OPENING AND CLOSING TRADES USING STREAMING

In this tutorial you will learn how to use streaming to open, store and close opened positions. The code will be written in C#, but it is not complicated and could be easily rewritten to any other language.

The key of this tutorial is to understand the trade records sent in streaming and to interpret them properly. The whole code used in this tutorial is available here:

DOWNLOAD CODE (http://developers.xstore.pro/public/files/tutorials/trading_streaming_code.zip)

Commands used in this tutorial:

- getSymbol,
- tradeTransaction,
- getTradeStatus(streaming),
- getTrades(streaming),
- getTickPrices(streaming).

Before you start...

- To complete this tutorial, you need C# wrapper (available here: developers.xstore.pro/api/wrappers (http://developers.xstore.pro/api/wrappers.html)) or be able to use all commands listed above by yourself,
- We assume that you have at least basic knowledge of xAPI and you are able to establish a connection and login to the server.

Step 1. Get initial symbol price

1. Pick a symbol you wish to trade on, in our case it will be EURUSD
2. Use `getSymbol` command to get initial *Ask* and *Bid* prices for the symbol.

```
SymbolResponse symbolResponese = APICommandFactory.ExecuteSymbolCommand(connector,
symbol);
currentAskPrice = symbolResponese.Symbol.Value.Ask;
currentBidPrice = symbolResponese.Symbol.Value.Bid;
```

The *symbol* variable is used to store the name of the symbol ("EURUSD"). The *currentAskPrice* and *currentBidPrice* are used to store current prices for the symbol.

Step 2. Create streaming listener class

1. Create streaming listener class, which will implement the `StreamingListener` interface. You can call it `TestStreamingListener`.
2. Define `ReceiveTradeRecord` function with the code listed below, which will take care of `StreamingTradeRecords` that come through streaming.

```
void StreamingListener.ReceiveTradeRecord(StreamingTradeRecord tradeRecord)
{
    Console.WriteLine("\n"+tradeRecord+"\n");
    if (tradeRecord.Type == STREAMING_TRADE_TYPE.OPEN)
    {
        Log("Trade with positionId: " + tradeRecord.Position + " opened");
        acceptedTradeOrders.Remove(tradeRecord.Order2.Value);
        openedPositions.Add(tradeRecord.Position.Value);
    }
    else if (tradeRecord.Type == STREAMING_TRADE_TYPE.CLOSE)
    {
        Log("Trade with positionId: " + tradeRecord.Position + " closed");
        openedPositions.Remove(tradeRecord.Position.Value);
    }
}
```

The `ReceiveTradeRecord` method checks the *type* field in the *tradeRecord*. Depending on its value, it adds or removes order and position numbers from appriopriate lists.

Every position is opened and closed by an order. Notice that the *acceptedTradeOrders* list holds numbers of the orders that open positions. When the field *type* has `STREAMING_TRADE_TYPE.OPEN` value we know that the position was opened. Then we remove the order that opened the position from *acceptedTradeOrders* using the *order2* field and add the position number to the *openedPositions* list using the *position* field.

In case the *type* has the `STREAMING_TRADE_TYPE.CLOSE` value we can remove the position from *openedPositions* list as the position is closed.

3. Define `ReceiveTickRecord` function which will get current symbol prices in real time and store them in `currentAskPrice` and `currentBidPrice` variables described in the Step 1. The code is listed below:

```
void StreamingListener.ReceiveTickRecord(StreamingTickRecord tickRecord)
{
    currentAskPrice = tickRecord.Ask.Value;
    currentBidPrice = tickRecord.Bid.Value;
}
```

4. Define the `ReceiveTradeStatusRecord` function which adds accepted trade orders which open and close positions to the *acceptedTradeOrders* list. The *unacceptedTradeOrders* list holds the trade orders that open and close the positions, but have not been unaccepted yet. The method removes the order from *unacceptedTradeOrders* in case it was accepted.

```
void StreamingListener.ReceiveTradeStatusRecord(StreamingTradeStatusRecord
tradeStatusRecord)
{
    Console.WriteLine("\n"+tradeStatusRecord.ToString()+"\n");
    if (tradeStatusRecord.RequestStatus == REQUEST_STATUS.ACCEPTED)
    {
        Log("Trade request with order: " + tradeStatusRecord.Order + " accepted");
        unacceptedTradeOrders.Remove(tradeStatusRecord.Order.Value);
        acceptedTradeOrders.Add(tradeStatusRecord.Order.Value);
    }
}
```

