A and quote B. However, the actual sequence that is sent to the exchange is A-C-B caused by reasons such as system internal scheduling or order blocking. Since the last quote when the exchange received quote B is quote C, quote C will be replaced. Thus, quote A and quote B will be self-traded.

Although the above reasons are valid theoretically, they are unlikely to occur in real trading scenarios. In most cases, investors don't trade the same instrument at two different OMS, nor do they mix normal quotes with modified last quotes. Stock index market makers will not build multi-level quotes and all quotes will be marked with a modify-last-quote flag, thus there will be no potential self-trading problems. In most cases, treasury bond derivatives market makers do build multi-level quotes but they only use normal quotes and modify-specific quotes, thus there will also be no potential self-trading problems. Therefore, when investors believe that there is no potential self-trading problem in their quote behavior, they can ask brokers to enable the CFFEX's relaxed self-trading check function. When this function is on, CFFEX's modificate-last-quote will not be checked for self-trading with other quotes, but CFFEX's modificate-last-quote still will be checked for self-trading with other orders, which is the same as other exchanges' self-trade check rule. Relax self-trading check function will not affect the modify-specific quote's check rule.

Investors can check the YD_AF_RelaxSelfTradeCheckForQuote flag through YDAccount.AccountFlag to determine whether this feature is enabled.

7.7.6 Related Account ID

According to regulatory requirements, when exchanges implement the management rules and regulations such as position limits, transaction limits, and abnormal trading behaviors(self-trading amount, order amounts, large-value-cancel orders amount, intraday open volume), all related accounts' trade and position should be treated as one's. To help brokers and investors fulfill their regulatory obligations, starting at the 1.386 version, YD OMS supports set accounts to related. Related accounts should be set according to broker, regulatory requirements, or investors; otherwise, YD OMS will not assume the accounts are related to each other.

YD supports the self-trade check based on related accounts: All related accounts will be treated as one when making a self-trade check. If one of the related accounts' self-trade check function is off, it will not be checked with other related accounts.

Since other amount limits of related accounts can be achieved through setting amount limits on each account, they are not supported in the related accounts function. If you have any support needs, please contact us.

Related accounts' account information is only saved locally, and will not be sent through API. Investors may consult their brokers for specific settings.

7.7.7 Monotonic increase check of order reference number

In previous API versions, there were no increase requirements for OrderRef for up-bound instructions such as orders and quotes, and investors could set OrderRef as needed. During the production, the UDP interfaces of YD OMSs have ever received a large number of repeated orders due to network reasons, affecting investors trades.

In order to avoid such problems, it is suggested that investors use non-zero order groups to submit orders and avoid repeated orders through the check function. Refer to [Order Group](#) for details.

In previous API versions, in order to solve the above problems, YD has enabled the order reference number check function for each new account on the management ends by default. The OMSs could be used to check whether the OrderRef of investors with this function enabled is monotonically increasing. If a non-monotonic increase order is received through OrderRef, an error with the ErrorNo being YD_ERROR_InvalidOrderRef=78 will be sent back.

The scope of monotonic increase check for order reference numbers is limited to API instances, namely, orders submitted under the same API instance require a monotonic increase, but no restrictions are made for the OrderRef of orders among different API instances created under the same account. Therefore, under multiple connections, using the last few digits of the OrderRef as the SessionID is still valid. Raw protocol orders are checked as one separate source and do not conflict with API's UDP orders. If an OMS is restarted (e.g. starting day trading) while the client is not restarted because of the enabled HA, the order reference number can start from the beginning.

Investors can check the YD_AF_OrderRefCheck flag for having been set for YDAccount.AccountFlag to make sure that this function has been enabled. If this flag is set, it means that the monotonic increase check function for order reference numbers has been enabled.

In order to avoid unnecessary repeated checks, it is suggested that investors disable the above-mentioned order reference number check function when conducting order services through non-zero order groups. If an investor still conducts order services through an order group with the reference number being 0 and does not want the order reference number check to affect the coding logic of the existing OrderRef, he/she can also ask the broker to turn off the order check On/Off under his/her account.

7.7.8 Counter flow control

To prevent the concentration of orders from certain investors from triggering [seat flow control](#) and affecting the normal trading of other investors on the same counter, brokers can implement counter flow control to limit the speed at which investors send instructions to the counter per second. The flow control is applied collectively by calculating the total number of login connections from multiple sessions under the same funding account. The following instructions sent by investors to the counter will be included in the flow control, including order submission, order cancellation, RFQ, combination and decombination, and exercise and fulfillment instructions. Regardless of whether the instructions are rejected by the counter, they will be counted in the flow control. Each order submission and cancellation included in a batch order submission and cancellation instruction is counted separately in the counter flow control. The following two scenarios are not affected by counter flow control:

- Administrators can trade on behalf of investors at the counter, including scenarios such as forced liquidation and forced reduction will be implemented for extreme market conditions or when investors encounter technical malfunctions, administrators assist in closing positions or canceling orders. In order to quickly mitigate risks, the trading instructions from administrators are not counted in the counter flow control.
- Considering that market makers at the counter are usually exclusive, the quote and quote cancellation instructions from market makers are not counted in the counter flow control.

Investors can obtain the counter flow control threshold from YDAccount.MaxRequestSpeed. If the threshold is set to -1, it indicates no counter flow control. When an investor's instruction is subject to counter flow control, the ErrorNo in the response will be YD_ERROR_AccountRequestTooFast=24.

7.8 Trading information query

ydApi cannot be used for storing flows, so the following query function can only be used under ydExtendedApi.

Query statements are subject to slow calling and should not be called under trading threads. Frequent query statement calling is not suggested for fear of system lagging.

7.8.1 Order query

```
1  /// getOrder by orderSysID can only be used for orders have been accepted by exchange
2  virtual const YDExtendedOrder *getOrder(int orderSysID, const YDExchange *pExchange, int
   YDOrderFlag=YD_YOF_Normal)
3  virtual const YDExtendedOrder *getOrder(long long longOrderSysID, const YDExchange *pExchange, int
   YDOrderFlag=YD_YOF_Normal)
```

The above method can be used to query **orders with received notifications from exchanges** or quote derived orders. All orders submitted through the insertOrder series order submission method can be queried through this function. By default, normal orders can be queried, and other types of orders can be queried by setting the YDOrderFlag.

```
1  /// getOrder by orderRef can only be used for orders using checkAndInsertOrder
2  virtual const YDExtendedOrder *getOrder(int orderRef, unsigned orderGroupID=0, const YDAccount
   *pAccount=NULL)=0;
3
4  /// getQuoteDerivedOrder can only be used for orders derived order by using checkAndInsertQuote
5  virtual const YDExtendedOrder *getQuoteDerivedOrder(int orderRef, int direction, unsigned
   orderGroupID=0, const YDAccount *pAccount=NULL);
```

Orders submitted using checkAndInsertOrder of ydExtendedApi can be queried through the above two methods. As long as checkAndInsertOrder and checkAndInsertQuote calls are completed, they can be queried through these two methods without receiving any exchange's notification. Among them, getOrder can be used to query orders except for those quote derived ones, while getQuoteDerivedOrder can be used to query quote derived orders. Compared to getOrder, order direction parameters are added to make a distinction between derived buy and sell orders under two-sided quote.

```
1  /// orders must have spaces of count, return real number of orders(may be greater than count). Only
   partial will be set if no enough space. Only orders accepted by exchange can be found in this function
2  virtual unsigned findOrders(const YDOrderFilter *pFilter, unsigned count, const YDExtendedOrder
   *orders[])=0;
3
4  /// User should call destroy method of return object to free memory after using following method
5  virtual YDQueryResult<YDExtendedOrder> *findOrders(const YDOrderFilter *pFilter);
6  virtual YDQueryResult<YDExtendedOrder> *findPendingOrders(const YDOrderFilter *pFilter);
```

YD provides the above-mentioned two methods for multi-query of **orders with received notifications from exchanges**. Their query criteria are used in the same way. The pseudo code logic for determining whether a certain order meets the query criteria is as follows:

```
1  if YDOrder.OrdersSysID < 0
2      return false
3
4  if YDOrder.YDOrderFlag not in YDOrderFilter.YDOrderFlags
5      return false
6
```

```

7  if YDOrderFilter.StartTime >= 0 and YDOrder.InsertTime > YDOrderFilter.StartTime
8      return false
9
10 if YDOrderFilter.EndTime >= 0 and YDOrder.InsertTime < YDOrderFilter.EndTime
11     return false
12
13 if YDOrderFilter.pExchange != NULL and YDOrderFilter.pExchange != YDOrder.pExchange
14     return false
15
16 if YDOrder.YDOrderFlag == YD_YOF_CombPosition
17     if YDOrderFilter.pCombPositionDef != NULL and YDOrderFilter.pCombPositionDef !=
YDOrder.pCombPositionDef
18         return false
19 else
20     if YDOrderFilter.pInstrument != NULL and YDOrderFilter.pInstrument != YDOrder.pInstrument
21         return false
22     if YDOrderFilter.pProduct != NULL and YDOrderFilter.pProduct != YDOrder.pProduct
23         return false
24 return true

```

Each field to be filled out is described as follows:

Parameter	Field	Description
YDOrderFilter	StartTime	The start time in the form of TimeID. -1 means unlimited. See the following example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h
	EndTime	The end time in the form of TimeID. -1 means unlimited. See the following example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h
	YDOrderFlags	For setting the flag bits of YDOrderFlag to query. More than one flag bit can be set. For example, when querying normal and combined orders, the following expression setting can be used: $1 << YD_YOF_Normal \mid 1 << YD_YOF_CombPosition$
	pCombPositionDef	Definition traditional combined positions. For YD_YOF_CombPosition orders, only this parameter and pExchange can be checked. When set to NULL, it means that no limit will be made.
	pInstrument	Instrument pointer. It is valid for non-YD_YOF_CombPosition orders. When set to NULL, it means that no limit will be made.
	pProduct	Product pointer. It is valid for non-YD_YOF_CombPosition orders. When set to NULL, it means that no limit will be made.
	pExchange	Exchange pointer. When set to NULL, it means that no limit will be made.
	pAccount	The "Investor" should always be set to NULL

The first method requires investors to allocate a fixed-length YDExtendedOrder pointer array in advance. If the pre-allocated length is not enough for the space, only the pre-allocated array length orders can be filled out. Whether the pre-allocated length is enough or not for the space, the return values regarding this method are the total orders meeting the query criteria. Investors, relying on this, can call findOrders(pFilter, 0, NULL) to quickly obtain the total orders meeting the criteria.

- The advantage of this method is that when there are not many orders and the length of the pre-allocated array is enough, the pre-allocated pointer array can be reused without allocating for each query;
- The disadvantage of this method is that if there are many orders, it may need to be called twice (the first time aims to obtain the total orders, the second time, to allocate the array that can include all orders before calling again) to fully obtain all orders meeting the criteria.

The second method can help investors allocate a space for all orders meeting the criteria. However, the allocated space, after being used, should be destroyed by investors actively by calling YDQueryResult.destory(). Compared with the first method:

- The advantage of this method is that all orders can be notified by one call
- The disadvantage of this method is that the investors need to actively release the space allocated according to this method, and each call will lead to the allocation of a new space, however, frequent allocation and release are very unfriendly to the cache.

7.8.2 Trade query

```

1  /// trades must have spaces of count, return real number of trades(may be greater than count). Only
    partial will be set if no enough space
2  virtual unsigned findTrades(const YDTradeFilter *pFilter,unsigned count,const YDExtendedTrade *trades[]);
3
4  /// User should call destroy method of return object to free memory after using following method
5  virtual YDQueryResult<YDExtendedTrade> *findTrades(const YDTradeFilter *pFilter);

```

The method for multi-query of trades is similar to that of [Order Query](#). Please refer to [Order Query](#) for the details. The pseudocode logic for determining whether a trade meets the query criteria is as follows:

```

1  if YDTradeFilter.StartTime >= 0 and YDTrade.InsertTime > YDTradeFilter.StartTime
2      return false
3
4  if YDTradeFilter.EndTime >= 0 and YDTrade.InsertTime < YDTradeFilter.EndTime
5      return false
6
7  if YDTradeFilter.pInstrument != NULL and YDTradeFilter.pInstrument != YDTrade.pInstrument
8      return false
9
10 if YDTradeFilter.pProduct != NULL and YDTradeFilter.pProduct != YDTrade.pProduct
11     return false
12
13 if YDTradeFilter.pExchange != NULL and YDTradeFilter.pExchange != YDTrade.pExchange
14     return false
15
16 return true

```

Each field to be filled out is described as follows:

Parameter	Field	Description
YDTradeFilter	StartTime	The start time in the form of TimeID. -1 means unlimited. See the following example: At 21:00 in night trading hours: 3600*(21-17) = 14400 At 9:00 in day trading hours: 3600*(24+9-17) = 57600 At 9:00 in Monday's trading hours: 3600*(24+9-17) = 5760 For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h
	EndTime	The end time in the form of TimeID. -1 means unlimited. See the following example: At 21:00 in night trading hours: 3600*(21-17) = 14400 At 9:00 in day trading hours: 3600*(24+9-17) = 57600 At 9:00 in Monday's trading hours: 3600*(24+9-17) = 5760 For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h
	pInstrument	Instrument pointer. When set to NULL, it means that no limit will be made.
	pProduct	Product pointer. When set to NULL, it means that no limit will be made.
	pExchange	Exchange pointer. When set to NULL, it means that no limit will be made.
	pAccount	The "Investor" should always be set to NULL

7.8.3 Quote query

```

1  /// getQuote by quoteSysID can only be used for quotes have been accepted by exchange
2  virtual const YDExtendedQuote *getQuote(int quoteSysID,const YDExchange *pExchange)
3  virtual const YDExtendedQuote *getQuote(long long longQuoteSysID,const YDExchange *pExchange)

```

The above method can be used to query **quotes with received notifications from exchanges**. All quotes submitted through the insertQuote series quote service method can be queried through this function.

```

1  /// getQuote by orderRef can only be used for quotes using checkAndInsertQuote
2  virtual const YDExtendedQuote *getQuote(int orderRef,unsigned orderGroupID=0,const YDAccount
    *pAccount=NULL);

```

Quotes submitted using checkAndInsertQuote of ydExtendedApi can be queried through the above method. As long as checkAndInsertQuote calls are completed, they can be queried through these two methods without receiving any exchange's notification.

```

1  /// quotes must have spaces of count, return real number of quotes(may be greater than count). Only
    partial will be set if no enough space. Only quotes accepted by exchange can be found in this function
2  virtual unsigned findQuotes(const YDQuoteFilter *pFilter,unsigned count,const YDExtendedQuote *quotes[]);
3
4  /// User should call destroy method of return object to free memory after using following method
5  virtual YDQueryResult<YDExtendedQuote> *findQuotes(const YDQuoteFilter *pFilter);
6  virtual YDQueryResult<YDExtendedQuote> *findPendingQuotes(const YDQuoteFilter *pFilter);

```

The method for multi-query of quotes is similar to that of [Order Query](#). Refer to [Order Query](#) for the details. The pseudocode logic for determining whether a quote meets the query criteria is as follows:

```

1  if YDQuote.OrdersSysID < 0
2      return false
3
4  if YDQuoteFilter.StartTime >= 0 and (any DerivedOrder of YDQuote).InsertTime > YDQuoteFilter.StartTime
5      return false
6
7  if YDQuoteFilter.EndTime >= 0 and (any DerivedOrder of YDQuote).InsertTime < YDQuoteFilter.EndTime
8      return false
9
10 if YDQuoteFilter.pInstrument != NULL and YDQuoteFilter.pInstrument != YDQuote.pInstrument
11     return false
12
13 if YDQuoteFilter.pProduct != NULL and YDQuoteFilter.pProduct != YDQuote.pProduct
14     return false
15
16 if YDQuoteFilter.pExchange != NULL and YDQuoteFilter.pExchange != YDQuote.pExchange
17     return false
18
19 return true

```

Each field to be filled out is described as follows:

Parameter	Field	Description
YDTradeFilter	StartTime	<p>The start time in the form of TimeID. -1 means unlimited. Any derived order of a quote that is later than the start time can be considered as valid.</p> <p>See the following example:</p> <p>At 21:00 in night trading hours: $3600 \times (21-17) = 14400$</p> <p>At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$</p> <p>At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$</p> <p>For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h</p>
	EndTime	<p>The end time in the form of TimeID. -1 means unlimited. Any derived order of a quote that is earlier than the start time can be considered as valid.</p> <p>See the following example:</p> <p>At 21:00 in night trading hours: $3600 \times (21-17) = 14400$</p> <p>At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$</p> <p>At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$</p> <p>For the conversion between TimeID and reference time, see string2TimeID and timeID2String of ydUtil.h</p>
	pInstrument	Instrument pointer. When set to NULL, it means that no limit will be made.
	pProduct	Product pointer. When set to NULL, it means that no limit will be made.
	pExchange	Exchange pointer. When set to NULL, it means that no limit will be made.
	pAccount	The "Investor" should always be set to NULL

7.9 Trading segment

YD supports sending summarized and detailed trading segment announcements:

- Summarized trading segment announcements are those showing continuous information about trading segments. In order to reduce the communication traffic and the impact on API, summarized trading segment announcements only show the different time points for changing status announced by client exchanges. If several events occur at the same time, only the first one will be sent, namely for events occurred at the same time, only the first one will be pushed.
- Detailed trading segment announcements are those that are not sent by default. Trading segment announcements of some exchanges are made based on instruments, so a large number of announcements can be sent to all investors at the same time, which can cause Internet congestion if some notifications regarding orders are sent at this time. Therefore, detailed trading segment announcements should be used prudently. Once this function is enabled, all investors using the same OMS will receive detailed trading segment announcements. This function can be checked for being enabled at an OMS through TradingSegmentDetail of SystemParam. Refer to [System Parameters](#) for details.

