# Tradex API C# Wrapper (v0.60)

## Terminology

- **Tradex instance**: a running instance of Tradex client that has already logged in; A TMS Client or a Multitradex Client could be a Tradex instance.
- **User program**: your program that communicates with the Tradex instance through this API.

## Features

- **Connect to Tradex instance's Trade Port**:
  Once User program is connected to Tradex instance's Trade Port, the following message could be pushed to User program: OrderUpdate, FillUpdate, PositionUpdate,AccountUpdate.

  Once User program is connected to Tradex instance's Trade Port, the following functions could be called by User program: AddOrder(…), AmendOrder(…), CancelOrder(…).

  Once a new connection between the User program and Tradex instance's Trade Port is established, intraday (or several days ago ) history data (including OrderUpdate, FillUpdate) and PositionUpdate will be pushed to User program at the very first time.

- **Connect to Tradex instance's Market Port**:
  Once User program is connected to Tradex instance's Price Port, the following message will be pushed to User program: PriceUpdate.

  Once User program is connected to Tradex instance's Trade Port, the following method could be called by User program: SubscribePrice(…), UnsubscribePrice(…).

## Prerequisite

In order to make your User Program ready for Tradex API, make sure that you have done the following steps:

1. TMS Client or MultiTradex Client is installed on your computer.
2. Activate the Tradex API function. (Will be explained in the next section)
3. The Tradex instance is logged in and running.

## How to Activate the API

**The API interface is disabled by default**. To activate the API listener on MultiTradex/TMS client, please create a file named "*remote.txt*" in the folder:

MultiTradex: **%appdata%\multitradex**

TMS: **%appdata%\TMClient**

(**Note that this is different from the installation folder of MultiTradex or TMClient**)

Open the txt file and add the line below:

**enabled=true**

and then save the changes to the file.

The next time you log into MultiTradex/TMS client, in either Demo or Live mode, the API listener will fire up (to accept text commands). Make sure when the firewall asks for your permission to open up TCP ports, approve it.

The trade interface listens on TCP Port 55241

The market data interface listens on TCP port 55242

## TradexAPI DLL （Add these DLLs into your C# project as reference）

1. CSharpLib.dll
2. pbAPIAdapter.dll
3. TradingPlatformCommons.dll

These 3 DLL files are located at the following directory: TradexAPI Root Folder\Release

# *CSharpLib.dll*

**This C# dll offers the interface to use Tradex API.**

This dll contains one class and one interface:

## 1. Tradexapi Class

Functions:

1. **Tradexapi(ITradeAPIListener listener, string orderIDPrefix, int orderIDSeed)**

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| ITradeAPIListener listener | no | An instance of a user-created class that inherits ITradeAPIListener interface. This is the callback object that will be used by TradexAPI to push history messages and latest updates to your User Program. |
| string orderIDPrefix | no | The prefix of orderID for each of your order submitted through TradexAPI. |
| int orderIDSeed | no | The very first OrderID for your orders submitted through TradexAPI. |

**Example**: if you pass “FT” as **orderIDPrefix** and 100 as **orderIDSeed**. Then the orderID of the first order submitted through TradexAPI will be “FT100”. The orderID of the next order will be “FT101”.

2. `void ConnectTrade(string ipAddress, int port)`

Utilize this function to connect to Tradex instance's Trade Host.

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `string` ipAddress | no | IP address on where the Tradex instance is running. When you are running your User program on the same machine with Tradex instance, simply give "127.0.0.1" |
| int port | no | The Trade port of our TradexAPI. By default, it's 55241 |

### 3. `void ConnectMarket(string ipAddress, int port)`

Utilize this function to connect to Tradex instance's Market Host.

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `string` ipAddress | no | IP address on where the Tradex instance is running. When you are running your User program on the same machine with Tradex instance, simply give "127.0.0.1" |
| int port | no | The Market port of our TradexAPI. By default, it's 55242 |

4. `ReturnedStatus AddOrder(string traderAccount, string exchangeName, string contractName, string contractDate, OrderTypes orderType, OrderSides buyOrSell, int lots, string price1, string price2, OrderOpenOrClose OC, PositionType hedgeFlag, Dictionary<string, string> options, out string clientOrderID)`

This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

This function is used to submit a new order to Tradex instance through Tradex API.

