FIX Protocol with C++ (FXCM Sample Program)

FIX API links:

General information:

https://www.investopedia.com/terms/f/financial-information-exchange.asp

https://www.fixtrading.org/implementation-guide

FXCM Github page about FIX:

https://github.com/fxcm/fixapi

Example projects:

https://github.com/fxcm/FIXAPI/tree/master/Sample%20Projects

FIX Specification – please always refer to this document, if you need details about fields, tags and values

https://apiwiki.fxcorporate.com/api/fix/docs/FXCM-FIX-BSI.pdf

FXCM data dictionary that contains FXCM unique tag starting from 9000:

https://apiwiki.fxcorporate.com/api/fix/docs/FIXFXCM10.xml

CFD Product guide:

https://docs.fxcorporate.com/user-guide/ug-cfd-product-guide-ltd-en.pdf

For C++ we recommend using Visual Stidio by Microsoft:

https://visualstudio.microsoft.com/downloads/

Our sample code is available here:

https://github.com/fxcm/FIXAPI/blob/master/Sample%20Projects/fix_example_x64.7z

Configuration, connection and a few quick tests

For using FIX you need to get credentials, which are issued by FXCM – separate for trading session and for market data session.

When you download the example code you don't need to install anything – it is ready to use after unzipping.

You can open by double clicking on the solution file (fix_example.sln) and choose Visual Studio as default opening program if prompted.

Before connecting you need to update the configuration settings, which are in 'settings.cfg' file. What you change here is inside the yellow box in the screenshot below. Basically it's just your username and password plus some parameters that you get from your FXCM FIX credentials.

```
25  PrintIncoming=Y
26  PrintOutgoing=Y
27  PrintEvents=Y
28  IgnorePossDupResendRequests=Y
29  TargetSubID=EUDEMO
30  Username=TestUser
31  Password=[redacted]
32
33
34  [SESSION]
35  SocketConnectHost=fixdemo.fxcorporate.com
36  SocketConnectPort=8043
37  BeginString=FIX.4.4
38  SenderCompID=testuser_client1
39  TargetCompID=FXCM
40  MDEntryType=Y
41
42  [SESSION]
43  SocketConnectHost=fixdemo.fxcorporate.com
44  SocketConnectPort=9043
45  BeginString=FIX.4.4
46  SenderCompID=MD_testuser_client1
47  TargetCompID=FXCM
48  MDEntryType=Y
49
```

If everything is entered correctly, you should be able to login and see some reports related to your account. In order to do this, you need to choose 'Debug' menu then 'Start Debugging' or simply press F5 key.

The program is built, which can take a few moments and then a terminal window should appear.

```
Session -> created
Session -> created
Session -> logon
Session -> logon
TradingSessionStatus -> TradSesStatus -2
  SecurityList via TradingSessionStatus ->
    Symbol -> ACA.fr
    Symbol -> JD.hk
    Symbol -> EUR/CAD
    Symbol -> USD/ZAR
    Symbol -> PING.hk
    Symbol -> UBER.us
    Symbol -> SMIC.hk
    Symbol -> QAN.au
    Symbol -> BEKE.us
    Symbol -> EUR/TRY
    Symbol -> BNGO.us
    Symbol -> PDD.us
    Symbol -> AMZN.us
    Symbol -> COIN.us
    Symbol -> NVDA.us
    Symbol -> NAS100
    Symbol -> DIS.us
    Symbol -> MCD.us
    Symbol -> VOLX
    Symbol -> ZM.us
```

As you can see it starts with creating sessions (Trading and Market data), logon for both, then trading session status check (value 2 meaning open, while 3 would mean closed) and after this the symbols you are subscribed for in Trading Station.

Then you can do a few quick tests of the functions that you see in main.cpp plus file:

- Get open positions by entering 1 in the terminal:

```
1
RequestForPositionsAck -> PosReqID - 8
PositionReport ->
   Account -> 01260808
   Symbol -> EUR/USD
   PositionID -> 175606221
   Open Time -> 20210901-16:41:47.000
```

- Subscribe for real-time market data updates by entering 2

- Unsubscribe with 3

```
2
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.1844 Ask - 1.18454
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.1844 Ask - 1.18453
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18439 Ask - 1.18453
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18439 Ask - 1.18452
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18438 Ask - 1.18451
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18438 Ask - 1.18449
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18439 Ask - 1.18449
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18438 Ask - 1.18449
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18438 Ask - 1.1845
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18439 Ask - 1.18451
MarketDataSnapshotFullRefresh -> Symbol - EUR/USD Bid - 1.18439 Ask - 1.18452
3
```