7.9.1 Summarized trading segment

The summarized trading segment information can be sent back through the following callback function. The segmentTime means the count of seconds from the beginning of a trading day to that time. For its coding method, refer to [Order Notification](#) for more details about the relevant description for InsertTime.

```

1 | virtual void notifyTradingSegment(const YDExchange *pExchange, int segmentTime)

```

The following shows different time points for changing trading statuses of current exchanges, which are only used for reference. The exchanges are subject to change without notice. The actual receiving time points during production shall prevail.

Exchange	Day trading hours	*Night trading hours
CFEX	09:25:00 09:29:00 09:30:00 11:30:00 13:00:00 14:57:00 15:00:00 15:15:00	
INE	09:00:00 10:15:00 10:30:00 11:30:00 13:30:00 15:00:00	20:55:00 20:59:00 21:00:00 23:00:00 01:00:00 02:30:00
SHFE	08:55:00 08:59:00 09:00:00 10:15:00 10:30:00 11:30:00 13:30:00 15:00:00	20:55:00 20:59:00 21:00:00 23:00:00 01:00:00 02:30:00
DCE	08:54:50 08:55:00 08:58:50 08:59:00 08:59:50 09:00:00 10:14:50 10:15:00 10:29:50 10:30:00 11:29:50 11:30:00 13:29:50 13:30:00 14:59:50 15:00:00	20:54:50 20:55:00 20:58:50 20:59:00 20:59:50 21:00:00 22:59:50 23:00:00

7.9.2 Detailed trading segment

The detailed trading segment information can be sent back through the following callback function. The dimensions of the returned trading stages may vary among different exchanges, possibly returning the detailed trading segments based on the dimensions of exchange, product, or instrument. The detailed trading segments will be notified to investors after the "notifyFinishInit" function, but investors should not assume that all statuses of exchanges, products, or instruments can be collected before the "notifyCaughtUp" function. Some detailed trading segments may be received after the "notifyCaughtUp" function. Therefore, the strategy program should consider the trading segment of an instrument as non-trading until receiving the corresponding notification of the detailed trading segments. OMS will ensure that the trading status of all instruments will be correctly sent, regardless of whether the OMS has been restarted during market hours.

The reasons for not being able to receive all the details before notifyCaughtUp can be primarily attributed to two factors. Firstly, it could be because the market is currently in a pre-trading phase, and the exchange has not yet pushed any detailed transaction segments. Secondly, it is because of the way the OMS sends transactions, if there are updated detailed transaction segments during the process of sending transactions, the older versions of those segments will be skipped. If the segment number of a new version of a detailed transaction is greater than the maximum order number at the time of API login, then the record of that detailed transaction segment will be sent after the function of notifyCaughtUp.

```
1 | virtual void notifyTradingSegmentDetail(const YDTradingSegmentDetail *pTradingSegmentDetail)
```

The fields of YDTradingSegmentDetail sent back are described below.

Field	Description
ExchangeRef	Exchange reference No. It is helpful when the detailed trading segment information relating to an exchange is sent back, otherwise it will be -1.
ProductRef	Product reference No. It is helpful when the detailed trading segment information relating to a product is sent back, otherwise it will be -1.
InstrumentRef	Instrument reference No. It is helpful when the detailed trading segment information relating to an instrument is sent back, otherwise it will be -1.
m_pExchange	Exchange pointer. It is helpful when the detailed trading segment information relating to an exchange is sent back, otherwise it will be NULL.

Field	Description
m_pProduct	Product pointer. It is helpful when the detailed trading segment information relating to a product is sent back, otherwise it will be NULL.
m_pInstrument	Instrument pointer. It is helpful when the detailed trading segment information relating to an instrument is sent back, otherwise it will be NULL.
SegmentTime	The count of seconds from the beginning of a trading day to that time. For its coding method, see the relevant description for InsertTime in Order Notification .
TradingStatus	Trading segment status: YD_TS_NoTrading: Non-Trading YD_TS_Continuous: Continuous trading YD_TS_Auction: Call auction

7.10 Order group

In order to better support investors' need for making a distinction between orders and monotonic increase check of order reference numbers, YD has added an order group function in Vers. '1.280'. Investors can set OrderGroupID and GroupOrderRefControl in YDInputOrder and YDInputQuote to use the order group function.

Important

When version 1.280 introduced the order group function, it supported a maximum of 63 order groups. Version 1.486 increased the total number of order groups from 63 to 255. When using the API before version 1.486 to connect to the counter of version 1.486, you can still only use 63 order groups. To use the maximum of 255 order groups, you must upgrade both the API and the counter to version 1.486 or later.

Field	Description
OrderGroupID	For specifying the logical group to which orders and quotes belong, which can be used to determine the strategy, connection, and other logical groups to which the orders belong. Investors can use Order Groups 0-255. To ensure the compatibility, for Order Group 0, the monotonic increase of OrderRef will not be checked, while for Order Group 1-255, the OrderRef monotonicity will be checked according to the settings
GroupOrderRefControl	For specifying the method required by investors for checking the monotonic increase of order orderRef at the OMSs: YD_GORF_Increase: Monotonic increase of OrderRef. The gap between two OrderRef numbers must be higher than or equal to 1 YD_GORF_IncreaseOne: The OrderRef numbers should be strictly monotonically increased, and the gap between two OrderRef numbers must be 1

After specifying OrderGroupID and GroupOrderRefControl for an order and a quote, the OMS will check the monotonicity of the order group to which the order belongs. If the check fails, an ErrorNo=YD_ERROR_InvalidGroupOrderRef failed order will be sent back through notifyOrder, and the current maximum OrderRef of OrderGroupID will be set in the MaxOrderRef notification. Whether the order is handled or not, the OrderGroupID and GroupOrderRefControl set for YDInputOrder and YDInputQuote will be sent back through YDOrder and YDQuote.

Specifying OrderGroupID and GroupOrderRefControl on each order is a great help for investors' easy operation. For orders with the same OrderGroupID, investors can set YD_GORF_Increase for the first one to enable the monotonic increase check function and YD_GORF_IncreaseOne for the second one to enable the stringent monotonic increase check function.

After the callback function notifyLogin for successful login for the first time or successful reconnection/login after disconnection, the OMS will send back the current maximum OrderRef for each OrderGroupID through the following callback function. Please be sure to record the return values in relation to this callback and use OrderRef numbers that meet the increase specification for subsequent orders.

```
1 | virtual void notifyGroupMaxOrderRef(const int groupMaxOrderRef[])
```

For investors using ydExtendedApi, the following method can be used to directly obtain the next OrderRef for orders or quotes:

```
1 | virtual int getNextOrderRef(unsigned orderGroupID, bool update=true);
```

orderGroupID specifies the order group numbers. If orderGroupID is set to 0, the next OrderRef rules can be set, see [Multiple Connections](#); If orderGroupID is set to 1 to 255, the next OrderRef must be the current maximum OrderRef plus 1.

The "update" indicates whether the maximum order reference number of the order group needs to be updated after obtaining the next OrderRef. Assuming that the next OrderRef sent back after calling getNextOrderRef is n, if the "update" for this call is "True", the next OrderRef sent back after calling getNextOrderRef next time will be n+1; If the "update" for this call is "False", the next OrderRef sent back after calling getNextOrderRef next time will still be n.

7.11 Multiple connections

By default, YD OMSs support multiple API instance connections to OMSs at the same time under the same fund account in addition to unlimited connections. A broker can control the maximum login connections for each investor at the OMS end. Investors can query whether the broker has set up an upper limit of connections through MaxLoginCount of YDAccount. If no any limit is made, the MaxLoginCount will be -1. At the same time, the LoginCount of YDAccount records the current total connections to the fund

account. For different connections under the same fund account, MaxLoginCount and LoginCount are always the same.

If investors need to make a distinction among different connection orders, the [Order Group](#) function should be preferred. It is **not recommended** to use the native SessionID mechanism mentioned in the following text.

In version 1.386, YD provides a native SessionID mechanism, which is designed to meet the regulatory requirements of the Shanghai and Shenzhen Stock Exchanges. However, this mechanism was not designed with consideration for investors' trading needs.

Therefore, it is not recommended for investors to use this SessionID mechanism in the trading process. However, it can be logged for troubleshooting purposes in the future. This mechanism is enabled by default in the spot trading OMS but disabled by default in the futures trading OMS. If you need to use it in the futures trading OMS, please contact your broker to enable this feature on the OMS side.

The 'SessionID' is assigned by the OMS, and a new 'SessionID' is allocated when the API establishes an initial connection or reconnects after disconnection. The maximum number of connections for all investors is 4096, and SessionID numbers range from 0 to 4095. The system randomly assigns SessionIDs and does not allocate them in the order of connection. After a disconnection, the corresponding SessionID will be reclaimed for reuse. Therefore, during different times of the same trading day, SessionIDs may be duplicated among different investors, and even the same investor may receive the same SessionID again. However, within the same time period, the SessionIDs of different connections are always different. Therefore, it is strongly recommended that investors always obtain the 'getSessionID' through 'notifyLogin'. The function to obtain the 'SessionID' is shown below:

```
1 | virtual int getSessionID(void)
```

The 'SessionID' for orders and quotes can be obtained from 'YDOrder.SessionID' and 'YDQuote.SessionID', respectively. After a regular OMS restart or failover switch between primary and standby counterparties, the 'SessionID' in the order feedback returned by the OMS will be set to -1.

For previous API versions, YD provided investors using ydExtendedApi with a low-order coding SessionID mechanism with OrderRef. However, considering many restrictions, YD no longer suggests its preference. In order to maintain the compatibility, this mechanism is still reserved for the order group numbered 0 in the current version. If investors use non-zero order groups, the following is invalid.

Please note that once decided, this mechanism should be configured for all connections, otherwise it may lead to potential OrderRef coding conflicts. For multi-connection investors who have used [Local Risk Control Order](#), the SessionID mechanism must be used, otherwise all OrderRef numbers issued through the connections will conflict.

This connection number mechanism involves the following functions:

```
1 | virtual bool setSessionOrderRefRule(unsigned sessionBitCount, unsigned sessionID);
2 | virtual void getSessionOrderRefRule(unsigned *pSessionBitCount, unsigned *pSessionID);
```

Please set the thread number and global thread number rules through setSessionOrderRefRule **before submitting any order and quote**. The parameters of setSessionOrderRefRule are described as follows:

- An end digit reserved for SessionID is set in sessionBitCount when OrderRef is used, which can be used for determining the maximum counts of connections and orders. The reserved digit can have at most 16 bits and must be the same for all connections
 - The sessionID marks the connection number of this connection. Please number each API instance connection from 0 and make sure that the total connections be kept within the range specified by sessionBitCount
- For example, assuming that the set sessionBitCount of a connection is 8 and SessionID is 3, then 8 bits in the OrderRef are used for expressing the connection number, and the remaining 24 bits (32-8=24 bits), expressing the actual order number. Therefore, under this setting mode, there can be 256 (2^8) connection numbers, and each connection can have 16,777,216 (2^{24}) orders, and the connection number for this API instance is 3. The first OrderRef for this connection is 259 (0b100000000+0b11=0b100000011=259).

After setting, the getSessionOrderRefRule can be used to obtain the set connection number coding rules, and the getNextOrderRef, to obtain the next order reference number.

7.12 Raw protocol

If a '1.280' or higher version API is used, the raw protocol below should be followed strictly for submitting orders and receiving notifications, otherwise they will be considered as failed orders. If a '1.188' or lower version API is used, please refer to the corresponding raw protocol versions.

7.12.1 Up-bound message

Starting from Version '1.52', YD has opened the order submission and cancellation message services, and added the quote submission, cancellation and various notification message services in Version '1.280'. Investors can send self-compiled UDP messages to YD OMSs for trading. Since the raw protocol order submission means that users can combine and send their own data packets, its performance directly depends on users' implementation and has nothing to do with the YD API. Whether submitting orders under a raw protocol or the UDP mode of YD API, there is no difference in look-through performance at the OMS ends. Therefore, comparing with API, the performance only differs at client sending ends.

After a long period of time of iteration, the performance of YD API tested in a YD's laboratory (through X10/25 and X2522 network cards) has reached the conditions for common FPGAs. If a customer wants to submit orders based on a raw protocol, the quicker method for production order submission should be selected after completing the implementation and comparing it with the order performance of YD API.

API provides the timestamp function `getYDNanoTimestamp`, which, relying on its high accuracy and quick speed, can be used for testing the API's order submission speed. The specific method is calling `getYDNanoTimestamp` before and after `insertOrder` and subtracting the results. The final result is the time difference between the sending time and the first byte appearing on the optical fiber.

The order submission & cancellation under a raw protocol can be conducted through UDP or XTCP. The UDP or XTCP port number can be found in `SystemParam`. Refer to [System Parameters](#) for details. Since being excluded in YD's check, any source port number can be used. When using the XTCP raw protocol for order placement, please send a [heartbeat message](#) to the OMS every 2 seconds to maintain the connection.

7.12.1.1 Preparation before operation

Investors can use the raw protocol after the approval of the broker. Just contact the broker to enable the protocol before using. Investors can check `YD_AF_BareProtocol` for being set for `YDAccount.AccountFlag` to confirm whether the raw protocol has been enabled or not.