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `string traderAccount` | yes | The trader account that will be linked with this new order.<br><br>You can pass an empty string as the traderAccount, if you do so, Tradex API will link your first trade account with the new order. |
| `string exchangeName` | no | Exchange name of your contract. For example **SHFE** or **SGX**. |
| `string contractName` | no | Contract name of your contract. For example **ag** or **ZUC.** |
| `string contractDate` | no | Contract date of your contract. For example **201806**. |
| `OrderTypes orderType` | no | The order type of your order. Currently, we support 4 types: **Market**, **Limit**, **Stop**, and **StopLimit**.<br><br>Please note that although Tradex API supports these 4 types of orders, your order may still be rejected by the exchange if the order type is not supported by that exchange. |
| `OrderSides buyOrSell` | no | Stands for the side of your order. Only have two possible value: **Buy** or **Sell**. |
| `int lots` | no | The quantity of your order. |
| `string price1` | yes | The price of your order.<br><br>Please note you could set **price1** as null only when your order type is **Market**. |
| `string price2` | yes | The price of your **StopLimit** order after the market price becomes worse than **price1**.<br><br>Please note you have to give **price2** when your order type is **StopLimit**. |
| `OrderOpenOrClose OC` | no | OpenClose flag for your order.<br><br>Currently, we support 4 types of OpenClose flag: **Open**, **Close**, **CloseToday**, and **Auto**. |
| `PositionType hedgeFlag` | no | Hedge flag for your order.<br><br>Currently, we support 3 types of hedge flag: **Speculate**, **Hedge**, and **Arbitrage**. |
| `Dictionary<string, string> options` | yes | Simply pass null as its value if no special tags are going to be sent.<br><br>Currently enabled tags:<br>1. C-Strategy |

| | | |
|---|---|---|
| `out string clientOrderID` | no | Upon successful calling of this function, **clientOrderID** will be your new submitted orderID. |

**Note**: the `ReturnedStatus` returned by this function as well as all later introduced functions indicate whether the request has been sent to Tradex instance through TradexAPI. It **does not** ensure that this request will be 100% sent to and handled by brokers or exchanges.

`ReturnedStatus.IsSuccess == true` indicates successfully sent to Tradex instance.
`ReturnedStatus.IsSuccess == false` indecates failed to send to Tradex instance.

5. `ReturnedStatus AddOrder(List<String> traderAccounts, string exchangeName, string contractName, string contractDate, OrderTypes orderType, OrderSides buyOrSell, int lots, string price1, string price2, List<OrderOpenOrClose> OCs, List<PositionType> hedgeFlags, Dictionary<string, string> options, out string clientOrderID)`

   This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

   This function is used to submit a new Strategy order to Tradex instance through Tradex API.

   参数解释:

| 参数 | 可为空 | 解释 |
|---|---|---|
| `List<String> traderAccounts` | yes | The trader accounts that will be linked with every leg order of this strategy order.<br><br>The first trader accounnt string in this list will be applied to the leg order whose ID=1 inside the configuration of its Strategy.<br><br>You can pass an empty string as the traderAccount, if you do so, Tradex API will link your first trade account with the new order. |
| `string exchangeName` | no | Exchange name of your contract. For example **SHFE** or **SGX**. |
| `string contractName` | no | Contract name of your contract. For example **ag** or **ZUC.** |
| `string contractDate` | no | Contract date of your contract. For example **201806**. |
| `OrderTypes orderType` | no | The order type of your order. Currently, we support 4 types: **Market**, **Limit**, **Stop**, and **StopLimit**.<br><br>Please note that although Tradex API supports these 4 types of orders, your order may still be rejected by the exchange if the order type is not supported by that exchange. |
| `OrderSides buyOrSell` | no | Stands for the side of your order. Only have two possible value: **Buy** or **Sell**. |
| `int lots` | no | The quantity of your order. |
| `string price1` | yes | The price of your order.<br><br>Please note you could set **price1** as null only when your order type is **Market**. |
| `string price2` | yes | The price of your **StopLimit** order after the market price becomes worse than **price1**.<br><br>Please note you have to give **price2** when your order type is **StopLimit**. |
| `List<OrderOpenOrClose> OCs` | no | OpenClose flags for every leg order of this strategy order.<br><br>The first `OrderOpenOrClose` flag in this list will be applied to the leg order whose ID=1 inside the configuration of its Strategy. |