- 4 is for Market order that is executed immediately. In the example it is for buying 10K EUR/USD, and it would be executed for each account associated with this username.

```
4
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380215197
  LastQty -> 0
  CumQty -> 0
  ExecType -> 0
  OrdStatus -> 0
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380215197
  LastQty -> 10000
  CumQty -> 10000
  ExecType -> F
  OrdStatus -> 7
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380215197
  LastQty -> 10000
  CumQty -> 10000
  ExecType -> F
  OrdStatus -> 2
```

- 5 – getting closed positions:

```
5
RequestForPositionsAck -> PosReqID - 6
PositionReport ->
   Account -> 01260808
   Symbol -> EUR/USD
   PositionID -> 175691156
   Open Time -> 20210903-20:15:23.000
PositionReport ->
   Account -> 01260808
   Symbol -> EUR/USD
   PositionID -> 175691132
   Open Time -> 20210903-20:14:50.000
PositionReport ->
   Account -> 01260808
   Symbol -> EUR/USD
   PositionID -> 175680076
   Open Time -> 20210906-03:28:50.000
```

- Creating stop order with 6 and limit order with 7:

```
6
ExecutionReport ->
  ClOrdID -> 5
  Account -> 01260808
  OrderID -> 380450954
  LastQty -> 0
  CumQty -> 0
  ExecType -> 0
  OrdStatus -> 0
7
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380450957
  LastQty -> 0
  CumQty -> 0
  ExecType -> 0
  OrdStatus -> 0
```

- Checking waiting orders that have yet to be filled with 8:

```
8
ExecutionReport ->
  ClOrdID -> 5
  Account -> 01260808
  OrderID -> 380450954
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380450957
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
```

- Canceling order with 9 and entering order ID:

```
9
Enter Order ID to be canceled: 380450957
ExecutionReport ->
  ClOrdID -> 8
  Account -> 01260808
  OrderID -> 380450957
  LastQty -> 0
  CumQty -> 0
  ExecType -> 6
  OrdStatus -> 6
ExecutionReport ->
  ClOrdID -> 8
  Account -> 01260808
  OrderID -> 380450957
  LastQty -> 0
  CumQty -> 0
  ExecType -> 4
  OrdStatus -> 4
8
ExecutionReport ->
  ClOrdID -> 5
  Account -> 01260808
  OrderID -> 380450954
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
```

More details about the code

All the functions available in ‘main.cpp’ are defined in ‘fix_application.cpp’.

The first one that is called upon starting the program is StartSession(). It is where your configuration settings are loaded, validated and login is initiated.

```cpp
// Starts the FIX session. Throws FIX::ConfigError exception if our configuration settings
// do not pass validation required to construct SessionSettings
void FixApplication::StartSession()
{
    try{
        settings      = new SessionSettings("settings.cfg");
        store_factory = new FileStoreFactory(* settings);
        log_factory   = new FileLogFactory(* settings);
        initiator     = new SocketInitiator(* this, * store_factory, * settings, * log_factory/*Optional*/);
        initiator->start();
    }catch(ConfigError error){
        cout << error.what() << endl;
    }
}
```

And of course once you decide to end your session by entering 0 in the console EndSession() function is called

```cpp
// Logout and end session
void FixApplication::EndSession()
{
    initiator->stop();
    delete initiator;
    delete settings;
    delete store_factory;
    delete log_factory;
}
```

Getting current open positions – function is GetPositions() and it requests info about open positions for all accounts associated with the login.

```
// Sends RequestForPositions which will return PositionReport messages if positions
// matching the requested criteria exist; otherwise, a RequestForPositionsAck will be
// sent with the acknowledgement that no positions exist. In our example, we request
// positions for all accounts under our login
void FixApplication::GetPositions(int PosReqType_)
{
    // Here we will get positions for each account under our login. To do this,
    // we will send a RequestForPositions message that contains the accountID
    // associated with our request. For each account in our list, we send
    // RequestForPositions.
    int total_accounts = (int)list_accountID.size();
    for(int i = 0; i < total_accounts; i++){
        string accountID = list_accountID.at(i);
        // Set default fields
        FIX44::RequestForPositions request;
        request.setField(PosReqID(NextRequestID()));
        request.setField(PosReqType(PosReqType_));
        // AccountID for the request. This must be set for routing purposes. We must
        // also set the Parties AccountID field in the NoPartySubIDs group
        request.setField(Account(accountID));
        request.setField(SubscriptionRequestType(SubscriptionRequestType_SNAPSHOT));
        request.setField(AccountType(
            AccountType_ACCOUNT_IS_CARRIED_ON_NON_CUSTOMER_SIDE_OF_BOOKS_AND_IS_CROSS_MARGINED));
        request.setField(TransactTime());
        request.setField(ClearingBusinessDate());
        request.setField(TradingSessionID("FXCM"));
        // Set NoPartyIDs group. These values are always as seen below
        request.setField(NoPartyIDs(1));
        FIX44::RequestForPositions::NoPartyIDs parties_group;
        parties_group.setField(PartyID("FXCM ID"));
        parties_group.setField(PartyIDSource('D'));
        parties_group.setField(PartyRole(3));
        parties_group.setField(NoPartySubIDs(1));
        // Set NoPartySubIDs group
        FIX44::RequestForPositions::NoPartyIDs::NoPartySubIDs sub_parties;
        sub_parties.setField(PartySubIDType(PartySubIDType_SECURITIES_ACCOUNT_NUMBER));
        // Set Parties AccountID
        sub_parties.setField(PartySubID(accountID));
        // Add NoPartySubIds group
        parties_group.addGroup(sub_parties);
        // Add NoPartyIDs group
        request.addGroup(parties_group);
        // Send request
        Session::sendToTarget(request, sessionID(false));
    }
```

