**Python Market API Development Specification**

Table of contents

[I. Overview 3](#_Toc184912737)

[1.1. Background 3](#_Toc184912738)

[1.2. API interface overview and architecture 4](#_Toc184912739)

[1.3. Global error code 5](#_Toc184912740)

[2. Futures and stock general market API interface reference 7](#_Toc184912741)

[1. Interface working process 7](#_Toc184912742)

[2. Interface mode 8](#_Toc184912743)

[3. Function interface description 8](#_Toc184912744)

[3.1. API instance object creation 8](#_Toc184912750)

[3.2. API function request and callback method 8](#_Toc184912751)

[1) onFrontConnected method 9](#_Toc184912752)

[2) onFrontDisconnected method 9](#_Toc184912753)

[3) onHeartBeatWarning method 9](#_Toc184912754)

[4) Set the heartbeat timeout setHeartBeatTimeout 9](#_Toc184912755)

[5) Login reqUserLogin /onRspUserLogin 11](#_Toc184912756)

[6) Market subscription request/response reqMarketData/onRspMarketData 11](#_Toc184912757)

[7) Futures market information push onRtnSnapData 14](#_Toc184912758)

[8) Futures transaction data push onRtnDealData 17](#_Toc184912759)

[9) Futures statistics data push onRtnStatisticsData 17](#_Toc184912760)

[10) Stock market push onRtnMarketData 18](#_Toc184912761)

[11) Hong Kong stock broker obtains request reqBrokerData/OnRtnBrokerData 21](#_Toc184912762)

[12) Hong Kong stock broker pushes onRtnBrokerData 22](#_Toc184912763)

[13) Push onRtnTradeDate 23 on trading day](#_Toc184912764)

[14) General error onRspError 23](#_Toc184912765)

[6) FAQ - Stock Quotes 24](#_Toc184912766)

# Overview

## background

This API can be used to obtain market data for the corresponding markets.

Currently supported exchanges/markets include (but not limited to ) the following:

Futures

|  |  |  |
| --- | --- | --- |
| No | Exchanges | illustrate |
| 1 | CME | , COMEX, NYMEX , CBOT under CME group  Official Website:  https://www.cmegroup.com/ |
| 2 | HKEX | Hong Kong Exchanges and Clearing Limited  Official Website:  http://www.hkex.com.hk |
| 3 | SGX | Singapore Exchange  Official Website:  http://www.sgx.com |
| 4 | I CE | Intercontinental Exchange  Official Website:  https://www.theice.com/index |
| 5 | LME | London Metal Exchange  Official Website:  https://www.lme.com/ |

Stocks

|  |  |  |
| --- | --- | --- |
| No | market | illustrate |
| 1 | NASD | Stocks in the US market.  It is important to note that after the 2008 financial crisis, the United States passed a bill allowing listed companies to be traded on all U.S. exchanges. For example, stocks listed on the NYSE can be traded on the NASDQ exchange and vice versa. |
| 2 | HKEX | Stocks on the Hong Kong Stock Exchange |
| 3 | KRX | South Korean stocks |

## API overview and architecture

This API is a Python-based class library that implements the functions required for trading by using and extending the interfaces provided by the class library. In view of the widespread use of CTP interface in the domestic futures industry, this API interface imitates the CTP interface so that users with experience in using CTP interface can quickly get started with development.

For the following error codes, a value of 0 indicates success, regardless of where in this API.

**Interface function classification:**

Futures market data callback function (futures market data must be connected and logged in to the futures market data)

|  |  |
| --- | --- |
| onRtnSnapData () | Order Book snapshot data push |
| onRtnDealData () | Transaction data push |
| onRtnStatisticsData () | Push data such as opening/high/low/settlement price on trading day |
| onRtnTradeDate () | Not implemented, reserved |

Stock market data callback function (stock quotes need to be connected and logged in to the stock quotes)

|  |  |
| --- | --- |
| onRtnMarketData () | Stock market data push |
| onRtnBrokerData () | Unique to Hong Kong stocks, broker queue push |

Quote subscription function and response

|  |  |
| --- | --- |
| reqMarketData () | Subscription contract function |
| onRspMarketData () | Subscribe to responses (only futures markets have responses) |