UDP message headers should be obtained by calling `getClientPacketHeader`. A message header contains information such as account and key, so no part of the message header can be modified. The message header can only be obtained during `notifyFinishInit` operation and its subsequent steps. Once a message header is successfully obtained, it will not change during the OMS operation. It should be obtained again once the OMS is restarted. Investors using raw protocol orders on the spot trading OMS, please take note. Due to the requirement of enabling the [Native SessionID Mechanism](#) on the spot trading OMS, if a disconnection and reconnection occur during trading hours, the raw protocol message header before the disconnection will become invalid. It is essential to retrieve the message header again in the 'notifyLogin' after a successful login.

```
1  /// definition of protocolVersion
2  ///      0: newest protocol version
3  ///      1: protocol for api version up to 1.188
4  ///      2: protocol for api version up to 1.386
5  ///      3: protocol for api version from 1.486, current newest version
6  virtual int getClientPacketHeader(YDPacketType type,unsigned char *pHeader,int len,int protocolVersion=0);
```

The `YDPacketType` can be `YD_CLIENT_PACKET_INSERT_ORDER`, `YD_CLIENT_PACKET_CANCEL_ORDER`, `YD_CLIENT_PACKET_INSERT_QUOTE` and `YD_CLIENT_PACKET_CANCEL_QUOTE`, which are used for obtaining message headers in relation to order submission/cancellation and quote submission/cancellation. Starting from version 1.280, YD has added four special types of order cancellation: `YD_CLIENT_PACKET_INSERT_NORMAL_ORDER`, `YD_CLIENT_PACKET_CANCEL_NORMAL_ORDER`, `YD_CLIENT_PACKET_INSERT_SPECIAL_ORDER`, and `YD_CLIENT_PACKET_CANCEL_SPECIAL_ORDER`. The first two can only be used for regular order cancellation, and if used for other business types, an error will occur. The last two can only be used for submitting and withdrawing non-trading operations such as exercise, abandonment of exercise, and covered positions. If used for regular order cancellation, an error will occur.

The "header" and "len" refer to the header pointer and length of the memory area created in advance by a user. The header is used for containing the header data generated by api. The len should be longer than the length of the corresponding type of header. At present, the header length of each operation is 16.

The 'protocolVersion' can be specified to indicate the version of the protocol message. The default version is the latest protocol message version. After selecting the specified protocol version, you need to refer to the protocol message description in the API document of the corresponding version. The following table shows the actual api versions of corresponding protocol message used by the protocol version. Investors using protocol version 0 should pay attention to whether the implementation is compatible with the protocol message in the latest version.

Version	InsertOrder	CancelOrder	InsertQuote	CancelQuote
1	1.188	1.188	1.280	1.280
2	1.280	1.280	1.280	1.280
3	1.280	1.280	1.486	1.486
0	1.280	1.280	1.486	1.486

As functions are added, the message length of the new version will increase. Investors need to choose message versions carefully for the following reasons:

*The OMS will check the consistency of the packet length in the packet header with the actual length of the packet. If a mismatched packet header and packet body are used, the OMS will discard it as an error message. For example, after upgrading to API version 1.486, the default quotation header length of this version is 88. If the program still sends a quotation with a length of 64 which used in the 1.280 version document, the quotation will be discarded by the OMS. Therefore, it is recommended that investors clearly specify the protocol message version to avoid compatibility issues after upgrading.

*The messages used by the old version are usually shorter; in that, the client might have a slight latency advantage at packaging, sending and receiving messages from the OMS. If you do not use any of the features from the new version, latency-sensitive investors can choose to continue using the old version. YD promises that the new version of the OMS is compatible with the old version of the message protocol.

The return value refers to the actual length of a returned message header. If the return value is 0, it means that message header obtaining fails, since, usually, the length of the pre-created memory area is insufficient, and the calling time is earlier than that for `notifyFinishInit`, or no raw protocol-based order submission function `YD_AF_BareProtocol` is enabled.

7.12.1.2 Order submission message

The following shows the message structure when submitting an order, which is equivalent to calling insertOrder.

Address offset	Length (bytes)	Field type	Description
0	16	Integer	Message header.
16	4	Integer (little-endian)	Instrument reference No. It can be obtained through the InstrumentRef of YDInstrument.
20	1	Integer	Trading direction (0: buy, 1: sell)
21	1	Integer	Position opening and closing flags (0: Position opening, 1: Position closing, 3: Today's position closing, 4: Pre position closing)
22	1	Integer	Hedge flags (1: Speculation, 2: Arbitrage, 3: Hedge)
23	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
24	8	IEEE 754 Double-accuracy float type (little endian)	Price
32	4	Integer (little-endian)	Order volume
36	4	Integer (little-endian)	Order reference
40	1	Integer	Order Type (0: Price-limited order, 1: FAK, 2: Market order, 3: FOK)
41	1	Integer	0
42	1	Integer	Connection number
43	5	Integer	0
48	1	Unsigned integer	Order group ID
49	1	Integer	OrderRef control mode of order groups, refer to Order Group 0: Monotonic increase OrderRef numbers. The gap between two OrderRef numbers must be higher than or equal to 1 1: The OrderRef numbers should be strictly monotonically increased, and the gap between two OrderRef numbers must be 1
50	1	Integer	Triggered order Type 0: No trigger 1: Profit taking trigger 2: Loss stopping trigger
51	1	Integer	The order attributes of the exchange, which shall be interpreted by the exchange to which the order belongs YD_EOA_DCE_GIS=1: DCE GIS(Good in Session)
52	4	Integer	Please refer to Remote User-defined field
56	8	Integer	0
64	8	IEEE 754 Double-accuracy float type (little endian)	Trigger price of triggered Order

7.12.1.3 Order cancellation message

The following shows the message structure for order cancellation, which is equivalent to calling cancelOrder. At present, three order cancellation modes i.e. full-accuracy exchange order reference No., exchange order reference No. and commissioned number OrderRef are supported. Refer to [Normal Order Cancellation](#) for details.

Address offset	Length (bytes)	Field type	Description
0	16	Integer	Message header.
16	4	Integer (little-endian)	Exchange order number or OrderRef for orders to be cancelled.
20	1	Integer	Exchange SN. Obtained from ExchangeRef of YDExchange.
21	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
22	1	Integer	Connection number
23	1	Integer	0

Address offset	Length (bytes)	Field type	Description
24	1	Unsigned integer	Order group ID
25	7	Integer	0
32	8	Integer (little-endian)	Full-accuracy exchange order ID No.

7.12.1.4 Quote submission message

The following shows the message structure when submitting a quote which is equivalent to calling insertQuote.

Address offset	Length (bytes)	Field type	Description
0	16	Integer	Message header.
16	4	Integer	Instrument reference No. It can be obtained through the InstrumentRef of YDInstrument.
20	1	Integer (little-endian)	Bid offset flag
21	1	Integer	Bid hedge flag
22	1	Integer	Ask offset flag
23	1	Integer	Ask hedge flag
24	8	IEEE 754 Double-accuracy float type (little endian)	Bid price
32	8	IEEE 754 Double-accuracy float type (little endian)	Ask price
40	4	Integer (little-endian)	Bid volume
44	4	Integer (little-endian)	Ask volume
48	4	Integer (little-endian)	Order reference, namely OrderRef
52	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
53	1	Integer	Connection number
54	1	Integer	0
55	1	Integer	Quote flag YD_YQF_ResponseOfRFQ: For automatically filling in the RFQ number to indicate the response price. It is supported by SHFE, INE, DCE, GFEX and CZCE. Other exchanges do not provide RFQ numbers. YD_YQF_ReplaceLastQuote: Whether to replace the last quote, only for CFFEX. YD_YQF_ReplceSpecifiedQuote: Whether to replace the specified quote, only for CFFEX.
56	1	Unsigned integer	Order group ID
57	1	Integer	OrderRef control mode of order groups, see Order Group 0: Monotonic increase OrderRef numbers. The gap between two OrderRef numbers must be higher than or equal to 1 1: The OrderRef numbers should be strictly monotonically increased, and the gap between two OrderRef numbers must be 1
58	1	Integer	The order attributes of the exchange, which shall be interpreted by the exchange to which the order belongs YD_EOA_DCE_GIS=1: DCE GIS(Good in Session)
59	5	Integer	0
64	4	Integer (little-endian)	Please refer to Remote User-defined field
68	12	Integer	0
80	8	Integer (little-endian)	QuoteSysID used by CFFEX quote modification.

7.12.1.5 Quote cancellation message

The following shows the message structure for quote cancellation, which is equivalent to calling cancelQuote. At present, three quote cancellation modes i.e. full-accuracy exchange order reference No., exchange order reference No. and commissioned number OrderRef are supported. Refer to [Normal Quote Cancellation](#) for details.

Address offset	Length (bytes)	Field type	Description
0	16	Integer	Message header.
16	4	Integer (little-endian)	Exchange quote number or OrderRef for quotes to be cancelled.
20	1	Integer	Exchange SN. Obtained from ExchangeRef of YDExchange.
21	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
22	1	Integer	Connection number
23	1	Unsigned integer	Order group ID
24	8	Integer (little-endian)	Full-accuracy exchange quote number
32	1	Integer	The method of cancelling quote. (Only for SSE, SZSE can only cancel quotes by both sides simultaneously) YD_CQIT_Buy=1: cancel buy side YD_CQIT_Sell=2: cancel sell side YD_CQIT_Both=3: cancel both sides
33	7	Integer	0

7.12.2 Down-bound message

The down-bound messages of YD are sent through TCP. YD will not consider adding UDP down-bound messages until it can effectively solve the problem on reliable UDP delivery. Investors who need to parse down-bound messages should bypass and monitor TCP message segments, and properly handle possible troubles such as retransmission and adhesion.

7.12.2.1 Message header

YD's down-bound messages have the same header structure. After receiving a down-bound message, the message type of the header should be parsed first to determine the message structure for parsing subsequent data. The message header structure is shown below:

Address offset	Length (bytes)	Field type	Description
0	2	Integer (little-endian)	Message length (including the header of this message)
2	2		Non-public field
4	4	Integer (little-endian)	Message type
8	4		Non-public field
12	4	Integer (little-endian)	Investor account SN

The possible message types for the above message headers are shown below:

Message type	Description
34	Order notification
35	Trade notification
37	RFQ
40	Quote notification
43	Notification for order or quote cancellation failure

7.12.2.2 Order notification message

Due to a large number of operations reusing order notification messages, please only focus on the order notification when the order flag is 0, and those non-zero order notifications should be ignored.

Address offset	Length (bytes)	Field type	Description
0	16		Message header

Address offset	Length (bytes)	Field type	Description
16	4	Integer (little-endian)	Instrument SN. Corresponding to InstrumentRef of YDInstrument.
20	1	Integer	Trading direction (0: buy, 1: sell)
21	1	Integer	Position opening and closing flags (0: Position opening, 1: Position closing, 3: Today's position closing, 4: Pre position closing)
22	1	Integer	Hedge flags (1: Speculation, 2: Arbitrage, 3: Hedge)
23	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
24	8	IEEE 754 Double-accuracy float type (little endian)	Order price
32	4	Integer (little-endian)	Order volume
36	4	Integer (little-endian)	Investor's order reference No.
40	1	Integer	Order type (0: Price limited order, 1: FAK, 2: Market order, 3: FOK)
41	1	Integer	Order flag
42	1	Integer	Specified connection number
43	1	Integer	Actually used connection number
44	4	Integer (little-endian)	Error No.
48	4	Integer (little-endian)	Exchange reference No.
52	4	Integer (little-endian)	Exchange order ID No.
56	4	Integer (little-endian)	Order status
60	4	Integer (little-endian)	Trade volume
64	4	Integer (little-endian)	Exchange order submission time
68	4	Integer (little-endian)	Local order reference No. of the OMS
72	1	Unsigned integer	Order group ID
73	1	Integer	OrderRef control mode of order groups, refer to Order Group 0: Monotonic increase OrderRef numbers. The gap between two OrderRef numbers must be higher than or equal to 1 1: The OrderRef numbers should be strictly monotonically increased, and the gap between two OrderRef numbers must be 1
74	1	Integer	Trigger order Type 0: No trigger 1: Profit taking trigger 2: Loss stopping trigger
75	1	Integer	The order attributes of the exchange, which shall be interpreted by the exchange to which the order belongs YD_EOA_DCE_GIS=1: DCE GIS(Good in Session)
76	4	Integer (little-endian)	Please refer to Remote User-defined field
80	8		Non-public field
88	8	IEEE 754 Double-accuracy float type (little endian)	Trigger price of triggered Order
96	4	Integer (little-endian)	Order trigger status 0: Not Triggered 1: Triggered
100	4	Integer (little-endian)	Millisecond counts from the beginning (17:00) to the order-insert time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h

Address offset	Length (bytes)	Field type	Description
104	8	Integer (little-endian)	Full-accuracy exchange order ID No.
112	4	Integer (little-endian)	Millisecond counts from the beginning (17:00) to the order-cancel time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
116	4	Integer (little-endian)	Connection number
120	4		Non-public field
124	4	Integer (little-endian)	sno used by GTJA Cash

7.12.2.3 Trade notification message

Address offset	Length (bytes)	Field type	Description
0	16		Message header
16	4	Integer (little-endian)	Instrument SN. Corresponding to InstrumentRef of YDInstrument.
20	1	Integer	Trading direction (0: buy, 1: sell)
21	1	Integer	Position opening and closing flags (0: Position opening, 1: Position closing, 3: Today's position closing, 4: Pre position closing)
22	1	Integer	Hedge flags (1: Speculation, 2: Arbitrage, 3: Hedge)
23	1		Non-public field
24	4	Integer (little-endian)	Exchange trade ID No.
28	4	Integer (little-endian)	Exchange order ID No.
32	8	IEEE 754 Double-accuracy float type (little endian)	Trade price
40	4	Integer (little-endian)	Trade volume
44	4	Integer (little-endian)	Exchange trade time
48	8	IEEE 754 Double-accuracy float type (little endian)	Trade commission
56	4	Integer (little-endian)	Local order reference No. of OMS
60	4	Integer (little-endian)	Investor's order reference No.
64	1	Unsigned integer	Order group ID
65	1	Integer	Actually used connection number
66	2		Non-public field
68	4	Integer (little-endian)	Millisecond counts from the beginning (17:00) to the trade time of a trading day. For example: 500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$ 500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$ 500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$ For the conversion between YD's timestamp time and reference time, see string2TimeStamp and timeStamp2String of ydUtil.h
72	8	Integer (little-endian)	Full-accuracy exchange order ID No.
80	8	Integer (little-endian)	Full-accuracy exchange trade ID No.
88	4	Integer (little-endian)	Please refer to Remote User-defined field
92	4		Non-public field

7.12.2.4 RFQ notification message

Address offset	Length (bytes)	Field type	Description
0	16		Message header
16	4	Integer (little-endian)	Instrument SN. Corresponding to InstrumentRef of YDInstrument.
20	4	Integer (little-endian)	Second counts from the beginning (17:00) to the RFQ time of a trading day. For example: At 21:00 in night trading hours: $3600 \times (21-17) = 14400$ At 9:00 in day trading hours: $3600 \times (24+9-17) = 57600$ At 9:00 in Monday's trading hours: $3600 \times (24+9-17) = 5760$ For the conversion between the integral time and reference time, see string2TimeID and timeID2String of ydUtil.h
24	4	Integer (little-endian)	RFQ ID No.
28	4		Non-public field
32	8	Integer (little-endian)	Full-accuracy RFQ ID No.