A few fields to mention here:

- PosReqType – if the set value is PosReqType_POSITIONS (which is 0), it would request open positions only. For closed positions it should have value of 1, or in the example it would be PosReqType_TRADES. Here are all options:

```
const int PosReqType_POSITIONS = 0;
const int PosReqType_BACKOUT_MESSAGE = 5;
const int PosReqType_DELTA_POSITIONS = 6;
const int PosReqType_SETTLEMENT_ACTIVITY = 4;
const int PosReqType_TRADES = 1;
const int PosReqType_EXERCISES = 2;
const int PosReqType_ASSIGNMENTS = 3;
```

- SubscriptionRequestType – except SubscriptionRequestType_SNAPSHOT for which the value is ‘0’ (as a char and not integer), other options are SubscriptionRequestType_SNAPSHOT (‘1’) and SubscriptionRequestType_DISABLE_PREVIOUS_SNAPSHOT_PLUS_UPDATE_REQUEST (‘2’)
- AccountType – please don’t change its current value
- ClearingBusinessDate – please don’t set any value here

Getting closed positions – option 5 in main.cpp file is pretty similar to getting open positions. It is using GetPositions() function again with just PostReqType set to PosReqType_TRADES instead of PosReqType_POSITIONS

```
24          switch(command){
25          case 0: // Exit example application
26              exit = true;
27              break;
28          case 1: // Get positions
29              app.GetPositions(PosReqType_POSITIONS);
30              break;
31          case 2: // Subscribe to market data
32              app.SubscribeMarketData();
33              break;
34          case 3: // Unsubscribe to market data
35              app.UnsubscribeMarketData();
36              break;
37          case 4: // Send market order
38              app.MarketOrder();
39              break;
40          case 5: // Get closed positions
41              app.GetPositions(PosReqType_TRADES);
42              break;
43          case 6: // Stop order
44              app.StopOrder();
```

You can also check some more fields related to closed positions request on page 52 of our FIX specification manual.

Subscribing and unsubscribing for market data – as you can see in our example it’s for EUR/USD and USD/CAD, but can be any instrument that you have subscribed for in Trading Station.