## Global error code

1. Common errors in business functions

// Dictionary of error messages in quotes

dict error

{

ErrorID [int]: Error code

ErrorMsg [str]: Error description

};

|  |  |  |
| --- | --- | --- |
| Error Code  ( ErrorID ) | Error description | illustrate |
| 0 | success | Various functions return results successfully |
| 00000 | Processing success |  |
| 10001 | Incorrect username |  |
| 10002 | Wrong login password |  |
| 10003 | The number of incorrect passwords has exceeded the limit, and the user has been frozen. Please contact customer service to unfreeze the account. |  |
| 10004 | The user has been frozen, please contact customer service to unfreeze |  |
| 10011 | Illegal login |  |
| 10012 | No login permission |  |
| 10013 | API authentication error |  |
| 10014 | No permission for this stock exchange |  |
| 10015 | No authority from this futures exchange |  |
| 10016 | Not in whitelist |  |
| 10017 | Limited login times |  |
| 10018 | MarketType is empty |  |
| 10019 | Please wait for the last subscription instruction to end |  |
| 10020 | Not logged in |  |
| 10021 | Send -1 Error |  |
| 10022 | Send 0 Error |  |
| 10023 | Subscription type and service do not match |  |
| 10024 | Invalid subscription format |  |
| 10025 | The permission verification response is empty |  |

1. Disconnection Generic Error

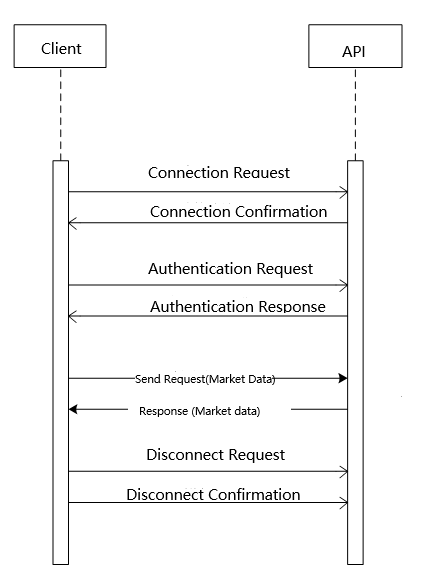
onFrontDisconnected (int nReason)

|  |  |  |
| --- | --- | --- |
| Error Code  ( nReason ) | Error description | illustrate |
| 4 001 | Setting blocking failed | Socket underlying error |
| 4 002 | Setting non-blocking failure | Socket underlying error |
| 4 003 | Failed to receive data | Socket underlying error |
| 4 004 | Received 0 length data, disconnected | Socket underlying error |
| 4 005 | Failed to send data | Socket underlying error |
| 4 006 | Sending 0 length data, disconnected | Socket underlying error |
| 4 007 | Failure of selection | Socket underlying error |
| 4 008 | Server Not Responding | Socket underlying error |
|  |  |  |
|  |  |  |

# Futures and Stocks General Quotes API Reference

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Business Type | Request interface | Request response interface | | Report interface | |
| connect | i nit |  |  | onFrontConnected | Connection Return |
| disconnect |  |  |  | onFrontDisconnected | Disconnection report |
| Heartbeat | setHeartBeatTimeout​ |  |  | onHeartBeatWarning | Heartbeat warning |
| Log in | reqUserLogin | onRspUserLogin | Login response |  |  |
| Quote Subscription | reqMarketData | onRspMarketData | Subscription Response |  |  |
|  |  |  | onRtnMarketData | Market push |
| Broker Access  (Hong Kong stocks only ) | reqBrokerData |  |  | onRtnBrokerData | Broker Queue Returns |
| Trading day push |  |  |  | onRtnTradeDate | Trading day push |

## Interface working process



## Interface Mode

The market\_demo.py file provides a MarketApi class . By inheriting the MarketApi class, you can issue operation requests and overload callback functions to handle the responses of background services.

## Function interface description



## API instance object creation

1. createMarketApi method

Create an API object instance.