7.12.2.5 Quote notification message

Address offset	Length (bytes)	Field type	Description
0	16		Message header
20	4	Integer (little-endian)	Instrument SN. Corresponding to InstrumentRef of YDInstrument.
24	1	Integer	Bid offset flag
25	1	Integer	Bid hedge flag
26	1	Integer	Ask offset flag
27	1	Integer	Ask hedge flag
28	8	IEEE 754 Double-accuracy float type (little endian)	Bid price
36	8	IEEE 754 Double-accuracy float type (little endian)	Ask price
44	4	Integer (little-endian)	Bid volume
48	4	Integer (little-endian)	Ask volume
52	4	Integer (little-endian)	Customer's order reference No. i.e.
56	1	Integer	Connection selection method (0: YD_CS_Any, 1: YD_CS_Fixed, 2: YD_CS_Prefered)
57	1	Integer	Specified connection number
58	1	Integer	Actually used connection number
59	1	Integer	Quote flag YD_YQF_ResponseOfRFQ: For automatically filling in the RFQ number to indicate the response price. It is supported by SHFE, INE, DCE, GFEX and CZCE. Other exchanges do not provide RFQ numbers.
60	1	Unsigned integer	Order group ID
61	1	Integer	OrderRef control mode of order groups, refer to Order Group 0: Monotonic increase OrderRef numbers. The gap between two OrderRef numbers must be higher than or equal to 1 1: The OrderRef numbers should be strictly monotonically increased, and the gap between two OrderRef numbers must be 1

Address offset	Length (bytes)	Field type	Description
62	1	Integer	The quote attributes of the exchange, which shall be interpreted by the exchange to which the order belongs YD_EOA_DCE_GIS=1: DCE GIS(Good in Session)
63	1		Non-public field
64	4	Integer (little-endian)	Error No.
68	4	Integer (little-endian)	Exchange SN. Corresponding to ExchangeRef of YDExchange.
72	4	Integer (little-endian)	If ErrorNo is YD_ERROR_InvalidGroupOrderRef, it means that the maximum OrderRef has been received by current OMS; Otherwise, it means a quote number for quote cancellation. If the return value of the exchange is too long, it will be truncated.
76	4	Integer (little-endian)	The OrderSysID of the bid quote, which is 0 when selling a one-side quote. If the return value of the exchange is too long, it will be truncated.
80	4	Integer (little-endian)	The OrderSysID of the ask quote, which is 0 when buying a one-side quote. If the return value of the exchange is too long, it will be truncated.
84	4	Integer (little-endian)	When YDQuoteFlag is YD_YQF_ResponseOfRFQ, the order response RFQ number will be recorded, otherwise will be 0. If the return value of the exchange is too long, it will be truncated.
88	4	Integer (little-endian)	Connection number
92	8	Integer (little-endian)	Full-accuracy quote number for quote cancellation
100	8	Integer (little-endian)	The OrderSysID of a full-accuracy bid quote, which will be 0 when selling a one-side quote
108	8	Integer (little-endian)	The OrderSysID of a full-accuracy ask quote, which will be 0 when buying a one-side quote
116	8	Integer (little-endian)	When YDQuoteFlag is YD_YQF_ResponseOfRFQ, the full-accuracy order response RFQ number will be recorded, otherwise will be 0
124	4	Integer (little-endian)	Please refer to Remote User-defined field
128	12		Non-public field
140	8	Integer (little-endian)	QuoteSysID used by CFFEX quote modification.

7.12.2.6 Notification message for order or quote cancellation failure

Due to a large number of operations reusing order or quote cancellation notification messages, please only focus on the order cancellation notification when the order flag is 0, and those non-zero order cancellation notifications should be ignored.

Address offset	Length (bytes)	Field type	Description
0	16		Message header
16	4	Integer (little-endian)	Exchange order reference No. or quote NO.
20	1	Integer (little-endian)	Exchange SN. Corresponding to ExchangeRef of YDExchange.
21	1	Unsigned integer	Order group ID
22	1	Integer	The method of cancelling quote. (Only for SSE, SZSE can only cancel quotes by both sides simultaneously) YD_CQIT_Buy=1: cancel buy side YD_CQIT_Sell=2: cancel sell side YD_CQIT_Both=3: cancel both sides
23	1	Integer	Order flag YDOrderFlag, invalid in case of a notification message for quote cancellation failure
24	4	Integer (little-endian)	Error No.

Address offset	Length (bytes)	Field type	Description
28	4	Integer (little-endian)	Quote flag IsQuote. 1: This message relates to a notification for a quote cancellation failure 0: This message relates to a notification for an order cancellation failure
32	4	Integer (little-endian)	Investor's order reference No. OrderRef
36	4		Non-public field
40	8	Integer (little-endian)	Full-accuracy exchange order reference No. or quote No.

7.12.2.7 Heartbeat message

Heartbeat messages are used to maintain the TCP connection between the client and server. Both the client and the server will send heartbeat messages.

Address offset	Length(bytes)	Field type	Description
0	2	Integer (little-endian)	Fixed as 8
2	2		Non-public field
4	4	Integer (little-endian)	Fixed as 0

7.13 Fixed connection

A front server refers to the server of an exchange trading system that help the OMSs to connect and receive OMS submitted orders and returned notifications. Generally, more than one server is used. Considering the difference in number of connections for each front server as well as the imbalance in instantaneous operation volume, the performance of different front servers in delivering orders to the trading core may also vary. Therefore, during trading, Investors should evaluate the performance of each front and exclude the slowest front before placing orders on the remaining seats.

A connection refers to an account configured in the OMSs for connecting to a front server. YD is always trying its best to make each connection compliantly connect to different front server for the sake of investors' freest choice. When a broker arranges a YD OMS, it usually configure the same number of connections as that of exchange front servers. Generally, the connections of YD OMSs can cover all exchange's front servers, and therefore, selecting a front server is essentially means selecting a connection.

Due to the existence of fixed connection orders, the order volume on different connection of YD OMS is usually unbalanced, especially in the case of [reporting the results of the connection selection](#), which can easily trigger connection flow control. It is recommended for investors to [obtain connection information](#), understand the mechanism of [connection flow control](#) of YD, and based on their production needs, [place fixed connection orders](#) or [fixed connection cancel orders](#). Investors who have exclusive access to the counter can use [reporting the results of the connection selection](#) to share results of the connection selection with all accounts.

7.13.1 Obtaining connection information

YD OMSs support connecting to multiple exchanges for simultaneous trading. The connection information for each exchange is completely independent, and therefore, the following contents relate to a single exchange.

The total number of seats of an exchange can be found through YDExchange.ConnectionCount. The number of first connection is 0 and the number of last connection is ConnectionCount minus 1. YD supports a maximum of 64 seats. YD categorizes seats into public seats and dedicated seats.:

- Public seats: Seats that can be used by all investors. All public seats are listed through YDExchange.IsPublicConnectionID[64]. A connection can be checked for being a public one according to the seat number. If IsPublicConnectionID[i] is "True", it means the ith connection is a public one, otherwise, a dedicated one.
- Dedicated seats: Seats that can only be used by specified investors. Each dedicated seat can be assigned to multiple investors, and each investor can also use multiple dedicated seat. The dedicated seat information for each investor is listed through YDAccountExchangeInfo.IsDedicatedConnectionID[64]. If IsDedicatedConnectionID[i] is "True", the ith connection will be a dedicated one for this investor, otherwise it will not be a dedicated one.

Thus, the **available seats** of investors are the superposition of public and dedicated seats, while the **non-available seats** are dedicated seats of other investors.

Investors may want to compare the performances of two OMSs under the same front server. In version 1.386, investors can retrieve the connection information through 'YDExchange.ConnectionInfos' (accessible directly by array indexing, with the array size being 'YDExchange.ConnectionCount'). Additionally, YD API also informs about the connection information through 'notifyExchangeConnectionInfo' after 'notifyFinishedInit'. It provides the initial status of all connections during the first startup. In case there are changes in connection information due to disconnection or other reasons during trading hours, such updates will also be notified through the aforementioned callback.

```
1 | virtual void notifyExchangeConnectionInfo(const YDExchangeConnectionInfo *pExchangeConnectionInfo)
```

The field information of 'YDExchangeConnectionInfo' is as follows:

Field	Description
ExchangeRef	Exchange reference No. It can be obtained from YDExchange.

Field	Description
ConnectionStatus	Connection status YD_ECS_DISCONNECTED=0: Seat disconnected YD_ECS_CONNECTED=1: Seat connected
Info	Front IP address information. By default it will not be published, please contact your broker if you need this info.
InsertFlowControl	Check the maximum number of orders per interface within the time window during order placement or quote. The expression format is count/window. In the expression, the numerator 'count' represents the quantity limit, and the denominator 'window' represents the size of the time window for checking. If the numerator is less than or equal to 0, it indicates no upper limit. If the denominator is not specified or is less than or equal to 1, it represents 1000. For example, 50/1000 means that within a sliding time window of 1000 milliseconds, no more than 50 orders are allowed to be placed. Generally, the SHFE allows 50 orders within a 1000-millisecond time window, while other exchanges allow 100 orders within the same time window.
CancelFlowControl	Check the maximum number of order cancellations per interface within the time window during order cancellation or cancellation of quotes. The expression format is count/window. In the expression, the numerator 'count' represents the quantity limit, and the denominator 'window' represents the size of the time window for checking. If the numerator is less than or equal to 0, it indicates no upper limit. If the denominator is not specified or is less than or equal to 1, it represents 1000. This field can be left empty, indicating the same as the seat's order rate limit. For example, 50/1000 means that within a sliding time window of 1000 milliseconds, no more than 50 order cancellations are allowed. Generally, the SHFE allows 50 order cancellations within a 1000-millisecond time window, while other exchanges allow 100 order cancellations within the same time window.

7.13.2 Seat flow control

At present, the exchanges have two connection flow control modes for restricting the connection order transmission speed. Please note that the connection flow control does not make exceptions to investors, and the orders of all investors using the same connection are calculated in summary on the OMSs:

- Sliding Window Flow Control: For limiting the maximum counts of order submissions and cancellations per second for each connection or gateway. If exceeding the limit, the specific behaviors of each exchange APIs differ. Some exchange APIs will directly reject orders, and some will cache the orders until the next second for transmission. At present, all exchanges have this connection flow control mode and have announced their flow control thresholds. See the following table for the specific threshold information;
- In-transit Flow Control: For limiting the volume of orders that have been sent to exchanges but have not received the notification. If exceeding the limit, The specific behaviors of each exchange's API may vary. Some exchange APIs may directly reject orders, while others may queue orders until the in-transit order flow control is below a certain threshold. Currently, only a few exchanges have in-flight order flow control, including SSE and SZSE with their velocity control rights. However, other futures exchanges have not disclosed specific rules and thresholds for in-flight order flow control.

Exchange	Sliding window flow control threshold	The threshold for in-transit order flow control
CFEX	50	
SHFE, INE, DCE, GFEX, CZCE	100	
SSE, SZSE		velocity control rights

In order to avoid orders being cached and delayed due to sliding window flow control and in-transit order flow control, and to meet the principle of reporting errors as early as possible, YD Counter has implemented sliding window flow control and in-transit order flow control (starting from version 1.386.40.24, supporting in-transit order flow control for SHFE and CFEX, other exchanges are not supported. As the current order volume on SSE and SZSE is relatively small, sliding window flow control is temporarily used as a substitute. Please contact us if you actually need this function.) Intercepting orders before submission to APIs will result in API flow control of order submission and cancellation. For specific error reporting information, refer to [Designated Connection for Order Submission](#). The term "flow control" in this context refers to the collective use of sliding window flow control and in-transit order flow control.

7.13.3 Designated connection for order submission

Investors can set the ConnectionSelectionType and ConnectionID under the YDInputOrder and YDInputQuote to select order submission connections in different ways. The selection logic for preferred selection methods of various connections is shown below. Please note that disconnected, flow-control-limit reached, and non-selectable connections are excluded in connection selection.

- When ConnectionSelectionType is YD_CS_Any, a global polling will be achieved. The specific rules are:
 - All connections will be traversed one by one from the one subsequent to that using YD_CS_Any for order submission (the traversal will start from the beginning after reaching the last connection until reaching the starting connection of the traversal). If a connection is found not involved in the queue, it should be selected directly. Otherwise, check whether the order queue length of the connection is shorter than that of each previous connection. If being the shortest, just record it until the traversal is completed. Select the connection with the shortest recorded queue length. Finally, insert the order into the selected connection queue for submission.

- The poll does not make exceptions to investors, namely the orders of all investors are uniformly polled. Therefore, an investor submitting order under this mode may not be able to obtain consecutive connection numbers from through RealConnectionID.
- When ConnectionSelectionType is YD_CS_Fixed, a fixed connection order submission will be achieved. The specific rules are:
 - Directly select a connection using the ConnectionID and insert the order into the queue for submission.
- When ConnectionSelectionType is YD_CS_Prefered, a preferred connection is selected for order submission and the other seats are used if the preferred connection is busy. The specific rules are:
 - All connections will be traversed one by one from the one subsequent to that specified by the ConnectionID (the traversal will start from the beginning after reaching the last connection until reaching the starting connection of the traversal). If a connection is found not in the queue, it should be selected directly. Otherwise, check whether the order queue length of the connection is shorter than that of each previous connection. If being the shortest, just record it until the traversal is completed. Select the connection with the shortest recorded queue length. Finally, insert the order into the selected connection queue for submission.

In some cases, perhaps no any connection will be selected. YD lists various errors that may exist when searching for selected connections through different error codes:

- As long as one connection reaches the flow control limit, and all other connections are not dedicated, disconnected or have reached the upper limit of the connection queue, an error of "YD_ERROR_CanNotSendToExchangeForFlowControl=79 no available connection is provided and some connections reach the flow control limit" will be returned.
- If all connections are not available (dedicated, disconnected or reached the queue limit), an error of "YD_ERROR_CanNotSendToExchange=9 No available connection" will be returned.

When an order that has already been involved in the connection queue moves to the top and is ready for submission to an exchange, if exchange APIs such as orders, quotes, combinations and exercises reach the in-transit limits during submission or an error is sent back due to a disconnection, a message "YD_ERROR_ExchangeConnectionSendError=80 Exchange API transmission error" will be sent back. When the above error is caused and the OMS notification function for YD_AF_NotifyOrderAccept is enabled under YDAccountFlag, the notifyOrder notifications received will randomly be one of the following, so investors' strategy program should not rely on the notification counts, but focus on the notification status:

- A notification for OrderStatus=YD_OS_Rejected and ErrorNo=YD_ERROR_ExchangeConnectionSendError will be received
- Two notifications will be received, the first of which is that for OrderStatus=YD_OS_Accepted and ErrorNo=0, and the second, that for OrderStatus=Rejected and ErrorNo=YD_ERROR_ExchangeConnectionSendError

The selected seat (the seat actually used for order submission) can be obtained through the RealConnectionID in the notification of YDOrder or YDQuote.

7.13.4 Designated connection for order cancellation

Investors can set the ConnectionSelectionType and ConnectionID under the YDCancelOrder and YDCancelQuote to select order cancellation connections in different ways. Unlike order submission, the cancellation is dependent on the type of the OMS connections and therefore cannot be completely selected according to investors' settings.

- When the OMS connections are subject to full-management, the processing logic is the same as that of designated connections for order submission;
- When the OMS connections are subject to "first mangement others non-management seat", the original order submission connection and first management connection should be checked in order. If a connection is not involved in the queue, select it directly, otherwise, select the connection with a shorter queue length. If the queue length is the same, select the original order submission connection. Finally, insert the order into the selected connection queue for submission.
- When the OMS connections are subject to full-non-management, the order cancellation must be conducted from the original order submission connection. At this time, the original order submission connection should be selected through the OMS and the order should be inserted into the connection queue for submission

As with the designated connection order submission, the designated connection order cancellation will cause errors for exactly the same reasons. Refer to [Designated Connection Order Submission](#).

There is no way for APIs to obtain the selected connection in order cancellation. If this information is needed during troubleshooting, just contact the broker and request assistance in querying the inputFlow.txt of the OMSs.

7.13.5 Reporting the results of the seat optimization

In order to eliminate the slowest one, investors should compare the performance of each front server. The specific evaluation method should be implemented by investors based on the strategy. Generally, a single connection can be preferably selected through sending YD_CS_Fixed orders to all connections at a minimal gap: The connection with the smallest exchange order reference number OrderSysID in notifications for all connections is the best one in the test round. Through repeating the above steps and making a statistical analysis, the best connection recognized by investors can be selected, then this connection can be designated for order submission through YD_CS_Fixed and YD_CS_Prefered. Please note that the evaluation should not be too frequent, otherwise it may lead to order blocking on an OMS, disrupting the normal trading of other investors using the OMS, and even being considered as abnormal trading by exchanges.