- SubscriptionRequestType is the parameter that is making the difference between subscribe and unsubscribe, as you would notice from the snippets below in first case we have SubscriptionRequestType_SNAPSHOT_PLUS_UPDATES, while in second case we have SubscriptionRequestType_DISABLE_PREVIOUS_SNAPSHOT_PLUS_UPDATE_REQUEST Another option is SubscriptionRequestType_SNAPSHOT, which gives just snapshot without subscription.

```cpp

// Subscribes to EUR/USD and USD/CAD trading securities
void FixApplication::SubscribeMarketData()
{
    // Subscribe to market data for EUR/USD and USD/CAD
    string request_ID = "EUR_USD_Request_";
    FIX44::MarketDataRequest request;
    request.setField(MDReqID(request_ID));
    request.setField(SubscriptionRequestType(
        SubscriptionRequestType_SNAPSHOT_PLUS_UPDATES));
    request.setField(MarketDepth(0));
    request.setField(NoRelatedSym(2));

    // Add the NoRelatedSym group to the request with Symbol
    // field set to EUR/USD
    FIX44::MarketDataRequest::NoRelatedSym symbols_group;
    symbols_group.setField(Symbol("EUR/USD"));
    request.addGroup(symbols_group);
    symbols_group.setField(Symbol("USD/CAD"));
    request.addGroup(symbols_group);
    // Add the NoMDEntryTypes group to the request for each MDEntryType
    // that we are subscribing to. This includes Bid, Offer, High, and Low
    FIX44::MarketDataRequest::NoMDEntryTypes entry_types;
    entry_types.setField(MDEntryType(MDEntryType_BID));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_OFFER));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_TRADING_SESSION_HIGH_PRICE));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_TRADING_SESSION_LOW_PRICE));
    request.addGroup(entry_types);

    Session::sendToTarget(request, sessionID(true));
}
```

```cpp

// Unsubscribes from EUR/USD and USD/CAD trading securities
void FixApplication::UnsubscribeMarketData()
{
    // Unsubscribe from EUR/USD and USD/CAD. Note that our request_ID is the exact same
    // that was sent for our request to subscribe. This is necessary to
    // unsubscribe. This request below is identical to our request to subscribe
    // with the exception that SubscriptionRequestType is set to
    // "SubscriptionRequestType_DISABLE_PREVIOUS_SNAPSHOT_PLUS_UPDATE_REQUEST"
    string request_ID = "EUR_USD_Request_";
    FIX44::MarketDataRequest request;
    request.setField(MDReqID(request_ID));
    request.setField(SubscriptionRequestType(
        SubscriptionRequestType_DISABLE_PREVIOUS_SNAPSHOT_PLUS_UPDATE_REQUEST));
    request.setField(MarketDepth(0));
    request.setField(NoRelatedSym(2));

    // Add the NoRelatedSym group to the request with Symbol
    // field set to EUR/USD + USD/CAD
    FIX44::MarketDataRequest::NoRelatedSym symbols_group;
    symbols_group.setField(Symbol("EUR/USD"));
    request.addGroup(symbols_group);
    symbols_group.setField(Symbol("USD/CAD"));
    request.addGroup(symbols_group);

    // Add the NoMDEntryTypes group to the request for each MDEntryType
    // that we are subscribing to. This includes Bid, Offer, High, and Low
    FIX44::MarketDataRequest::NoMDEntryTypes entry_types;
    entry_types.setField(MDEntryType(MDEntryType_BID));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_OFFER));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_TRADING_SESSION_HIGH_PRICE));
    request.addGroup(entry_types);
    entry_types.setField(MDEntryType(MDEntryType_TRADING_SESSION_LOW_PRICE));
    request.addGroup(entry_types);

    Session::sendToTarget(request, sessionID(true));
}
```

- MarketDepth – please leave this one as 0
- NoRelatedSym – this is the number of symbols that we would want to subscribe for
- MDEntryType is about the rates that you want to see.

Sending Market order – in the example you can see a very simple order for buying 10K EUR/USD at current market price.

```
// Sends a basic NewOrderSingle message to buy EUR/USD at the
// current market price
void FixApplication::MarketOrder()
{
    // For each account in our list, send a NewOrderSingle message
    // to buy EUR/USD. What differentiates this message is the
    // accountID
    int total_accounts = (int)list_accountID.size();
    for(int i = 0; i < total_accounts; i++){
        string accountID = list_accountID.at(i);
        FIX44::NewOrderSingle request;
        request.setField(ClOrdID(NextRequestID()));
        request.setField(Account(accountID));
        request.setField(Symbol("EUR/USD"));
        request.setField(TradingSessionID("FXCM"));
        request.setField(TransactTime());
        request.setField(OrderQty(10000));
        request.setField(Side(FIX::Side_BUY));
        request.setField(OrdType(OrdType_MARKET));
        request.setField(TimeInForce(FIX::TimeInForce_GOOD_TILL_CANCEL)); // For newer versions of QuickFIX change this to TimeInForce_GOOD_TILL_CANCEL
        Session::sendToTarget(request, sessionID(false));
    }
}
```

If you want to close existing open position, the way to do this is by MarketOrder() with the same quantity and the opposite side (so Side parameter in the case above would be Side_SELL instead of Side_BUY).

Creating Stop order:

```
void FixApplication::StopOrder()
{
    int total_accounts = (int)list_accountID.size();
    for (int i = 0; i < total_accounts; i++) {
        string accountID = list_accountID.at(i);
        FIX44::NewOrderSingle request;
        request.setField(FIX::ClOrdID(NextRequestID()));
        request.setField(FIX::Account(accountID));
        request.setField(FIX::Symbol("EUR/USD"));
        request.setField(FIX::Side(FIX::Side_BUY));
        request.setField(FIX::TransactTime(FIX::TransactTime()));
        request.setField(FIX::OrderQty(10000));
        request.setField(FIX::OrdType(FIX::OrdType_STOP));
        request.setField(FIX::StopPx(1.1870));

        FIX::Session::sendToTarget(request, sessionID(false));
    }
}
```

This one is pretty similar to MarketOrder() with the only difference in OrdType being OrdType_STOP instead of OrdType_MARKET (value is ‘3’ instead of ‘1’). We also set StopPx (stop price), so buy orders will be filled at or above this price and sell orders will be filled at or below this price.