**Function prototype:**

createMarketApi (bRecordLog : bool, pszFlowPath : str = "" ,isFutureMode : str = " Y ")

**parameter:**

bRecordLog: Whether to record market data (market data files will be very large, it is usually recommended to turn off recording )

true: record market data; false : do not record

pszFlowPath: log data file path

isFutureMode: Y: Futures mode (default value ) ; N: Stock mode

Note: For futures and stock quotes, you need to connect and log in to the quotation frontend.

When connecting to the futures market frontend, isFutureMode is set to Y;

When connecting to the stock quote frontend, isFutureMode is set to N

## API function request and callback method

### 1) onFrontConnected method

This method is called when the client establishes a communication connection with the frontend (before logging in).

**Function prototype:**

onFrontConnected()

This method is called after initialization is completed, in which the user login task can be completed.

### 2) onFrontDisconnected method

This method is called when the client loses the connection with the frontend communication. When this happens

After that, the API will automatically reconnect and the client does not need to do anything. Automatic reconnection address, which may be the original registration address

It may also be other available communication addresses supported by the system, which are automatically selected by the program.

**Function prototype:**

onFrontDisconnected (nReason : int )

**parameter:**

nReason : Reason for disconnection

### 3) onHeartBeatWarning method

Heartbeat timeout warning. This method is called when no message is received for a long time.

**Function prototype:**

onHeartBeatWarning(nTimeLapse : int )

**parameter:**

nTimeLapse : The time since the last message was received

SetHeartBeatTimeout

### 4 ) Set the heartbeat timeout setHeartBeatTimeout

Function prototype:

|  |
| --- |
| setHeartBeatTimeout ( iTimeout : int , bIsStrictCheck : bool = true ) |

Parameters: iTimeout : heartbeat timeout, in seconds. Please set the heartbeat time to at least 30 seconds .

bIsStrictCheck : Whether to check heartbeat timeout. The default value is true, which means checking the heartbeat; if it is false, the heartbeat is not checked.

### 5 ) Login r eqUserLogin / o nRspUserLogin

Function prototype:

|  |
| --- |
| reqUserLogin( req: dict, reqid: int) |

Parameter: req

|  |
| --- |
| // User ID  UserId[str]  // User password  UserPwd[str]  // User type  UserType[str]  // Software Name  SoftwareName[str]  // Software version number  SoftwareVersion[str]  // Authorization code  AuthorCode[str]  // error message  ErrorDescription[str]  // Market distributor ID (since v2.1.0.1)  BrokerIDForMarketPrice [str] |

Function prototype:

|  |
| --- |
| onRspUserLogin( error: dict, reqid: int, last: bool) |

Parameters: error

|  |
| --- |
| // Error code  ErrorID[int]  // Error description  ErrorMsg[str] |

Error code reference 1.3

### 6 ) Market subscription request/response reqMarketData/onRspMarketData

Function prototype:

|  |
| --- |
| reqMarketData(req: dict, reqid: int) |

Parameter: req

|  |
| --- |
| // Financial type  MarketType [str]  // Request type  SubscMode [str]  // Exchange  ExchangeCode [str]  // Maximum number of subscriptions at a time  MarketCount [int]  // Single subscription contract array  MarketTrcode [list[str]]  // error message  ErrorDescription [str] |
|  |

illustrate:

1. Subscribe to quotes by single contract. The exchange field is required. for example

req[ “ExchangeCode” ] = “HKEX”

req[ “MarketCount” ] = 1

req[ “ MarketTrcodes ” ] = [ “00001.HK ” ]

1. the Nth subscription call response result ( whether successful or failed) comes back, the N +1th subscription call can be made .
2. If there is no response to the Nth subscription call within the specified maximum time of 1 second, the user is allowed to make the N +1th subscription call .
3. Special instructions for setting **ExchangeCode** in subscription requests
   1. For exchanges not under the C ME group, please refer to the exchange code returned by " Exchange List Query " in the document [Direct Access Software Member Market API Commodity Contract Acquisition Interface Document.docx] , or the exchange code seen on the Direct Access Client Interface.
   2. There are 4 exchanges under C ME group , CME\_CBT , CME\_COMEX , CME\_NYMEX, CME. When subscribing, fill in the exchange code of the product.