If a broker has the ability to perform a connection selection and hopes its customers use its results, or perform a preferred selection at its exclusive OMS through one of the accounts and hopes that other accounts can also be available for submitting orders according to the preferred results, it can regularly transmit the preferably selected connection results for other accounts using the same OMS through the preferred connection reporting interface selectConnections. Only when the YD_AF_SelectConnection flag is set for YDAccount.AccountFlag can investors and administrators be allowed to use this interface.

```
1 | virtual bool selectConnections(const YDExchange *pExchange,unsigned long long connectionList);
```

The connectionList can be considered as a sequence of elements with a length of 4 bits, with each 4-bit representing one connection number, and the lowest bit representing the quickest connection. All exchanged connection numbers must be specified. For example, if you want to set the connection sequence (front server) from quick to slow to 2-3-1-4, the binary representation of the connectionList will be: 0100 0001 0000 0011 0010.

The validity period of the preferred seat result is MaxSelectConnectionGap (default value is 5 minutes), otherwise the submitted results will be invalid; If the result is expired after a timeout, it is recommended to submit a new report within the MaxSelectConnectionGap time limit to prevent the expiration and invalidation of the seat selection mechanism. At the same time, the reporting interval for seat selection results should not be too frequent, and the minimum gap shall not be shorter than MinSelectConnectionGap (60s by default). For the obtaining method of MaxSelectConnectionGap and MinSelectConnectionGap, refer to [System Parameters](#).

When the results of the seat optimization on the counter takes effect, all orders and cancellations using the YD_CS_Any method on that counter will be placed in seats according to the results of the seat optimization. Seat selection rule: according to the results of the seat optimization, iterate through all seats in order and prioritize seats with no queue. If all seats have a queue, then select the seat with the shortest queue length. During the selection process, unavailable seats will be skipped, such as seats under flow control, dedicated seats, disconnected seats, or seats that have reached the maximum queue limit.

At the counter where the results of the seat optimization takes effect, because orders tend to choose faster seats in priority order, it may lead to faster seats being more prone to flow control. When the counter is configured with all non-management seats, it can result in orders on seats under flow control being unable to be canceled. At the same time, in order to avoid the same front suppression effect, starting from version 1.280, YD OMS supports setting the number of rotations for preferred seats. This feature only takes effect when the preferred seat selection result at the counter takes effect.

The same front suppression effect refers to the situation in production where two sets of the same brand and configuration of counters send orders to the same front successively, and the counter that sends the order first always stays ahead of the counter that sends the order later. The reason for this kind of problem is that there is a logical relationship between the orders submitted to the same front by two sets of YD OMS, even if there is only an extremely small time difference (for example, 100ns), there will still be an absolute order relationship, which leads to the exchange's preference for processing the order that arrives first. However, in fact, there is no essential difference in performance between these two sets of YD OMS, and such a disparate result should not occur. At this point, if using the second or third ranked seat for order placement, it may increase the likelihood of taking the lead, as fundamentally seat prioritization only excludes the poorer seats, and the performance of the seats at the forefront may be essentially close. Therefore, avoiding using the same front for order placement may alleviate the issue of front suppression.

After setting the number of seat rotation to n, the first n preferred seats will be used in a loop for order placement each time. Assuming the preferred results are still 2-3-0-1-4-5, and the seat rotation number is set to 4, it means rotating the first 4 seats, which are 2, 3, 0, and 1. The seat selection rules are basically the same as when the seat rotation number is not set, the only difference is that after setting the seat rotation number, the "effective" seat selection results for each order placement will change according to the seat rotation:

- The seat preference order for the first order placement is 2-3-0-1-4-5.
- The seat preference order for the second order placement is 3-0-1-2-4-5.
- The seat preference order for the third order placement is 0-1-2-3-4-5.
- The seat preference order for the fourth order placement is 1-2-3-0-4-5.
- The seat preference order for the fifth order placement is 2-3-0-1-4-5.
- And so on.

7.14 Unknown timeout order processing

An unknown overtime order refers to that that without received notification from an exchange after being submitted by an OMS for a certain period of time. Since no any notification from the exchange is received, the status of the order will not change. Because the margin or position is frozen continuously, this order will remain in the OMS and cannot be released. Unknown timeout orders are usually caused by submissions performed when exchanges are disconnected or when the states of exchanges are switched, however, the probability is extremely low. YD provides a function for processing unknown overtime orders, however, when using it, the following instructions should be followed strictly. The orders can be processed only when confirmed as unknown ones for fear of losses.

YD OMSs can be used to continuously monitor YD_OS_Accepted orders (except those RFQ orders and instructions of DCE and GFEX with the YDOrderFlag set to YD_YOF_Mark). If no any notification regarding an order is received from the OMS within the MissingOrderGap (60s by default, which can be modified by brokers. For details, refer to [System Parameters](#)), the investor will be notified of an unknown timeout order via notifyMissingOrder:

```
1 | virtual void notifyMissingOrder(const YDMissingOrder *pMissingOrder)
```

The structure YDMissingOrder of unknown timeout orders is the same as YDOrder, however, when sending back a notification, YDMissingOrder.InsertTime is filled out with the time of receipt at the OMS, which is different from the exchange time YDOrder.InsertTime and should be noted.

After receiving a notification, investors should first confirm the order status at the primary OMS:

- If the primary OMS has the information about this order, it is obvious that this order has been received by the exchange and the notification sent by the exchange has been lost;
- If the primary OMS has no any information about this order, it means that the notification has not been sent to the exchange, or the exchange has not yet sent the notification. Considering that there have been notifications from exchanges with a timeout for more than one minute during production, waiting for a further period of time is suggested.

If it has been confirmed that the order is an unknown timeout one, just ask the broker for help. The broker's operators should also abide by the above rules and help the investor to cancel the unknown timeout order after a **prudent** judgment. The unknown timeout order can only be cancelled through the broker's administrator account rather than the investor himself/herself. After the broker cancels the unknown timeout order, the cancellation notification for the unknown timeout order will be sent back through notifyOrder. The returned YDOrder.ErrorNo will be YD_ERROR_InternalRejected, and YDOrder.OrderStatus will be YD_OS_Rejected.

If a notification from the exchange is received after cancelling the "unknown timeout order", YD OMS will handle it as an external order. Both the OrderRef and OrderLocalID of an external order are -1, thus losing the association with the original order. Of course, at this time, the investor can cancel the order. If then a trade notification is received, YD will update the positions and send the trade notification to the investor, so it is unnecessary to worry about the correctness of the positions and funds.

7.15 Performance tuning

The performance tuning aims to reduce overall look-through delay, namely from receiving the market data to sending an order request from the OMS to the exchange, steps such as market data receiving, strategy calculation, client order submission, switch forwarding, OMS risk control and submission to exchange are included. Each of the above steps is set on the critical path for trading. The low performance of one step can offset the high performance of other steps. The market leadership can only be ensured when the highest look-through performance of each step is kept.

Among them, client order submission and OMS risk control and submission to exchange are key steps in the trading services provided by YD. The look-through time points of these two steps are called API look-through performance and the OMS look-through performance, respectively.

7.15.1 API look-through performance

The API look-through performance refers to the time difference between starting to call insertOrder (or other order submission methods) and appearing the first byte for order submission on the optical fiber.

Since most OMSs' APIs will send back information immediately after being called, the only way to test the API look-through performance is to divide the "TX" ports of the up-bound OMSs of the investor strategy host and loopback to the sniff network card on the strategy host, and keep the clock on the sniff network card synchronized with the system clock. Record the timestamp of the system clock before submitting the order as the start timestamp. Use tcpdump or similar tools to timestamp the order packets received by the sniff network interface as the end timestamp. The difference between the two timestamps is the API's latency. The sniff network interface needs to support nanosecond-precision hardware timestamps and should be synchronized with the system time using the corresponding tools. If using the tcpdump tool, you can refer to the following command:

```
1 | tcpdump -i <if_name> -n --time-stamp-type=adapter_unsynced --time-stamp-precision=nano -w <pcap_file>
```

The return of the insert order function in the YD API indicates that the data packet has been sent over the optical fiber. Therefore, a simple method to obtain latency is to directly record system timestamps before and after the order submission and calculate the difference. It is recommended that investors measure the latency performance of the YD API using the aforementioned standard testing method for latency and compare it with the simple method. If the final conclusion shows that the results of the standard testing method and the simple testing method are essentially consistent, then the simple method can be used for measuring the latency performance of the YD API in subsequent evaluations. Please note that before using the simple method to measure the latency performance of other APIs or raw protocol, it is essential to calibrate the feasibility of the simple method using the standard testing method.

YD provides a quick and high-accuracy timestamp function with a low performance overhead and no excessive impact on the critical path for order submission can be caused. When testing the API look-through performance, timestamps can be obtained by using this function before and after using the order submission function, and the nanosecond count for API look-through delay can be obtained by calculating the difference value between the timestamps.

```
1 // Returns nanoseconds elapsed since current process starts up
2 YD_API_EXPORT unsigned long long getYDNanoTimestamp();
```

After a long period of iterative optimization of APIs, YD can realize its API look-through performance of less than 250ns when using Solarflare X2522 or Exanic X10/X25 network cards and its acceleration software. If the actual look-through performance tested by investors does not reach this indicator, the network card and startup method should be checked first for meeting YD's requirements and, if possible, reaching the official performance standard. If not, contact YD through a broker.

Investors who have used raw protocols on other systems tend to use raw protocols. Considering investors' trading habits, YD also provides a method of [Raw Protocol](#) for order submission and cancellation. However, the look-through performance of YD APIs has reached the level of ordinary FPGAs. Therefore, if investors try to implement their own raw protocol-based order submission, they can compare its look-through performance with that of YD APIs after implementing the raw protocol-based order submission, and the quicker method can be selected for production of orders.

7.15.2 OMS look-through performance

The OMS look-through performance refers to the time difference when the first byte for order submission enters and leaves an OMS. This indicator can be used for objectively and impartially demonstrate the OMS performance and is the most commonly used one to judge the OMS performance.

The measurement is generally performed with an optical splitter or a port mirror in the industry. The optical splitter is characterized by high accuracy and low cost and is suitable for temporarily arranged measurement though being hard to adjust; The port mirror is easy to adjust, but only high-end models such as Arista 7130 can simultaneously and functionally help to achieve minimal performance impact and maximum measurement accuracy. Mid-to-low end models are not suitable for OMS look-through performance measurement. YD provides brokers with special tools to measure and analyze the up-bound look-through performance of each order submission, quote submission, and order cancellation through OMSs. If you want to know your order look-through data, just ask the broker for help.

In most cases, the look-through performance should not be a concern for investors. YD has conducted comprehensive tests for all versions to ensure that YD's expected performance indicators can be achieved during production. If investors suspect that the look-through performance deteriorates, resulting in a decline in earnings, they can contact a broker to check the look-through delay for meeting the performance standard. The standard performance will be provided to the broker upon completion of each server test.

Under some special trading behaviors, the look-through performance of OMSs will deteriorate. At present, it is known that sending orders through OMSs too quick or too slow may significantly cause a prolongation of the look-through delay, which should be handled separately. If the look-through performance deterioration is not caused by these trading behaviors, please contact YD's customer service department for troubleshooting.

7.15.2.1 Order blocking

Order blocking can be caused when multiple orders are sent from the same designated connection to an OMS at a gap shorter than the look-through delay. Due to the TCP connection of exchanges, the orders need to be queued for submission. On one hand, they will arrive at the same time, while on the other hand, they are queued for submission, thus, in terms of look-through performance, in addition to the normal look-through delay of the first order, the look-through delay of each subsequent order will be prolonged by a relatively fixed period of time compared to the previous one, resulting in order blocking.

Order blocking may be caused by one or more of the following circumstances.

- When orders are queued through "Fix" and sent by the same connection ("Any" and "Preferred" orders do not queue), it is liable to be caused by that investors always consistently use their the fixed connections, if determined, to submit orders.
- Some investors always submit orders too frequently, resulting in a prolongation of the busy time of the connection and a higher possibility of the subsequent Fix instructions for queuing. For example, a customer is sending preferred orders frequently through a connection and has not avoided the arrival time of the market data segments.
- The arrived market data segments or trading nodes have a starting gun effect, and most investors hurry on submitting orders at the same time.
- Some investors used a wrong warm-up method, resulting in a huge volume of orders at an OMS. For example, sending and cancelling orders at the OMS at a very high frequency in order to achieve a warming effect.
- Sending orders to different connections at the same time can also cause a sequential prolongation of delay periods, however, the prolongation is much less than that for one connection. It is common that when sending preferred orders to all connections, the look-through delay of preferred orders for subsequent connections will prolong slightly.

Order blocking is essentially caused by normal trading. In addition to further reducing the look-through delay. There is no better way for YD to alleviate such problem.. For brokers, investors should be dissuaded in time from taking such wrong warm-up measures. If investors submit their preferred orders too frequently through connections or fail to adjust the time periods suggested according to the market data segments for sending preferred orders, they, when causing a server confliction, should become separated and be arranged on different OMSs.

7.15.2.2 Slow order submission

When the order submission speed on an OMS is too slow, the cache of the OMS server will cool down, and the look-through delay of the order will significantly be prolonged when new orders arrive. In most cases, the OMS will not encounter this problem since the orders of all investors trading through the OMS will warm up the cache. This problem can only be caused when all investors on the OMS submit orders at a very low speed (such as at a frequency of 1 order / min for the entire OMS). When this problem is confirmed by the investors and brokers, it is suggested that investors send warm-up orders in advance to avoid this problem. The sending method and timing of warm-up orders are described below.

The best warm-up effect can be achieved when the warm-up orders and formal orders are sent through the same execution path.

- The warm-up effect is best when the execution path is the same for warm-up orders and formal orders. For example, when formal orders are submitted, the warm-up effect achieved through order submission is better than that through order cancellation.
- The warm-up effect of warm-up orders is only suitable for connections specified for the warm-up orders, while, for other connections, which is limited.
- The execution path of warm-up orders that fail to reach exchanges is shorter than that of formal orders, and the warm-up effect cannot achieve the best.
- When warm-up orders and formal orders are sent from the same connection under different instruments, the execution path is the same, which can help to achieve the warm-up effect.

The gap between warm-up order and formal order submission can also affect the warm-up effect. Tests have shown that the best warm-up effect can be achieved when the two are sent at a gap of 5s. Shorter gaps cannot help to achieve a better effect but can cause freezing of the order margin for a longer time before receiving notifications, which can also cause a higher pressure or even order blocking on the OMS. Warm-up orders should not be submitted when receiving market data, otherwise normal orders may be blocked. Since warm-up orders are not intensively sent, it is relatively easy to solve this problem.

The warm-up effects for different investors do not affect each other, and therefore do not worry that other customers' warm-up behaviors will affect your own warm-up effect. In fact, they also have a certain warm-up effect on your formal order submission.

According to the sending conditions for the above-mentioned warm-up effect, namely the connection for warm-up order submission should be the same as that for formal order submission, the time for receiving market data should be calculated in order to send warm-up orders in advance and ensure that they have been sent to the exchange. This precise warm-up method is always suggested. For investors, the regularly sent preferred orders through their own connections are also feasible warm-up orders, however, the effect is not as good as that achieved through precise warming up.

8 Market data

YD OMSs can be used to forward market data segments from the front server of an exchange to investors. The market data provided by YD is mainly used for calculating refreshed position profits/losses and margins, and in the event of a disruption in receiving multicast market data, providing backup market data. Compared to the sent multicast market of the exchange, it is slower and may case miss of trading opportunities, and therefore trading directly based on the forwarded market data is not recommended.

For investors setting RecalcMode to auto or subscribeOnly, the simplest way to obtain the **market data relating to a position instrument** is to directly query the market data relating to the corresponding instrument through the market data pointer `m_pMarketData`. Refer to [Fund Refresh Mechanism](#) for details.