5. For the application to build without errors, you also need to define `ReceiveBalanceRecord` and `ReceiveProfitRecord` in your class. We won't use them in this tutorial so we will just leave them blank.
   The full code of `TestStreamingListener` class is listed below:

Expand Code

Step 3. Initialize streaming

1. Now we can use our `TestStreamingListener` class to initialize the streaming. Here is the code:

```
TestStreamingListener strListener = new TestStreamingListener();
connector.Streaming.Connect(strListener);
connector.Streaming.SubscribeTrades();
connector.Streaming.SubscribePrice(symbol);
connector.Streaming.SubscribeTradeStatus();
```

Step 4. Open the trade

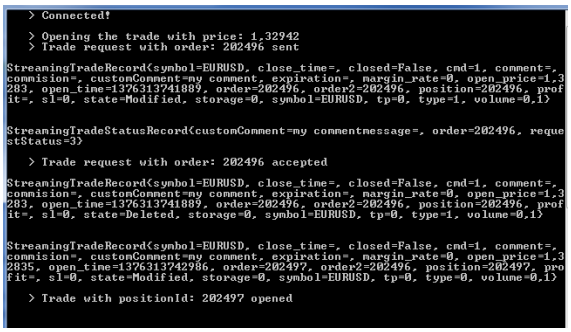
1. The opening of the trade must be done by `tradeTransaction` command. Here is the code:

```
double sl = 0.0;
double tp = 0.0;
double volume = 0.1;
long order = 0;
string customComment = "my comment";
long expiration = 0;
TradeTransactionResponse tradeResponse =
APICommandFactory.ExecuteTradeTransactionCommand(connector,
TRADE_OPERATION_CODE.SELL, TRADE_TRANSACTION_TYPE.ORDER_OPEN, currentBidPrice, sl,
tp, symbol, volume, order, customComment, expiration);
unacceptedTradeOrders.Add(tradeResponse.Order.Value);
```

   We open the *SELL* trade using the *currentBidPrice* and *symbol* variables which were explained in the beginning.

Additionally, we add the *order* number to the *unacceptedTradeOrders* and wait for it to be accepted via streaming in the `ReceiveTradeStatusRecord`.

2. This is the output of the application in the console:



(public/img/tutorials/trading_streaming/console1.png)

We can see that there are three `StreamingTradeRecords` during opening the position:

- two for the order (number 202496) that opens the position with *type* = 1
- one for the opened (*type* = 0) position (number 202497) with the number of the order that opened the position (202496) under *order2* field.

Step 5. Close the trade

1. Closing of the trade is also executed by the `tradeTransaction` command, but in this case we use the Ask price and add the position number we are willing to close. We can get the position number from *openedPositions* list.

```
double sl = 0.0;

double tp = 0.0;

double volume = 0.1;

string customComment = "my comment";

long expiration = 0;

long positionToClose = openedPositions[openedPositions.Count - 1];

TradeTransactionResponse tradeResponse =

APICommandFactory.ExecuteTradeTransactionCommand(connector,

TRADE_OPERATION_CODE.SELL, TRADE_TRANSACTION_TYPE.ORDER_CLOSE, currentAskPrice,

sl, tp, symbol, volume, positionToClose, customComment, expiration);
```

2. The output of the console after closing the position is displayed on the image below:

(public/img/tutorials/trading_streaming/console2.png)

Now, we can see that there are four `StreamingTradeRecords` for the closing of the position:

- two for the order (number 202503) that closes the position (but this time it is a BUY order, which can be seen in the *cmd* field) with *type* = 1
- additional one with the info about the position

    - last one which holds the information that the position (number 202497) was closed (we know this because: *type* = 2 and *closed* = true).

Summary

In this tutorial you have learned how to:

- open the trade,
- get current prices for the symbol using streaming,
- get the information about accepted orders that open the position using streaming,
- get the information about opened positions using streaming,
- close the position using prices from streaming.