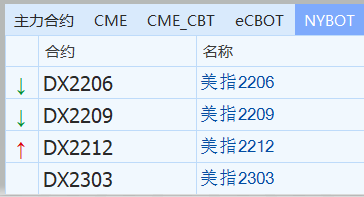
(For example, the crude oil CL product of NYMEX under CME , the exchange code is CME\_NYMEX when subscribing)

Note: After successfully subscribing, the exchange codes in the market data pushed are only CME (including CME, CME\_COMEX, CME\_NYMEX) and CME\_CBT.

1. **MarketTrcode** description in subscription request

You can refer to the contract code returned by " Contract List Query " in the document [Direct Access Software Member Market API Commodity Contract Acquisition Interface Document.docx] , or the exchange code seen on the Direct Access Client Interface.

The first line in the figure below is the exchange code in the direct access system, and the first column is the contract code



Function prototype:

|  |
| --- |
| onRspMarketData( data: dict , error: dict , reqid: int, last: bool ) |

Parameters: req

|  |
| --- |
| // Financial type  MarketType [str]  // Request type  SubscMode [str]  // Exchange  ExchangeCode [str] |

Parameters: error

|  |
| --- |
| // Error code  ErrorID[int]  // Error description  TDAStringType [str] |

Note:

Each time the user calls the subscription function of **r eqMarketData** , if the value of reqid is set to an increasing serial number , the subscription result can be determined by calling back the reqid of the function onRspMarketData .

### 7 ) Futures market information push onRtnSnapData

Function prototype:

|  |
| --- |
| o nRtnSnapData ( data: dict ) |

Parameters: data

|  |
| --- |
| // Exchange code  ExchangeCode[str]  //Contract code  TreatyCode[str]  // Bid price  BuyPrice[float]  // Buy volume  BuyNumber[int]  // Ask price  SalePrice[float]  //Selling volume  SaleNumber[int]  // Latest price  CurrPrice[float]  // Current quantity  CurrNumber[int]  // Market time unixepoch (accuracy to milliseconds)  Time[int]  // Volume  FilledNum[int]  // Position  HoldNum[int]  // Bid price 2  BuyPrice2[float]  // Bid price 3  BuyPrice3[float]  // Bid price 4  BuyPrice4[float]  // Buy price 5  BuyPrice5[float]  // Buy 2  BuyNumber2[int]  // Buy quantity 3  BuyNumber3[int]  // Buy quantity 4  BuyNumber4[int]  // Buy quantity 5  BuyNumber5[int]  // Selling price 2  SalePrice2[float]  // Selling price 3  SalePrice3[float]  // Selling price 4  SalePrice4[float]  // Selling price 5  SalePrice5[float]  //Selling quantity 2  SaleNumber2[int]  //Sell quantity 3  SaleNumber3[int]  //Sell quantity 4  SaleNumber4[int]  //Sell quantity 5  SaleNumber5[int]  // Hide the bid price  HideBuyPrice[float]  //Hide purchase volume  HideBuyNumber[int]  // Hide the selling price  HideSalePrice[float]  // Hide sales volume  HideSaleNumber[int]  // Trading day  TradeDay[str]  // Bid price 6  BuyPrice6[float]  // Bid price 7  BuyPrice7[float]  // Buy price 8  BuyPrice8[float]  // Buy price 9  BuyPrice9[float]  // Buy price 10  BuyPrice10[float]  // Buy 6  BuyNumber6[int]  // Buy quantity 7  BuyNumber7[int]  // Buy quantity 8  BuyNumber8[int]  // Buy quantity 9  BuyNumber9[int]  // Buy quantity 10  BuyNumber10[int]  // Selling price 6  SalePrice6[float]  // Selling price 7  SalePrice7[float]  // Selling price 8  SalePrice8[float]  // Selling price 9  SalePrice9[float]  // Selling price 10  SalePrice10[float]  //Selling quantity 6  SaleNumber6[int]  //Selling quantity 7  SaleNumber7[int]  //Selling quantity 8  SaleNumber8[int]  // Selling quantity 9  SaleNumber9[int]  //Sell quantity 10  SaleNumber10[int]  // Data source  DataSourceId[str] |