| | | |
|---|---|---|
| | | Currently, we support 4 types of OpenClose flag: **Open**, **Close**, **CloseToday**, and **Auto**. |
| `List<PositionType> hedgeFlag` | no | Hedge flags for every leg order of this strategy order.<br><br>The first `PositionType` flag in this list will be applied to the leg order whose ID=1 inside the configuration of its Strategy.<br><br>Currently, we support 3 types of hedge flag: **Speculate**, **Hedge**, and **Arbitrage**. |
| `Dictionary<string, string> options` | yes | Simply pass `null` as its value if no special tags are going to be sent.<br><br>Currently enabled tags:<br>1. C-Strategy |
| `out string clientOrderID` | no | Upon successful calling of this function, **clientOrderID** will be your new submitted orderID. |

**Note**: the `ReturnedStatus` returned by this function as well as all later introduced functions indicate whether the request has been sent to Tradex instance through TradexAPI. It **does not** ensure that this request will be 100% sent to and handled by brokers or exchanges.

`ReturnedStatus.IsSuccess == true` indicates successfully sent to Tradex instance.
`ReturnedStatus.IsSuccess == false` indecates failed to send to Tradex instance.

This function is used only for submission of Strategy order.

6. `ReturnedStatus AmendOrder(String clientOrderID, String traderAccount, OrderTypes orderType, int lots, String price1, String price2)`

This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

This function should be called when you want to amend your submitted order's info (including lots, price1, or price2).

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `String clientOrderID` | no | The **clientOrderID** of your target order. |
| `string traderAccount` | yes | The trader account that your target order is linked with.<br><br>It is recommended to pass an empty string as **traderAccount** to avoid confusion. This is because you are not able to amend the traderAccount of a submitted order. |
| `OrderTypes orderType` | no | The order type of your target order.<br><br>You are recommended to pass 0 as **orderType** here to avoid confusion. This is because you are not able to amend the order type of a submitted order. |
| `string price1` | no | The new **price1** for your submitted order. |
| `string price2` | yes | The new **price2** for your submitted **StopLimit** order.<br><br>If your order type is not **StopLimit**, simply give null for **price2**. |

### 7. `ReturnedStatus CancelOrder(String clientOrderID)`

This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

This function should be called when you want to cancel a submitted order.

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `String clientOrderID` | no | The **clientOrderID** of your target order. |

8. `ReturnedStatus SubscribePrice(String exchangeName, String contractName, String contractDate, bool DOM)`

This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

This function should be called when you want to get latest price update of a certain contract.

Upon successfully calling of this function, you will continuously get price updates of the target contract (if there a new price update for that contract).

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `String` exchangeName | no | The name of the exchange your contract belongs to. |
| `String` contractName | no | The name of the target contract. |
| `String` contractDate | no | The date of the target contract. |
| `bool` DOM | no | Not used.<br>Simply give `true` to avoid confusion. |

9. `ReturnedStatus UnsubscribePrice(String exchangeName, String contractName, String contractDate, bool DOM)`

This function should be called only after your **listener** class (to be introduced in next section) have called the OnReady() function.

This function should be called when you want to block latest price update of a certain contract.

Parameter explanation:

| Parameter | Nullable | Detail |
|---|---|---|
| `String` exchangeName | no | The name of the exchange your contract belongs to. |
| `String` contractName | no | The name of the target contract. |
| `String` contractDate | no | The date of the target contract. |
| `bool` DOM | no | Not used.<br>Simply give `true` to avoid confusion. |

**Note**: The mechanism of SubscribePrice(…) and UnsubscribePrice(…) works differently: One can call both functions for the same contract for multiple times. **Lets say you called the SubscribePrice(…) for SHFE/al/201808 for 3 times. Then you will also need to call UnsubscribePrice(…) for SHFE/al/201808 for 3 times if you want to stop getting price update of SHFE/al/201808.**

The reason why we do this is to allow multi-user programs to use TradexAPI at the same time. Thus, it is not good that one user program will stop getting price update of a contract if another user program calls UnsubscribePrice(…) for that contract.