For investors who want to obtain market data relating to non-position instruments, or set RecalcMode to "off", or be notified when the market data is updated, they can subscribe to them actively. Before actively subscribing to market data, the API parameters should be configured according to the following method (When RecalcMode is set to auto or subscribeOnly, the API will overwrite the following parameters, so do not worry about the setting of API market data parameters). Since no UDP market data is not suggested to be used by YD OMSs, `ReceiveUDPMarketData` must always be kept at "no", otherwise no market data will be received.

```
1 | ConnectTCPMarketData=yes
2 | ReceiveUDPMarketData=no
```

When `ConnectTCPMarketData` is set to "yes", a separate market data thread will be created after API is started. In order to prevent the market data thread from affecting the operation of other threads of the strategy system, the market data thread can be bound to a CPU through configuring API parameters.

```
1 | # Affinity CPU ID for thread to receive TCP market data, -1 indicate no need to set CPU affinity
2 | TCPMarketDataCPUID=-1
```

If the market data is expected to be notified to the strategy program as soon as possible, the working mode of the market data receiving thread can be set:

- When set to -1, the system will be in a busy query state, and the market notification performance is the best, however, the thread will occupy the operation CPU core;
- Investors who do not want the market thread to occupy too much CPU space and do not care about the market data notification performance can set a timeout for the "select" function. The operation under this mode is slower than that under busy query, but the CPU space can be greatly saved. It is suggested that investors set a timeout at the GUI client to reduce unnecessary performance overhead.

```
1 | # Timeout of select() when receiving TCP market data, in millisec. -1 indicates running without select()
2 | TCPMarketDataTimeout=10
```

For a detailed description of the above parameters, refer to [Market Data Configuration](#).

APIs provide the following function for subscribing to and unsubscribing to instrument-based market data. Any actively subscribed instrument-based market data can be queried directly through the market data pointer `m_pMarketData` mentioned in the above instrument pointer.

```
1 | virtual bool subscribe(const YDInstrument *pInstrument);
2 | virtual bool unsubscribe(const YDInstrument *pInstrument);
```

After the instrument-based market data is updated, investors will be notified through the following callback function.

```
1 | virtual void notifyMarketData(const YDMarketData *pMarketData) {}
```

Parameter	Field	Description
YDMarketData	InstrumentRef	Instrument reference No.
	TradingDay	Current trading day
	PreSettlementPrice	Pre-settlement price
	PreClosePrice	Pre-close price
	PreOpenInterest	Unilateral Pre-position volume
	UpperLimitPrice	Upper limit price
	LowerLimitPrice	Lower limit price
	LastPrice	Last price
	BidPrice	Bid price
	AskPrice	Ask price
	BidVolume	Bid volume
	AskVolume	Ask volume
	Turnover	Nominal transaction amount

Parameter	Field	Description
	OpenInterest	Unilateral position volume
	Volume	Daily trading volume.
	TimeStamp	<p>Millisecond counts from the beginning (17:00) to the order submission time of a trading day. For example:</p> <p>500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$</p> <p>500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$</p> <p>500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$</p> <p>For the conversion between YD's timestamp time and reference time, refer to string2TimeStamp and timeStamp2String of ydUtil.h</p>
	AveragePrice	Average trade price
	DynamicBasePrice	Dynamic benchmark price. Please refer to Risk control of price deviation for more details.
	LastTradeTimeStamp	<p>Millisecond counts from the beginning (17:00) to the most recent transaction time of a trading day, only apply to SSE and SZSE. For example:</p> <p>500 ms past 21:00 in night trading hours: $3600 \times (21-17) \times 1000 + 500 = 14400500$</p> <p>500 ms past 9:00 in day trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 57600500$</p> <p>500 ms past 9:00 in Monday's trading hours: $3600 \times (24+9-17) \times 1000 + 500 = 5760500$</p> <p>For the conversion between YD's timestamp time and reference time, refer to string2TimeStamp and timeStamp2String of ydUtil.h</p>
	m_plInstrument	Order instrument pointer

9 Risk control

The risk control rules of YD can be divided into two categories: before-trade risk control and after-trade risk control.

The before-trade risk control intercepts non-compliant orders and prevents them from reaching the exchange. The before-trade risk control is typically regulatory requirements, and violating them may result in regulatory penalties such as trading restrictions, including self-trades, order cancellation limits and position limits. The before-trade risk control is performed by the OMS before sending instructions to the exchange. If they fail to pass, error notifications will be sent back via the API. Investors using `ydExtendedApi` can utilize the `checkAndInsert` series methods to perform local API risk control checks according to the `checkAndInsert` series methods. If the check results are acceptable, orders can be submitted normally to the OMS. If the check results are unacceptable, the function will return "False". Investors can check `ErrorNo` within "YDInputOrder" or "YDInputQuote". It's important to note that even if the API performs local risk checks, the OMS still conducts its own risk checks during the order processing.

The after-trade risk control is a concern for brokers or investors themselves. Violation of these risk control rules will increase the risk for both brokers and investors. Because the boundaries of risk control thresholds are relatively lenient, even if there is a slight breach, it will not result in significant changes, such as information volume and other risk control rules. However, if there is a business need to ensure the effectiveness of thresholds under extreme conditions (for example, when investors place orders at very short intervals, and by the time risk control receives the report, it has exceeded the threshold significantly), a stronger risk control threshold can be set to avoid exceeding the threshold too much under extreme conditions. For example, if the business requires a position limit threshold of 100 lots, it can be set to 90, leaving a margin of 10 lots for exceeding. The typical after-trade risk control measures include automatically adjusting the investor's trading permissions or sending risk control alerts.

9.1 Before-trade risk control

9.1.1 Before-trade risk control of open position volume

The risk control of open position volume can be divided into two dimensions: product level and instrument level. Each dimension can independently control the total open position volume, as well as buy-to-open volume and sell-to-open volume. The following example will be explained using the open position volume. The processing methods for risk control of buy-to-open volume and sell-to-open volume are similar.

The risk control rules at the product level involve the aggregation of the open position volume for all instruments belonging to that product. Once the risk control threshold at the product level is triggered, no further open instructions can be sent for all instruments within that product.

The risk control rules at the instrument level aim to perform a risk control for the total open position volume under an instrument. Once the risk control threshold of this instrument level is reached, this instrument will be unavailable for sending open position instructions.

This risk control will not be started after the open position volume reaches the threshold, otherwise, once a new open position order is submitted to an exchange, a value being higher than the risk control threshold will be caused. Therefore, what YD actually controls is a possible open position volume. When an open position order is sent, the possible open position volume of YD will increase, which will not change in case of "pending order". When the final order status is reached, the untraded open position volume can be deducted from the possible open position volume. Therefore, if there are a large number of pending open position orders, the subsequent open position orders will also be rejected due to this risk control.

For each risk control parameter at the product level, refer to `OpenLimit` and `DirectionOpenLimit[2]` of `YDAccountProductInfo.TradingConstraints[HedgeFlag]`. For each risk control parameter at the instrument level, refer to `OpenLimit` and `DirectionOpenLimit[2]` of `YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]`. For field meanings, refer to [Risk Control Parameters](#).

9.1.2 Before-trade risk control of trade volume

The risk control of trade volume can be divided into two levels: product level and instrument level.

The risk control rules at the product level aim to perform a risk control for the total bid/ask offset volume under all instruments relating to a product. Once the risk control threshold of this product level is reached, all instruments relating to this product will be unavailable for sending any instruction.

The risk control rules at the instrument level aim to perform a risk control for the total bid/ask offset volume under an instrument. Once the risk control threshold of this instrument level is reached, this instrument will be unavailable for sending any instruction.

This risk control will not be started after the trade volume reaches the threshold, otherwise, once a new order is submitted to an exchange, a value being higher than the risk control threshold will be caused. Therefore, what YD actually controls is a possible trade volume. When an order is sent, the possible trade volume of YD will increase, which will not change in case of "pending order". When the final order status is reached, the untraded order volume can be deducted from the possible trade volume. Therefore, if there are a large number of pending orders, the subsequent orders will also be rejected due to this risk control.

For each risk control parameter at the product level, refer to `TradeVolumeLimit` of `YDAccountProductInfo.TradingConstraints[HedgeFlag]`. For each risk control parameter at the instrument level, refer to `TradeVolumeLimit` of `YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]`. For field meanings, refer to [Risk Control Parameters](#).

9.1.3 Before-trade risk control of position volume

The risk control of position volume can be divided into two levels: product level and instrument level. Each level aims to control the position volume, long position volume and short position volume, respectively. Taking the position volume as an example, the processing methods for risk control of long position volume and short position volume are similar.

The risk control rules at the product level aim to perform a risk control for the total long/short position volume under all instruments relating to a product. Once the risk control threshold of this product level is reached, all instruments relating to this product will be unavailable for sending open position instructions.

The risk control rules at the instrument level aim to perform a risk control for the total long/short position volume under an instrument. Once the risk control threshold of this instrument level is reached, this instrument will be unavailable for sending open position instructions.

This risk control will not be started after the position volume reaches the threshold, otherwise, once a new open position order is submitted to an exchange, a value being higher than the risk control threshold will be caused. Therefore, what YD actually controls is a possible position volume. When an open position order is sent, the possible position volume of YD will increase, which will not change in case of "pending order". When the final order status is reached, the untraded order volume can be deducted from the possible position volume. Therefore, if there are a large number of pending open position orders, the subsequent open position orders will also be rejected due to this risk control.

For each risk control parameter at the product level, refer to `PositionLimit` and `DirectionPositionLimit[2]` of `YDAccountProductInfo.TradingConstraints[HedgeFlag]`. For each risk control parameter at the instrument level, refer to `PositionLimit` and `DirectionPositionLimit[2]` of `YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]`. For field meanings, refer to [Risk Control Parameters](#).

9.1.4 Risk control of order cancellation counts

The risk control of order cancellation counts can be divided into two levels: product level and instrument level.

The risk control rules at the product level aim to perform a risk control for the specific order cancellation counts under all instruments relating to a product. Once the risk control threshold of this product level is reached, all instruments relating to this product will be unavailable for sending any instruction.

The risk control rules at the instrument level aim to perform a risk control for the specific order cancellation counts under an instrument. Once the risk control threshold of this instrument level is reached, this instrument will be unavailable for sending any instruction.

The definition of the above-mentioned specific order is as follows:

- FAK and FOK orders of CFFEX of treasury bond futures
- Limit orders of CFFEX except treasury bond future
- Limit orders of SHFE, INE, DCE, CZCE and GFEX

This risk control will not be started after the price-limited order cancellation counts reach the threshold, otherwise, once a new order is submitted to an exchange, a value being higher than the risk control threshold will be caused. Therefore, what YD actually controls is a possible order cancellation counts. When a price-limited order is sent, the possible order cancellation counts of YD will increase, which will not change in case of "pending order" or active order cancellation. When all trades are made, one order cancellation count will be deducted. Therefore, if there are a large number of pending price-limited orders, the subsequent price-limited orders will also be rejected due to this risk control.

For each risk control parameter at the product level, refer to `CancelLimit` of `YDAccountProductInfo.TradingConstraints[HedgeFlag]`. For each risk control parameter at the instrument level, refer to `CancelLimit` of `YDAccountInstrumentInfo.TradingConstraints[HedgeFlag]`. For field meanings, refer to [Risk Control Parameters](#).

9.1.5 Risk control of single order volume

Although the maximum order volumes based on each instrument are control by exchanges, considering that the strategy-specified volumes of some investors are still too large, they should be controlled under the help of OMSs. Therefore, YD supports limiting the maximum volume of orders submitted each time.

Among common risk control parameters, those order volume risk control parameters support setting at three levels i.e. exchange, product, and instrument, respectively:

Level	Field	Description
Exchange	<code>GeneralRiskParamType</code>	Fixed to <code>YD_GRPT_ExchangeMaxOrderVolume</code>
	<code>AccountRef</code>	-1 indicates that it is valid for all investors, otherwise it indicates the user's <code>AccountRef</code> value, which is the account itself for investor account login
	<code>ExtendedRef</code>	Corresponding to <code>YDExchange.ExchangeRef</code> , the only identified exchange
	<code>IntValue1</code>	Upper limit of order volume
Product	<code>GeneralRiskParamType</code>	Fixed to <code>YD_GRPT_ProductMaxOrderVolume</code>
	<code>AccountRef</code>	-1 indicates that it is valid for all investors, otherwise it indicates the user's <code>AccountRef</code> value, which is the account itself for investor account login
	<code>ExtendedRef</code>	Corresponding to <code>YDProduct.ProductRef</code> , the only identified product
	<code>IntValue1</code>	Upper limit of order volume
Instrument	<code>GeneralRiskParamType</code>	Fixed to <code>YD_GRPT_InstrumentMaxOrderVolume</code>
	<code>AccountRef</code>	-1 indicates that it is valid for all investors, otherwise it indicates the user's <code>AccountRef</code> value, which is the account itself for investor account login
	<code>ExtendedRef</code>	Corresponding to <code>YDInstrument.InstrumentRef</code> , the only identified instrument
	<code>IntValue1</code>	Upper limit of order volume

Since the AccountRef under each level involves all accounts or designated accounts, six levels can be formed after pair combination. Their priorities from high to low (a high priority parameter configuration overrides a low priority one) are:

- Instrument level and for designated accounts
- Product level and for designated accounts
- Exchange level and for designated accounts
- Instrument level and for all investors
- Product level and for all investors
- Exchange level and for all investors

9.1.6 Risk control of price deviation

When the difference between the order price and the dynamic base price exceeds the threshold, or when the difference between the order price and the latest price exceeds the threshold, the order will be rejected by the OMS. The dynamic base price differs in different exchanges in terms of value rules:

- For SSE and SZSE, the dynamic base price refers to a trade price generated during the latest call auction under an instrument. If no any trade price is generated during the opening call auction, the previous settlement price should be used as the latest reference price; If no any trade price is generated during the trading call auction, the last trade price generated this call auction should be used as the latest reference price;
- For futures exchanges, if the trade volume in the market data is not 0, the dynamic base price should be the latest one, otherwise, the pre settlement price should be used.

YD directly provides the dynamic base price YDMarketData.DynamicBasePrice in the market data.

This risk control rules involving a large number of parameters can be triggered when meeting any of the following conditions:

- Order price \geq dynamic base price \times (1 + upper deviation ratio limit of dynamic base price)
- Order price \leq dynamic base Price \times (1 – lower deviation ratio limit of dynamic base price)
- Order price \geq dynamic base price \times (1 + upper limit of dynamic base price deviating from tick count)
- Order price \leq dynamic base price \times (1 + lower limit of dynamic base price deviating from tick count)
- Order price \geq latest price \times (1 + upper deviation ratio limit of latest price)
- Order price \leq latest price \times (1 – lower deviation ratio limit of latest price)
- Order price \geq latest price \times (1 + upper limit of latest price deviating from tick count)
- Order price \leq latest price \times (1 – lower limit of latest price deviating from tick count)

Due to field constraints, YD needs to synthesize multiple general risk control rules configured for the same product to represent one of its own risk control rules. Since this risk control rule only allows configuration at the global and product level. Setting ExtendedRef to empty indicates that it applies to all products, while setting it to a product name indicates that it only applies to that specific product. and no account is allowed to be designated, the following AccountRef and ExtendedRef are omitted. Please note that it is possible to receive multiple sets of parameters with ExtendedRef being empty and specific products set. Please pay attention to distinguishing between them. The remaining parameters for price deviation are shown in the table below:

Level	Field	Description
Upper deviation ratio limit of dynamic base price	GeneralRiskParamType	Fixed to YD_GRPT_DynamicPriceLimitUpperRatio
	FloatValue	Ratio threshold
Lower deviation ratio limit of dynamic base price	GeneralRiskParamType	Fixed to YD_GRPT_DynamicPriceLimitLowerRatio
	FloatValue	Ratio threshold
Upper limit of dynamic base price deviating from tick count	GeneralRiskParamType	Fixed to YD_GRPT_DynamicPriceLimitUpperTickCount
	IntValue1	tick count
Lower limit of dynamic base price deviating from tick count	GeneralRiskParamType	Fixed to YD_GRPT_DynamicPriceLimitLowerTickCount
	IntValue1	tick count
Upper deviation ratio limit of latest price	GeneralRiskParamType	Fixed to YD_GRPT_DynamicLastPriceLimitUpperRatio
	FloatValue	Ratio threshold
Lower deviation ratio limit of latest price	GeneralRiskParamType	Fixed to YD_GRPT_DynamicLastPriceLimitLowerRatio
	FloatValue	Ratio threshold
Upper limit of latest price deviating from tick count	GeneralRiskParamType	Fixed to YD_GRPT_DynamicLastPriceLimitUpperTickCount
	IntValue1	tick count
Lower limit of latest price deviating from tick count	GeneralRiskParamType	Fixed to YD_GRPT_DynamicLastPriceLimitLowerTickCount
	IntValue1	tick count

Due to 'ExtendedRef' being applicable to all products and specific products, there are two levels in total. Their priority, from high to low (with higher priority overriding lower priority parameter configurations), is as follows:

- Specific products
- All products

9.1.7 Option calling amount

This risk control rule can be used for controlling the option calling amount (total long position cost of options). The long position cost of options is the sum of long position costs of options at corresponding levels (all costs under an account or those costs under an account in an exchange). The long position cost of a single option is the sum of its position details, and the cost of a single position detail is its trade price times the trade volume.