### 8) Futures transaction data push onRtnDealData

Function prototype:

|  |
| --- |
| onRtnDealData ( data: dict ) |

Parameters: data

|  |
| --- |
| // Exchange code  ExchangeCode[str]  //Contract code  TreatyCode[str]  // Transaction price  TradePrice [float]  // Volume  TradeVol [int]  // Active buy or active sell (unimplemented, reserved)  AggressorSide [str]  // (unused, reserved)  TradeFlag [str]  // Timestamp unixepoch (accuracy to milliseconds)  DateTimestamp [str]  // (unused, reserved)  Extends [str] |

### 9) Futures statistics data push onRtnStatisticsData

Function prototype:

|  |
| --- |
| onRtnStatisticsData( data: dict ) |

Parameters: data

|  |
| --- |
| // Exchange code  ExchangeCode[str]  //Contract code  TreatyCode[str]  // Highest price  HighPrice [float]  // Lowest price  LowPrice [float]  // Opening price  OpenPrice [float]  // Closing price (unimplemented)  ClosePrice [float]  // Yesterday's settlement price  TySettlePrice [float]  // Real-time settlement price ( the settlement price pushed by the exchange, reflected in real time)  IntradaySettlePrice [float]  // Price limit (unimplemented, reserved)  HighLimit [float]  //Lower limit price (unrealized, reserved)  [float] LowLimit [float]  // Position  Position [int]  // Yesterday's position  PrePosition [int]  // Opening price Flag (unimplemented, reserved)  OpenPrxBitFlag [str]  // Real-time settlement price Flag (unimplemented, reserved)  SetlPrxBitFlag [str]  // Timestamp unixepoch (accuracy to milliseconds)  DateTimestamp [int]  // Data source  DataSourceId [str] |
|  |
|  |

### 10 ) Stock quote push onRtnMarketData

Function prototype:

|  |
| --- |
| onRtnMarketData( data: dict ) |

Parameters: data

|  |
| --- |
| // Exchange code  ExchangeCode[str]  //Contract code  TreatyCode[str]  // Bid price  BuyPrice[str]  // Buy volume  BuyNumber[str]  // Ask price  SalePrice[str]  //Selling volume  SaleNumber[str]    // Latest price  CurrPrice[str]  // Current quantity  CurrNumber[str]  // Highest price of the day  High[str]  // The lowest price of the day  Low[str]  // Opening price  Open[str]  //Closing price  IntradaySettlePrice [str]  //Stock: Yesterday's closing price  Close[str]  // Quote time  // Quote time  Time[str]  // Trading volume: If it is a stock quote, this field is the trading volume  FilledNum[str]  // Holdings: If it is a stock quote, this field is the transaction amount  HoldNum[str]  // Bid price 2  BuyPrice2[str]  // Bid price 3  BuyPrice3[str]  // Bid price 4  BuyPrice4[str]  // Buy price 5  BuyPrice5[str]  // Buy 2  BuyNumber2[str]  // Buy quantity 3  BuyNumber3[str]  // Buy quantity 4  BuyNumber4[str]  // Buy quantity 5  BuyNumber5[str]  // Selling price 2  SalePrice2[str]  // Selling price 3  SalePrice3[str]  // Selling price 4  SalePrice4[str]  // Selling price 5  SalePrice5[str]  //Selling quantity 2  SaleNumber2[str]  //Sell quantity 3  SaleNumber3[str]  //Sell quantity 4  SaleNumber4[str]  //Sell quantity 5  SaleNumber5[str]  // Hide the bid price  HideBuyPrice[str]  //Hide purchase volume  HideBuyNumber[str]  // Hide the selling price  HideSalePrice[str]  // Hide sales volume  HideSaleNumber[str]  // Limit down price  LimitDownPrice[str]  // Price limit  LimitUpPrice[str]  // Trading day  TradeDay[str]  // Bid price 6  BuyPrice6[str]  // Bid price 7  BuyPrice7[str]  // Buy price 8  BuyPrice8[str]  // Buy price 9  BuyPrice9[str]  // Buy price 10  BuyPrice10[str]  // Buy 6  BuyNumber6[str]  // Buy quantity 7  BuyNumber7[str]  // Buy quantity 8  BuyNumber8[str]  // Buy quantity 9  BuyNumber9[str]  // Buy quantity 10  BuyNumber10[str]    // Selling price 6  SalePrice6[str]  // Selling price 7  SalePrice7[str]  // Selling price 8  SalePrice8[str]  // Selling price 9  SalePrice9[str]  // Selling price 10  SalePrice10[str]  //Selling quantity 6  SaleNumber6[str]  //Selling quantity 7  SaleNumber7[str]  //Selling quantity 8  SaleNumber8[str]  // Selling quantity 9  SaleNumber9[str]  //Sell quantity 10  SaleNumber10[str]  // HKEX Stock Quotes: Transaction Type  TradeFlag[str]  // Exchange data timestamp  DataTimestamp[str]  // Data source  DataSourceId[str]  // Number of shares that can be shorted (for US stock quotes)  CanSellVol[str]  // Market conditions are divided into two cases (meaning that the transaction volume in Y and 2 can be counted into minute data, but that in Z cannot )  // Directly connected to the exchange's market conditions -- Y: The current callback data is the latest transaction data; Z: The current callback data is the snapshot data  // Quotes of non-directly connected exchanges -- 2: Current callback data includes the latest transaction data and market changes Z: Transaction volume in snapshot data // Not available for statistics  QuoteType[str] |