## 2. ITradeAPIListener interface

User program should first create a class which inherits this interface.

User program should then use the created class to instantiate an object and use that object in the instantiation of the `Tradexapi` object (You will use this `Tradexapi` object to connect to Trade Host or Market Host and perform trading operations as introduced in the above session).

```
// Create listener instance
ApiListener listener = new ApiListener();
string clientOrderID;

// Create API wrapper instance and connect to client
Tradexapi api = new Tradexapi(listener, DateTime.Now.ToString("yyyyMMddHHmmss"), 0);
```

Take the above code as an example: here we defined a class `ApiListener` which inherits the `ITradeAPIListener` interface. We first created an instance of this `ApiListener` class and then used this instance to instantiate a new `Tradexapi` object.

---

This interface consists of many CallBack functions which will be automatically called by `Tradexapi` object during runtime.

In the definition of each callback function of your ITradeAPIListener-inherited class, you should decide what to do with these updates (PriceUpdate, OrderUpdate, PositionUpdate, FillUpdate) that you received.

---

Functions:

1. **OnFillReceived(ITradeAPI tradeAPI, pbFillInfo fillupdate)**

   This function will be automatically called by the `Tradexapi` object if you have successfully connected to Trade Host and there is really a Fill Update that needs to be pushed to you.

   Parameter explanation:

| Parameter | Detail |
|---|---|
| `ITradeAPI` tradeAPI | The `Tradexapi` object that you instantiated. |
| `pbFillInfo` fillupdate | The fill information pushed to you by Tradex API. |

2. **OnHostConnected (ITradeAPI tradeAPI)**

   This function will be automatically called by the `Tradexapi` object if you have successfully connected to Trade Host.

   It is recommended that you call functions including `AddOrder(……)`, `AmendOrder(……)`, `CancelOrder(……)` only after this function has been called by TradexAPI.

   Parameter explanation:

| Parameter | Detail |
|---|---|
| `ITradeAPI` tradeAPI | The `Tradexapi` object that you instantiated. |

3. **OnHostDisconnected (ITradeAPI tradeAPI)**

This function will be automatically called by the `Tradexapi` object if you have disconnected from Trade Host.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The `Tradexapi` object that you instantiated. |

4. **OnOrderUpdated (ITradeAPI tradeAPI, pbOrderInfo orderUpdate)**

This function will be automatically called by the `Tradexapi` object if you have successfully connected to Trade Host and there is an Order Update that needs to be pushed to you.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The `Tradexapi` object that you instantiated. |
| pbOrderInfo orderUpdate | The order information pushed to you by Tradex API. |

5. **OnPositionUpdated (ITradeAPI tradeAPI, List<pbPositionInfo> positions)**

This function will be automatically called by the `Tradexapi` object if you have successfully connected to Trade Host and there is a Position Update that needs to be pushed to you.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The `Tradexapi` object that you instantiated. |
| List<pbPositionInfo> positions | The position information list pushed to you by Tradex API. |

6. **OnPriceConnected (ITradeAPI tradeAPI)**

This function will be automatically called by the `Tradexapi` object if you have successfully connected to Market Host.

You may call functions including `SubscribePrice(……)`, `UnsubscribePrice(……)` only after this function has been called by TradexAPI.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The `Tradexapi` object that you instantiated. |

7. **OnPriceDisconnected (ITradeAPI tradeAPI)**

This function will be automatically called by the Tradexapi object if you have disconnected from Market Host.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The Tradexapi object that you instantiated. |

8. **OnPriceUpdated (ITradeAPI tradeAPI, pbPriceInfo priceUpdate)**

This function will be automatically called by the Tradexapi object if you have successfully connected to Market Host, you have subscribed to the price of a certain contract and there is a Price Update that needs to be pushed to you.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The Tradexapi object that you instantiated. |
| pbPriceInfo priceUpdate | The price update pushed to you by Tradex API. |

9. **OnReady (ITradeAPI tradeAPI)**

This function will be automatically called by the Tradexapi object if all history data has been pushed to you and the API now is ready to accept new requests.