Limit order, which is also very similar:

```
void FixApplication::LimitOrder()
{
    int total_accounts = (int)list_accountID.size();
    for (int i = 0; i < total_accounts; i++) {
        string accountID = list_accountID.at(i);
        FIX44::NewOrderSingle request;
        request.setField(FIX::ClOrdID(NextRequestID()));
        request.setField(FIX::Account(accountID));
        request.setField(FIX::Symbol("EUR/USD"));
        request.setField(FIX::Side(FIX::Side_BUY));
        request.setField(FIX::TransactTime(FIX::TransactTime()));
        request.setField(FIX::OrderQty(10000));
        request.setField(FIX::OrdType(OrdType_LIMIT));
        request.setField(FIX::Price(1.1860));

        FIX::Session::sendToTarget(request, sessionID(false));
    }
}
```

In this case we have OrdType_LIMIT for OrdType (value is ‘2’), and this time the field related to price is ‘Price’.

GetWaitingOrders() function is for entry orders that have yet to be filled. When we enter 8 as input in the console (for my testing account I have 3 orders for which I can see the order IDs):

```
void FixApplication::GetWaitingOrders()
{
    int total_accounts = (int)list_accountID.size();
    for (int i = 0; i < total_accounts; i++) {
        string accountID = list_accountID.at(i);
        FIX44::OrderMassStatusRequest request;
        request.setField(MassStatusReqID(NextRequestID()));
        request.setField(MassStatusReqType(MassStatusReqType_STATUS_FOR_ALL_ORDERS));
        request.setField(Account(accountID));

        Session::sendToTarget(request, sessionID(false));
    }
}
```

```
8
ExecutionReport ->
  ClOrdID -> 6
  Account -> 01260808
  OrderID -> 380383000
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
ExecutionReport ->
  ClOrdID -> 7
  Account -> 01260808
  OrderID -> 380383001
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
ExecutionReport ->
  ClOrdID -> 8
  Account -> 01260808
  OrderID -> 380383002
  LastQty -> 0
  CumQty -> 0
  ExecType -> I
  OrdStatus -> 0
```

CancelOrder() – here you need to specify Side, Symbol and OrderID, which is a string.

```
void FixApplication::CancelOrder()
{
    int total_accounts = (int)list_accountID.size();
    for (int i = 0; i < total_accounts; i++) {
        string accountID = list_accountID.at(i);
        FIX44::OrderCancelRequest request;
        request.setField(FIX::ClOrdID(NextRequestID()));
        request.setField(FIX::Account(accountID));
        request.setField(FIX::OrigClOrdID());
        request.setField(FIX::Side(FIX::Side_BUY));
        request.setField(FIX::Symbol("EUR/USD"));
        request.setField(FIX::TransactTime(FIX::TransactTime()));
        request.setField(FIX::OrderID("380374833"));

        FIX::Session::sendToTarget(request, sessionID(false));
    }
}
```

Another variant of this, where you input the order ID in the console:

```
void FixApplication::CancelOrder()
{
    int total_accounts = (int)list_accountID.size();
    string order_id;
    cout << "Enter Order ID to be canceled: ";
    cin >> order_id;
    for (int i = 0; i < total_accounts; i++) {
        string accountID = list_accountID.at(i);
        FIX44::OrderCancelRequest request;
        request.setField(FIX::ClOrdID(NextRequestID()));
        request.setField(FIX::Account(accountID));
        request.setField(FIX::OrigClOrdID());
        request.setField(FIX::Side(FIX::Side_BUY));
        request.setField(FIX::Symbol("EUR/USD"));
        request.setField(FIX::TransactTime(FIX::TransactTime()));
        request.setField(FIX::OrderID(order_id));

        FIX::Session::sendToTarget(request, sessionID(false));
    }
}
```

```
9
Enter Order ID to be canceled: 380383000
ExecutionReport ->
  ClOrdID -> 10
  Account -> 01260808
  OrderID -> 380383000
  LastQty -> 0
  CumQty -> 0
  ExecType -> 6
  OrdStatus -> 6
ExecutionReport ->
  ClOrdID -> 10
  Account -> 01260808
  OrderID -> 380383000
  LastQty -> 0
  CumQty -> 0
  ExecType -> 4
  OrdStatus -> 4
```