### 11 ) Hong Kong stock brokers get request reqBrokerData/OnRtnBrokerData

Function prototype:

|  |
| --- |
| reqBrokerData(req: dict, reqid: int) |

Parameter: req

|  |
| --- |
| //Contract code  HeyuCode[str]  // error message  ErrorDescription[str] |

Broker responses and returns are placed in the interface onR **tn** BrokerData

### 12 ) Hong Kong stock brokers push onRtnBrokerData

Function prototype:

|  |
| --- |
| onRtnBrokerData(data: dict, error: dict, reqid: int, last: bool) |

Parameters: data

|  |
| --- |
| // Broker data  BrokerData[str] |

Parameters: error

|  |
| --- |
| // Error code  ErrorID[int]  // Error description  ErrorMsg[str] |

Note: Broker queue is not available yet

### 13 ) Push notification on the trading day

Function prototype:

|  |
| --- |
| onRtnTradeDate( data: dict, error: dict, reqid: int, last: bool) ) |

Parameters: data

|  |
| --- |
| // Transaction date ( yyyy-MM-dd)  TradeDate [str]  // Trading variety list, format is "GC,SI,HG"  TradeProduct [str] |

illustrate:

1. Only futures have this function of pushing trading days.
2. The push function on trading days requires API users to adjust it themselves when encountering holidays, for example:

The Hong Kong Stock Exchange is on holiday on 2019/4/5 . The trading day pushed by the Hang Seng Index HSI at 2019/4/4 17:00 is 2019-04-05, but the actual trading day should be 2019-04-08

1. Each product will be pushed 15 minutes in advance of the opening time.

Parameters: error

|  |
| --- |
| // Error code  ErrorID[int]  // Error description  ErrorMsg[str] |

### 14 ) General Error onRspError

Function prototype:

|  |
| --- |
| o nRspError ( error: dict, reqid: int, last: bool ) |

Parameters: error

|  |
| --- |
| // Error code  ErrorID[int]  // Error description  ErrorMsg[str] |

Generally, if there is an error in the parsing of market data returns, it will be pushed to this interface.

# FAQ - Stock Quotes

1. Question:

There are two time fields, one is time and the other is DataTimestamp. What is the difference between the two? Some products have data in the datatimestamp field, while others do not. What is going on?

-->Answer:

time is converted to local time in China

DataTimestamp is the data timestamp sent by the exchange, in Unix epoch format

For products directly connected to the exchange, DataTimestamp will have data, and for products not directly connected, it will be empty

2. Question:

In the received price, bidprice and askprice are sometimes empty.

-->Answer:

Please refer to the description of the QuoteType field in the data structure in the market development document.

// Market conditions are divided into two cases (meaning that the transaction volume in Y and 2 can be counted into minute data, but that in Z cannot)

// Directly connected to the exchange's market conditions -- Y: The current callback data is the latest transaction data; Z: The current callback data is the snapshot data

// Quotes from non-directly connected exchanges -- 2: Current callback data includes the latest transaction data and market changes Z: Transaction volume in snapshot data // cannot be used for statistics