History data (includes OrderUpdate and FillReceived) and PositionUpdate will be pushed to you everytime when you have successfully connected to Trade Host.

It is always recommended that you only start submitting new requests through API only when this function has been called.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The Tradexapi object that you instantiated. |

10. **OnTraderAccountAdded (ITradeAPI tradeAPI, List<pbAccountInfo> accounts)**

This function will be automatically called by the Tradexapi object if you have successfully connected with Trade Host and there is Account Update that needs to be pushed to you.

Parameter explanation:

| Parameter | Detail |
|---|---|
| ITradeAPI tradeAPI | The Tradexapi object that you instantiated. |
| List<pbAccountInfo> accounts | The latest account information list pushed to you by Tradex API. |

# *Explanation for trade update classes:*

**Note**: For the classes below, DO NOT use any field that is not included in the explanation forms.

## 1. pbAccountInfo Class

Fields explanation:

| Field | Explanation |
|---|---|
| double Balance | The balance of your trader account. |
| double Cash | The available funds of your trader account. |
| double ClosedPL | Closed profit/loss. |
| double UnPL | Floating profit/loss. |
| double UsedMargin | Used margin. |
| string TraderAccount | Trader account. |

## 2. pbFillInfo Class

Fields explanation:

| Field | Explanation |
|---|---|
| `string FillID` | Fill ID. |
| `string ClientOrderID` | Related order ID of this fill info. |
| `string ContractDate` | The date of this contract. |
| `string ContractName` | The name of this contract. |
| `string ExchangeName` | The name of this contract's exchange. |
| `DateTime FilledTime` | The time when the TMS Server received this fill info, not the real fill time. |
| `int Lots` | The lots of this fill. |
| `string Price` | The price of this fill. |
| `OrderSides Side` | Is the order's side **Buy** or **Sell.** |
| `string TraderAccount` | The trader account that this fill is related to. |

## 3. pbPositionInfo Class

Fields explanation:

| Field | Explanation |
|---|---|
| `string ContractDate` | The date of this contract. |
| `string ContractName` | The name of this contract. |
| `string ExchangeName` | The name of this contract's exchange. |
| `PositionType HedgeFlag` | The hedge flag of your position. |
| `double BuyAveragePrice` | The average buy price of this position. |
| `double SellAveragePrice` | The average sells price of this position. |
| `int Buys` | The total buys lots of this position. |
| `int Sells` | The total sells lots of this position. |
| `int PrevBuys` | The total previous buy lots of this position.（Buys-DayBuys = PrevBuys）. |
| `int PrevSells` | The total previous sell lots of this position.（Sells-DaySells = PrevSells）. |
| `double ClosedPL` | Closed profit/loss. |
| `string TraderAccount` | Trader account. |

## 4. pbOrderInfo Class

Fields explanation:

| Field | Explanation |
|---|---|
| `string ClientOrderID` | Order ID. |
| `string ContractDate` | The date of this contract. |
| `string ContractName` | The name of this contract. |
| `string ExchangeName` | The name of this contract's exchange. |
| `string TraderAccount` | Trader account. |
| `OrderSides Side` | Is the order's side **Buy** or **Sell.** |
| `OrderStates Status` | The status of this order（9 possible values）<br>1. **Submitted**[this order is being sent from Tradex to TMS Server].<br>2. **Queued**[this order has been received by TMS Server, is going to be sent to upstream].<br>3. **Sent**[this order has been sent to upstream].<br>4. **Working**[this order is working].<br>5. **AmendPending**[this order is now being amended].<br>6. **CancelPending**[the order is now being canceled].<br>7. **Filled**[this order has been totally filled].<br>8. **Cancelled**[this order has been canceled].<br>9. **Rejected**[this order is rejected]. |
| `int Lots` | The quantity of your order. |
| `string Price1` | The **price1** for your submitted order. |
| `string Price2` | The **price2** for your submitted **StopLimit** order. |
| `int AmountFilled` | The amount of already filled lots. |
| `string AveragePrice` | The average price for thoes filled lots of your order. |
| `OrderTypes Type` | Type of your order<br>1. Market<br>2. Limit<br>3. Stop<br>4. StopLimit |