This risk control rule supports the following sub-rules:

- Aggregate calling amount control, which involves controlling the total calling amount by summing up the calling amounts across all exchanges. If the aggregate calling amount exceeds the threshold, order placement on all exchanges will be restricted.
- Single exchange calling amount control, which involves aggregating the calling amounts of individual exchanges. Only when the calling amount of a specific exchange exceeds the threshold will order placement on that exchange be restricted.

The parameters for aggregate calling amount control are shown in the table below:

Level	Field	Description
Global	GeneralRiskParamType	Fixed to YD_GRPT_OptionLongPositionCost
	AccountRef	-1 indicates that it is valid for all investors, otherwise it indicates the user's AccountRef value, which is the account itself for investor account login
	FloatValue	Position cost limit threshold

If ydExtendedApi is used, the position cost threshold of an investor at the global level can be found through YDExtendedAccount.OptionLongPositionCostLimit. The current summarized position cost value of the investor at the global level can also be found through YDExtendedAccount.OptionLongPositionCost.

Since AccountRef involves all accounts or designated accounts, two levels can be covered. Their priorities from high to low (a high priority parameter configuration overrides a low priority one) are:

- For designated accounts
- For all investors

The parameters for single exchange calling amount control are shown in the table below:

Level	Field	Description
Exchange level	GeneralRiskParamType	Fixed to YD_GRPT_ExchangeOptionLongPositionCost
	AccountRef	-1 indicates that it is valid for all investors, otherwise it indicates the user's AccountRef value, which is the account itself for investor account login
	ExtendedRef	It can be set to empty to apply to all exchanges or set to a specific exchange's 'ExchangeRef' to apply only to that particular exchange. 'ExchangeRef' can be obtained from 'YDExchange'.
	FloatValue	Position cost limit threshold

Due to AccountRef being applicable to all accounts or specific accounts, and ExtendedRef being applicable to all exchanges or specific exchanges, there are a total of four levels. Their priorities from high to low (a high priority parameter configuration overrides a low priority one) are:

- For designated accounts, for specific exchanges
- For designated accounts, for all exchanges
- For all investors, for specific exchanges
- For all investors, for all exchanges

9.1.8 Before-trade risk control of message count

To help investors control their maximum message count in advance and to make up for deficiencies in the force of [Message Count Risk Control](#), YD implements a preemptive risk control based on the maximum message count. If an order will exceed the maximum message count limit, it will be rejected by the OMS. Refer to [Message Count Commission](#) for the method of calculating message count.

The reason that risk control based on the OTR message count upper limit cannot be provided is that after reaching a higher OTR level, investors can submit more successful orders to reduce OTR. If orders are forbidden after reaching the OTR, this will cut off the possibility of investors making adjustments themselves. Since the message count is a monotonically increasing number, its maximum value can be limited.

Only when an investor is charged a message count commission on a contract and YDAccountInstrumentInfo.MaxMessage is greater than 0, will the maximum message count of the investor on the contract be controlled. Similar to other preemptive risk controls, this risk control does not start only after the message count of the order reaches the threshold. Otherwise, once a new order is submitted to the exchange, it will exceed the risk control threshold. Therefore, what YD actually controls is the possible message

count. When an order is submitted, YD will increase the possible message count. If the order is cancelled, the possible message count does not change. If the order is completely filled, the possible message count is reduced. Therefore, if there are currently a large number of pending orders, subsequent orders may be rejected due to this risk control.

Unlike other preemptive risk controls, this risk control setting is directly converted from CTP daily initial data. As long as the investor's maximum message count risk control is set in CTP, the risk control rule will be automatically set in YD. Similar to CTP, YD also supports mid-session adjustments to the maximum message count. Please contact the administrator if you need to make adjustments during the session.

If the broker adjusts the maximum information volume risk control parameters during trading hours, the API will notify the investor through the following callback function:

```
1 | virtual void notifyUpdateMessageCommissionConfig(const YDUpdateMessageCommissionConfig
    *pUpdateMessageCommissionConfig)
```

The field information for 'YDUpdateMessageCommissionConfig' is as follows:

Field	Description
AccountRef	Account reference No. which can be obtained from YDAccount.
ExchangeRef	Exchange reference No. which can be obtained from YDExchange.
ProductRef	Product reference No. which can be obtained from YDProduct.
InstrumentRef	Instrument reference No. which can be obtained from YDInstrumentRef.
MaxMessage	The updated maximum information volume threshold.

9.2 After-trade risk control

9.2.1 After-trade risk control of open position volume

When the total buy-to-open volume of the option series is higher than or equal to the maximum buy-to-open volume, or the sell-to-open volume is higher than or equal to the maximum sell-to-open volume, the "Only position closing allowed" right can be set for all instruments relating to the option series under the account.

The YD after-trade risk control system supports the following six types of risk control:

- Single instrument open position volume limit: Restricts the open position volume of a particular instrument.
- Product instrument open position volume limit: For each instrument under a specified product, restricts the open position volume of individual instruments.
- Exchange instrument open position volume limit: For each instrument under a specified exchange, restricts the open position volume of individual instruments.
- Option series aggregate open position volume limit: Restricts the total open position volume of all instruments under a specified option series.
- Product aggregate open position volume limit: Restricts the total open position volume of all instruments under a specified product.
- Exchange product aggregate open position volume limit: For each product under a specified exchange, restricts the total open position volume of all instruments within that product.

The parameters of this risk control rule are as follows:

Level	Field	Description
Single instrument open position volume limit	GeneralRiskParamType	Fixed to 1019
	AccountID	Funds account
	ExtendedID	Instrument
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume
Product instrument open position volume limit	GeneralRiskParamType	Fixed to 1020
	AccountID	Funds account
	ExtendedID	Product
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume
Exchange instrument open position volume limit	GeneralRiskParamType	Fixed to 1021
	AccountID	Funds account
	ExtendedID	Exchange

Level	Field	Description
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume
Option series aggregate open position volume limit	GeneralRiskParamType	Fixed to 1007
	AccountID	Funds account
	ExtendedID	It can be in the following two formats: Option product code: Option expiration year: Option expiration month Option product code: Underlying instrument code
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume
Product aggregate open position volume limit	GeneralRiskParamType	Fixed to 1022
	AccountID	Funds account
	ExtendedID	Product
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume
Exchange product aggregate open position volume limit	GeneralRiskParamType	Fixed to 1023
	AccountID	Funds account
	ExtendedID	Exchange
	IntValue1	Maximum buy-to-open volume
	IntValue2	Maximum sell-to-open volume

There is no overlapping relationship between different types of risk control rules in this rule. If risk control rules are set at multiple levels simultaneously, these rules will take effect concurrently.

At present, the external risk control system is responsible for the execution of this risk control rule. Therefore, investors cannot obtain specific risk control parameter values or determine whether the risk control rules have been triggered through the API.

9.2.2 After-trade risk control of trade volume

When the total trade volume of this option series is greater than or equal to the maximum trade volume, set 'only close' permission for all instruments of this option series in the account.

The YD after-trade risk control system supports the following six types of risk control:

- Single instrument trade volume limit: Restricts the trade volume of a specific instrument.
- Product instrument trade volume limit: For each instrument under a specified product, restricts the trade volume of individual instruments.
- Exchange instrument trade volume limit: For each instrument under a specified exchange, restricts the trade volume of individual instruments.
- Option series aggregate trade volume limit: Restricts the total trade volume of all instruments under a specified option series.
- Product aggregate trade volume limit: Restricts the total trade volume of all instruments under a specified product.
- Exchange product aggregate trade volume limit: For each product under a specified exchange, restricts the total trade volume of all instruments within that product.

The parameters of this risk control rule are as follows:

Level	Field	Description
Single instrument trade volume limit	GeneralRiskParamType	Fixed to 1014
	AccountID	Funds account
	ExtendedID	Instrument
	IntValue1	Maximum trade volume
Product instrument trade volume limit	GeneralRiskParamType	Fixed to 1015
	AccountID	Funds account
	ExtendedID	Product
	IntValue1	Maximum trade volume

Level	Field	Description
Exchange instrument trade volume limit	GeneralRiskParamType	Fixed to 1016
	AccountID	Funds account
	ExtendedID	Exchange
	IntValue1	Maximum trade volume
Option series aggregate trade volume limit	GeneralRiskParamType	Fixed to 1006
	AccountID	Funds account
	ExtendedID	It can be in the following two formats: Option product code: Option expiration year: Option expiration month Option product code: Underlying instrument
	IntValue1	Maximum trade volume
Product aggregate trade volume limit	GeneralRiskParamType	Fixed to 1017
	AccountID	Funds account
	ExtendedID	Product
	IntValue1	Maximum trade volume
Exchange product aggregate trade volume limit	GeneralRiskParamType	Fixed to 1018
	AccountID	Funds account
	ExtendedID	Exchange
	IntValue1	Maximum trade volume

There is no overlapping relationship between different types of risk control rules in this rule. If risk control rules are set at multiple levels simultaneously, these rules will take effect concurrently.

At present, the external risk control system is responsible for the execution of this risk control rule. Therefore, investors cannot obtain specific risk control parameter values or determine whether the risk control rules have been triggered through the API.

9.2.3 After-trade risk control of position volume

When the total long position volume of this option series is greater than or equal to the long position volume limit, or when the total short position is greater than or equal to the short position volume limit, set 'only close' permission for all instruments of this option series in the account. If, subsequently, the total position volume becomes less than the position volume limit through closing positions, this rule will not automatically set trading permission. However, if the administrator manually sets it to allow trading permission, the system will not change it back to 'only close' permission.

The YD after-trade risk control system supports the following six types of risk control:

- Single instrument position volume limit: Restricts the position volume of a specific instrument.
- Product instrument position volume limit: For each instrument under a specified product, restricts the position volume of individual instruments.
- Exchange instrument position volume limit: For each instrument under a specified exchange, restricts the position volume of individual instruments.
- Option series aggregate position volume limit: Restricts the total position volume of all instruments under a specified option series.
- Product aggregate position volume limit: Restricts the total position volume of all instruments under a specified product.
- Exchange product aggregate position volume limit: For each product under a specified exchange, restricts the total position volume of all instruments within that product.

The parameters of this risk control rule are as follows:

Level	Field	Description
Single instrument position volume limit	GeneralRiskParamType	Fixed to 1009
	AccountID	Funds account
	ExtendedID	Instrument
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit
Product instrument position volume limit	GeneralRiskParamType	Fixed to 1010
	AccountID	Funds account

Level	Field	Description
	ExtendedID	Product
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit
Exchange instrument position volume limit	GeneralRiskParamType	Fixed to 1011
	AccountID	Funds account
	ExtendedID	Exchange
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit
Option series aggregate position volume limit	GeneralRiskParamType	Fixed to 1005
	AccountID	Funds account
	ExtendedID	It can be in the following two formats: Option product code: Option expiration year: Option expiration month Option product code: Underlying instrument
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit
Product aggregate position volume limit	GeneralRiskParamType	Fixed to 1012
	AccountID	Funds account
	ExtendedID	Product
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit
Exchange product aggregate position volume limit	GeneralRiskParamType	Fixed to 1013
	AccountID	Funds account
	ExtendedID	Exchange
	IntValue1	Long position volume limit
	IntValue2	Short position volume limit

There is no overlapping relationship between different types of risk control rules in this rule. If risk control rules are set at multiple levels simultaneously, these rules will take effect concurrently.

At present, the external risk control system is responsible for the execution of this risk control rule. Therefore, investors cannot obtain specific risk control parameter values or determine whether the risk control rules have been triggered through the API.

9.2.4 Risk control of failed order submission counts

Failed order submission counts are supervised by exchanges, however, the supervision measures are relatively flexible. In order to avoid this, brokers usually want to control the failed order submission counts of investors to exchanges, and even those counts slightly exceeding the limit. Therefore, YD controls the failed order submission counts to exchanges through after-trade risk control. When reaching the threshold of failed order submission counts, YD's system will set the trading rights of customers to "trade prohibited". Please note that only failed orders sent back from exchanges can cause an increase of the total failed order submission counts. Those failed orders intercepted by the OMSs will not cause an increase of the failed order submission counts.

This risk control rule is inaccurate when calculating failed order submission counts to exchanges. If ydServer is restarted and the client does not recover it through the HA mode, the failed order submissions that occurred during the previous ydServer operation will not be counted.

The global level risk control parameters are shown in the following table:

Level	Field	Description
Global	GeneralRiskParamType	Fixed to 1004
	AccountID	Null indicates that it is valid for all investors, otherwise it indicates the user's AccountID value
	FloatValue	Threshold of failed order submission counts

At present, the external risk control system is responsible for executing this risk control rule. Therefore, investors cannot obtain specific risk control parameter values or determine whether the risk control rules have been triggered through the API.

9.2.5 Risk control of trade position ratio

When the trade volume under all instruments relating to a product reaches a certain extent, the ratio of the trade volume to the position volume of the product is not allowed to exceed a certain value, otherwise it will be considered a violation. When the risk control rule is triggered, YD will send a warning to the operators of the broker for their attention, however, no required risk control measures will be taken. This risk control rule is only available for options of SSE and SZSE. This risk control will be set on the counter, and the risk control parameters will also be distributed to the API.

The trade volume refers to the total volume of trades obtained after bid/ask offset under all instruments relating to a product.

The position volume refers to the higher one of the real-time net position volume and net preday position volume under all instruments relating to a product, namely the $\max(\text{product total net preday position volume}, \text{product real-time total net position volume})$. The calculation formula for the net position volume under a single instrument is: $|\text{long position volume} - \text{short position volume}| + \text{long frozen volume} + \text{short frozen volume}$, where the long/short position volumes are the values after deducting the corresponding frozen position volumes. The frozen volume mainly includes positions frozen due to position closing orders and combinations, etc. However, the frozen volume will not be included in the following two cases:

- The current day is the last trading day, and the frozen volume under this instrument is 0;
- When 2 days are left before the expiry date and the combination type relates to bull call spread, bear call spread, bull put spread and bear put spread, the frozen volume of the combination part will not be included, because these combinations will be decombined by exchanges according to the trading rules when settlement is made on the same day.

Assume that the net preday position volume is 5 lots, currently, the long/short position volumes under Instrument A are 11 lots and 5 lots, respectively, the long/short position volumes under Instrument B are 2 lots and 3 lots, respectively. At this time, for the next bull call spread combination, the first leg refers to the long position volume under Instrument A, the second leg refers to the short position volume under Instrument B, then the position volume calculation process is as follows:

Net real-time position volume under Instrument A = $| (11-1) - 5 | + 1 = 5$, where the minus one in two brackets refers to the combined frozen position volume

Net real-time position volume under Instrument B = $| 2-(3-1) | + 1 = 1$, where the minus one in two brackets refers to the combined frozen position volume

Net real-time position volume = $5+1=6$

Net position volume = $\max(5,6)=6$

The risk control parameters are shown in the following table:

Level	Field	Description
Product	GeneralRiskParamType	Fixed to YD_GRPT_TradePositionRatio
	AccountRef	-1 indicates that it is valid for all investors, otherwise it indicates the user's AccountRef value, which is the account itself for investor account login
	ExtendedRef	It can be set as empty to apply to all products, or a single product's ProductRef can be set to apply only to that specific product. ProductRef can be obtained from YDProduct.
	IntValue1	Trade volume threshold. The trade position ratio can be calculated only when the total trade volume reaches the threshold
	FloatValue	Trade position ratio threshold

Due to AccountRef being applicable to all accounts or specific accounts, and ExtendedRef being applicable to all exchanges or specific exchanges, there are a total of four levels. Their priorities from high to low (a high priority parameter configuration overrides a low priority one) are:

- For designated accounts, for designated exchanges.
- For designated accounts, for all exchanges.
- For all investors, for designated exchanges.
- For all investors, for all exchanges.

Currently, this risk control rule is executed by the ydClient. Investors can view the risk control parameters and risk rule status through ydClient, or obtain risk control parameters through APIs.

9.2.6 Order cancellation/submission ratio risk control

When the sum of order submission counts under all instruments relating to a product reaches a certain extent, (assumed as X), the ratio of the (order cancellation counts - X) / (order submission counts-X) for this product cannot exceed a certain value, otherwise it will be considered a violation. When the risk control rule is triggered, YD will send a warning to the operators of the broker for their attention, however, no required risk control measures will be taken. The order cancellation counts refer to the sum of those under all instruments relating to the product. The order submission counts refer to the sum of those under all instruments relating to the product. This risk control will be set on the counter, and the risk control parameters will also be delivered to the API.

The risk control parameters are shown in the following table:

Level	Field	Description
Product	GeneralRiskParamType	Fixed to YD_GRPT_OrderCancelRatio

Level	Field	Description
	AccountRef	-1 indicates that it is valid for all investors, otherwise it indicates the user's AccountRef value, which is the account itself for investor account login
	ExtendedRef	It can be set as empty to apply to all products, or a ProductRef of a specific product can be set to apply only to that particular product. ProductRef can be obtained from YDProduct.
	IntValue1	The order submission count threshold. The order cancellation / submission ratio is calculated only when the total order submission count reaches the threshold
	FloatValue	Order cancellation / submission ratio threshold

Due to AccountRef being applicable to all accounts or specific accounts, and ExtendedRef being applicable to all exchanges or specific exchanges, there are a total of four levels. Their priorities from high to low (a high priority parameter configuration overrides a low priority one) are:

- For designated accounts, for designated exchanges.
- For designated accounts, for all exchanges.
- for all accounts, for designated exchanges.
- for all accounts, for all exchanges.

Currently, this risk control rule is executed by the ydClient. Investors can view the risk control parameters and risk rule status through ydClient, or obtain risk control parameters through APIs.

9.2.7 After-trade message count risk control

At present, SHFE, DCE and CZCE manage investors' order placement/cancellation volume through message count control rather than mandatory control to reduce the pressure of invalid orders on exchanges. Many investors are not accustomed to this change, and even some of them may have paid high message count commissions due to failing to control the order volume. Considering that no corresponding message count commission is required on the primary OMS during trading, and its operation characteristics obtained through market-wide summary calculation also make it difficult to calculate message count commission temporarily, YD provides an after-trade risk control measure regarding message count to help brokers and investors reduce the risk on message count commission as much as possible. For the calculation methods of Message Count and OTR (Order to Trade Ratio), please refer to [Derivatives Message Count Commission](#).

YD provides two different message count control methods, namely, single message count control and message count / OTR control.

For single message count control, as long as the message count exceeds the set thresholds, investors' trading rights under the instrument will be set to "Only position closing allowed" or "Trade prohibited" depending on different thresholds. YD provides [before-trade-message-count-risk-control](#) that corresponds to after-trade risk control. When using the [before-trade-message-count-risk-control](#) for single message quantity, it is recommended to prioritize the use of before-trade risk control.

The risk control parameters are as follows, which are set in external risk control programs and will not be sent to APIs at present:

Level	Field	Description
Product	GeneralRiskParamType	Fixed to 1001
	AccountID	Funds account
	ExtendedID	Corresponding to YDProduct.ProductID, the only identified product
	IntValue1	message count threshold. The "Only position closing allowed" right can be set under the instrument after being triggered
	IntValue2	message count threshold. The "Trade prohibited" right can be set under the instrument after being triggered

Currently, this risk control rule is executed by an external risk control system, so investors cannot obtain specific risk control parameter values through APIs, nor can they know whether the risk control rule is triggered.

For message count / OTR control, as long as both the OTR and message count exceed the set thresholds, investors' trading rights under the instrument will be set to "Only position closing allowed" or "Trade prohibited" depending on different thresholds. The risk control parameters are as follows:

Level	Field	Description
Product	GeneralRiskParamType	Fixed to 1002
	AccountID	Funds account
	ExtendedID	Corresponding to YDProduct.ProductID, the only identified product
	IntValue1	message count threshold. The "Only position closing allowed" right can be set under the instrument after being triggered
	IntValue2	message count threshold. The "Trade prohibited" right can be set under the instrument after being triggered
	FloatValue	OTR threshold

Currently, this risk control rule is executed by an external risk control system, so investors cannot obtain specific risk control parameter values through APIs, nor can they know whether the risk control rule is triggered.

9.2.8 Position opening not allowed when approaching to expiry date

In order to facilitate brokers to set trading rights for prohibiting position opening under instruments approaching their expiry dates, YD provides a convenient setting method according to after-trade risk control rules.

This risk control rule can be used for checking for reaching the set threshold through `YDInstrument.ExpireTradingDayCount`. If yes, the "Only position closing allowed" trading right of each investor under the corresponding instrument can be set accordingly.

The parameters of this risk control rule are as follows:

Level	Field	Description
Product	GeneralRiskParamType	Fixed to 1003
	AccountID	Funds account
	ExtendedID	Corresponding to <code>YDProduct.ProductID</code> , the only identified product
	IntValue1	Threshold of days left to expiry date

Currently, this risk control rule is executed by an external risk control system, so investors cannot obtain specific risk control parameter values through APIs, nor can they know whether the risk control rule is triggered.

10 Management

10.1 Export Inverstors Data

Investors and administrators can use following method to export investors data at any time. dir is the directory for exporting data, accountIDs is the list of exported accounts(separated by spaces). Investors should fill accountIDs with their own accounts, while administrators can fill accountIDs with accounts to be exported(Empty strings indicated all accounts).

```
1 | virtual bool exportData(const char *dir, const char *accountIDs="")
```

The following files will be generated in the export directory:

File name	Description
instrument.csv	Instrument info
rate.csv	Derivatives commission rate and margin rate
cashCommissionRate.csv	Spots' commission rate
tradingRight.csv	Trading right
account.csv	Account info, including fund information
position.csv	Derivatives position
holding.csv	Spots position
combPosition.csv	Comb position
marginSideInfo.csv	Large-side margin information
order.csv	Order
trade.csv	Trade
quote.csv	Quote

10.2 Password Modification

Within the trading session, investors can modify their trading passwords via API. Once modified, the new password will remain valid indefinitely. The method for calling this API is as follows:

```
1 | virtual bool changePassword(const char *username, const char *oldPassword, const char *newPassword)
```

The result of the password modification will be returned through the following callback. If the value of 'errorNo' is 0, it indicates a successful modification. Other values indicate an error occurred, usually due to the incorrect old password(YD_ERROR_OldPasswordMismatch) or failed to pass password strength check(YD_ERROR_WeakPassword).

```
1 | virtual void notifyChangePassword(int errorNo)
```

10.3 Logging

The YD API, by default, will write logs to the 'log' directory under the path of the executable program. Investors can utilize this mechanism to write their own log information. If the 'log' directory is not found, no log files will be generated.

```
1 | virtual void writeLog(const char *format,...)
```

The logs are split on a daily basis, and typically the API will generate the following log sample.

```
1 | 14:59:54 YD API start
2 | 14:59:54 version 1.108.36.33
3 | 14:59:54 build time Mar 3 2022 17:37:32
4 | 14:59:54 build version GCC 10.2.0
5 | 14:59:54 TCP trading server 0 connected
```

10.4 Deposit and Withdrawal

YD provides an on-site deposit and withdrawal function. Administrator users can deposit and withdraw funds for investors and adjust the investor's fund usage limit during the trading session. For deposit and withdrawal services, please refer to [Net Deposit and Withdrawal Amount](#). This function can only be called by administrator users with deposit and withdrawal permissions.

```
1 | virtual bool alterMoney(const YDAccount *pAccount, int alterMoneyType, double alterValue)
```

The parameters of the above method are described as follows:

Parameter	Description
YDAccount	Pointer to the account which needs deposit and withdrawal to

Parameter	Description
alterMoneyType	alter money type
alterValue	alter value

The deposit and withdrawal types and their meanings are shown in the following table:

Alter Money Type	Descriptions
YD_AM_ModifyUsage	Adjust the investor's fund usage limit. If the adjusted value is less than the original value, it will check whether the adjustment will cause Available to be negative. If it is negative, the adjustment is not allowed. For the calculation method of Available, please refer to usable , please note that the check condition here is Available(), not Useable()
YD_AM_Deposit	Deposit. Add the amount specified by alterValue to the existing accumulated deposit amount.
YD_AM_FrozenWithdraw	Freeze withdrawal. Add the amount specified by alterValue to the existing frozen withdrawal amount.
YD_AM_CancelFrozenWithdraw	Cancel frozen withdrawal. Reduce the amount specified by alterValue based on the existing frozen withdrawal amount. If alterValue is 0, it means canceling all frozen withdrawals.
YD_AM_TryWithdraw	First check whether the available funds are sufficient to withdraw money. If so, withdraw money. If not, return an error.
YD_AM-Withdraw	Withdrawal: Add the amount specified by alterValue to the existing cumulative withdrawal amount.
YD_AM_DepositTo	Deposit to the specified amount. Directly set the current cumulative deposit amount. If the adjustment value is less than the current value, the adjustment is not allowed.
YD_AM-WithdrawTo	Withdraw to a specified amount. Directly set the current cumulative withdrawal amount. If the adjustment value is less than the current value, no adjustment is allowed.
YD_AM_ForceModifyUsage	Forces investors to adjust their fund usage limits without checking whether Available will become a negative number after the adjustment.

YD has many deposit and withdrawal related tools, which will affect the final net deposit and withdrawal results of the counter. We recommend that brokers only choose to use one of them and not mix them, otherwise it will be difficult to analyze the net deposit and withdrawal results when these tools act on deposits and withdrawals at the same time. In order to help investors and brokers analyze and understand the impact of using multiple tools in certain special situations, the following briefly introduces how each tool affects counter deposits and withdrawals:

- Manual deposits and withdrawals on ydClient and the management side: use YD_AM_Deposit and YD_AM-Withdraw to deposit and withdraw
- ydSync: read deposit and withdrawal flow from deposit and withdrawal sources (CTPRisk, files, etc.), summarize in ydSync, and deposit and withdraw through YD_AM_DepositTo and YD_AM-WithdrawTo
- ydFundManager and the management side's fund synchronization function: read deposit and withdrawal flow from deposit and withdrawal sources (CTPRisk, files, manual operations, etc.), summarize in ydSync, and deposit and withdraw through YD_AM_DepositTo and YD_AM-WithdrawTo

11 Tools

11.1 ydcmd

ydcmd is a tool for investors to use. It mainly solves the basic query and emergency processing needs when brokers restrict investors from using YDClient. Currently, it supports batch order withdrawal, batch quote withdrawal, password modification, and customer data export. The help information of this command is as follows, which can be obtained by entering ydcmd in the command line.

```
1 ydcmd ydExport <config file> [<dir> [<username> [<password>]]]
2 ydcmd ydChangePassword <config file> [<username> [<password> [<new password>]]]
3 ydcmd ydCancelOrders <config file> [<username> [<password> [once]]]
4 ydcmd ydCancelQuotes <config file> [<username> [<password> [once]]]
```

ydcmd is developed based on ydApi. Its operating principle and characteristics are consistent with those of ordinary API clients. The general conventions of this tool are as follows:

- The first parameter is always the function name. After changing the ydcmd file name to the function name, the corresponding function can be used directly through the function name, and the subsequent parameters remain unchanged. For example, after renaming ydcmd to ydExport, you can directly call ydExport config.txt <dir> <username> <password> in the command line.
- The second parameter is always the configuration file path. The configuration method is exactly the same as the configuration file requirements of ydApi. Usually, the configuration file in production can be directly reused
- There are multiple ways to enter the username and password. If the username and password are not specified in the command line, you will be prompted to enter them after the program runs. You can only specify the username in the command line, and you will be prompted to enter the password after the program runs.

Although ydcmd is mainly used by investors, some functions can also be used by administrators in need. For functions that can be used by administrators, please refer to the detailed description of each function below.

11.1.1 Export-Investor-Data Tools

Export all business data of investors to multiple files. The output content is the data snapshot when notifyCaughtUp is received. Administrators can use it to export all business data of all investors on the counter. The output files please refer to [Export Investors Data].(#export-investors-data)

You need to specify the export directory when running the program. If you do not specify it, it will be exported to the current directory.

11.1.2 Change-Password Tools

Change investor's password. Administrators can use this function to change their own password.

A new password can be specified in the command line. If a new password is not specified in the command line, the program will prompt you to enter the new password twice after running.

11.1.3 Cancel-All-Orders Tools

Cancel all ordinary orders of investors in pending order status, including:

- Ordinary orders in pending order status, including trigger orders
- quote-derived orders in pending order status, and the exchange allows the cancellation of quote-derived orders

Please note that non-ordinary orders (YDOrderFlag is not 0) are not within the cancellation range, such as exercise, abandonment of automatic exercise, etc.

If once is specified in the parameter, the program will issue cancellation instructions one by one according to the ordinary pending order list when notifyCaughtUp is received, and then end the operation. It will not wait for the result, nor will it issue cancellation instructions again. The actual number of cancellation orders issued will be output on the command line. If the number of pending orders is huge, it may trigger counter flow control or exchange flow control, so one run may not ensure all orders are cancelled. In this case, you can run it multiple times or use the interactive cancellation mode.

When using once mode, the return value of the program is defined as follows:

- 0: No cancelable orders
- 1: Other errors
- 2: There are cancelable orders, and the API has issued all cancel orders
- 3: There are cancelable orders, and the API has issued a cancel order instruction but failed to send

If once is not specified in the parameters, the program will interactively run the order cancellation logic, and the following information will be continuously refreshed on the command line:

```
1 | xxx order(s) can be canceled, press return to start
```

Every time the investor presses Enter, a cancel order instruction will be issued for the current cancelable order, and then the number of cancelable orders will continue to be refreshed in the next line. When the number of cancelable orders is 0, the program ends automatically.

11.1.4 Cancel-All-Quotes Tools

Cancel all quotes that are still pending orders on at least one side of the investor.

If once is specified in the parameter, the program will issue quote cancellation instructions one by one according to the list of revocable quotes when notifyCaughtUp is received, and then end the operation. It will not wait for the result, nor will it issue quote cancellation instructions again. The command line will output the actual number of quote cancellations issued. If the number of revocable quotes is huge, it may trigger counter flow control or exchange flow control, so one run may not ensure that all quotes are withdrawn. In this case, you can run it multiple times or use the interactive quote cancellation mode.

When using once mode, the return value of the program is defined as follows:

- 0: No revocable quotes
- 1: Other errors
- 2: There are revocable quotes, and the API has issued all quote cancellation instructions
- 3: There are revocable quotes, and the API has issued a quote cancellation instruction but failed to send

If once is not specified in the parameters, the program will interactively run the quote cancellation logic, and the following information will be continuously refreshed on the command line:

```
1 | xxx quote(s) can be canceled, press return to start
```

Every time the investor presses Enter, a quote cancellation instruction will be issued for the current revocable quote, and then the number of revocable quotes will continue to be refreshed on the next line. When the number of revocable quotes is 0, the program ends automatically.