Using QuickFix

Prof. Dr. Bernd Ulmann

18-OCT-2010

Hochschule fuer Oekonomie und Management, Frankfurt

Introduction

The following talk describes the development of a QuickFIX based interface which was developed for a major financial institution to transmit trade data from in-house systems to Bloomberg employing the FIX Protocol 4.4.

A first trial implementation focused on a pure Perl implementation without a FIX engine at all – although this attempt looked rather promising given the simple structure of the trades to be transmitted, it was decided to use a FIX engine and abandon the pure Perl implementation due to company regulations.

Eventually the application was written in C# which was the result of a company requirement and it turned out that QuickFIX and C# form a good team – the integration is quite seamless and simple from a programmer's perspective.

Quoting from the QuickFIX web site[1]:

> "QuickFIX is a full-featured open source FIX engine,
> currently compatible with the FIX 4.0-5.0 spec. It runs
> on Windows, Linux, Solaris, FreeBSD and Mac OS X.
> APIs are available for C++, .NET, Python and Ruby."

Compiling QuickFIX in a Windows environment is easy – the
distribution kit contains the necessary solution files for various
Visual Studio versions.

Be prepared that compiling QuickFIX takes some time – especially
when working on non-local disks. In our environment a complete
compilation run takes between 15 and 20 minutes.

---

[1]Cf. http://www.quickfixengine.org/

FIX distinguishes between two message classes as described in the talk before:

- Admin messages
- Application messages

The administrative messages deal with connection establishment, heartbeating, connection test, connection terminataion etc. while the application messages transfer actual trade data and the like.

The QuickFIX library takes care of all the administration message logic including sequence numbering etc. so the application programmer can concentrate on the application specific logic without being bothered by FIX specific details.

QuickFix uses an XML description of the FIX standard to be used for communication.

Messages are validated according to this description so in straight forward cases QuickFix will work right out of the box.

If your application needs customer defined fields you will have to modify this XML file to reflect the structure of your particular messages.

An example of such an extension is shown at the end of section 2 where some customer defined fields and repeating groups are required.

Code

Using QuickFix – Prof. Dr. Bernd Ulmann

Using QuickFIX in a project is quite simple – all you need are the files which are placed into the subdirectory quickfix_executor_csharp by the QuickFIX build.

Make sure that QuickFIX and your application relying on it are both built as either 32 bit or as 64 bit applications. We had some problems with QuickFIX being built as "Any CPU" and the application being a 32 bit application.

As a quick hack you can set the 32 bit flag using CorFlags.Exe:

```
corflags quickfix_net.dll /32BIT+
corflags quickfix_messages_net.dll /32BIT+
```

These two DLLs have to be included in your project using the Project→Add Reference menu.

Nearly all of the QuickFIX customization is done in the class
Application which will be shown in more detail in the following.

```
                            Application
1   public class Application : MessageCracker, QuickFix.Application
2   {
3     private SocketInitiator socket_initiator;
4     private MessageStoreFactory message_store_factory;
5     private SessionSettings settings;
6     private LogFactory log_factory;
7     private QuickFix44.MessageFactory message_factory;
8     private SessionID session_id;
9
10    ...
11  }
                            Application
```

This class will hold the necessary method overrides for receiving
messages as well as methods for connection establishment etc. so
the following slides all show methods within this particular class.

Establishing a session is simple since QuickFIX takes care of all the administrative message logic, sequence numbering etc.:

Logon

```
                              logon
1  public void connect(string ini_file)
2  {
3    this.settings             = new SessionSettings(ini_file);
4    this.message_store_factory = new FileStoreFactory(this.settings);
5    this.log_factory          = new FileLogFactory(this.settings);
6    this.message_factory      = new QuickFix44.MessageFactory();
7    this.socket_initiator     = new SocketInitiator(this,
8                                   this.message_store_factory,
9                                   this.settings, this.log_factory,
10                                  this.message_factory);
11   this.socket_initiator.start();
12 }
                              logon
```

Calling this `connect`-method results in a message exchange like this:

Connection establishment
Sending a logon request to the remote system:

```
8=FIX.4.4^9=74^35=A^34=1^49=<SenderCompID>^
52=20101102-00:00:14.198^56=<TargetCompID>^
98=0^108=30^141=Y^10=192
```

This results in the following reply message being received:

```
8=FIX.4.4^9=0076^35=A^49=<SenderCompID>^
56=<TargetCompID>^34=1^52=20101102-00:00:14^
98=0^108=30^141=Y^789=2^10=106
```

QuickFIX makes sure that the sequence numbers are not out of order and takes care of the heartbeat process as well as of test and resend requests.
The heartbeats on an idle connection look like this:

```
20101102-00:00:44.202 : 8=FIX.4.4^9=56^35=0^34=4^49=<SenderCompID>^52=20101102-00:00:44.202^
                         56=<TargetCompID>^10=097
20101102-00:00:44.477 : 8=FIX.4.4^9=0052^35=0^49=<SenderCompID>^56=<TargetCompID>^34=4^
                         52=20101102-00:00:44^10=251
20101102-00:01:14.208 : 8=FIX.4.4^9=56^35=0^34=5^49=<SenderCompID>^52=20101102-00:01:14.208^
                         56=<TargetCompID>^10=102
20101102-00:01:14.479 : 8=FIX.4.4^9=0052^35=0^49=<SenderCompID>^56=<TargetCompID>^34=5^
                         52=20101102-00:01:14^10=250
```

Terminating a session is even simpler than establishing it:

Session termination

```
                          Logout
1  public void disconnect()
2  {
3    this.socket_initiator.stop();
4  }
                          Logout
```

This results in a message like this:

```
8=FIX.4.4^9=56^35=5^34=1^49=<SenderCompID>^
52=20101102-00:00:05.100^56=<TargetCompID>^10=093
```

QuickFIX has a simple mechanism of notifying applications of events by means of callbacks. For logon and logoff the methods are these:

Logon/logout callbacks

```
                        Callbacks
1   public void onLogon(SessionID sessionID)
2   {
3     ...handle the event...
4   }
5
6   public void onLogout(SessionID sessionID)
7   {
8     ...handle the event...
9   }
                        Callbacks
```

Sending messages is quite straight forward although repeating groups can be a bit tricky. In the following the generation of a so called "Trade Capture Report" is described in more detail.
A typical message will contain standard application fields and repeating groups as well as customer defined fields and repeating customer defined groups. An example message will look like this:

```
8=FIX.4.4^               BeginString
9=384^                   BodyLength
35=AE^                   MsgType = TradeCaptureReport
34=1954^                 MsgSeqNum
49=<SenderCompID>^       SenderCompID
52=20101102-16:15:37.711^ SendingTime
56=<TargetCompID>^       TargetCompID
22=4^                    SecurityIDSource = ISIN number
31=92^                   LastPx (price of this fill)
32=10000^                LastQty (quantity)
48=DE00VAX11780^         SecurityID
55=[N/A]^                Ticker Symbol
60=20101101-16:08:38^    TransactTime
64=20101103^             SettlDate
75=20101101^             TradeDate
```

```
150=F^                          ExecType (Trade partial fill or fill)
423=1^                          PriceType (percentage)
487=0^                          TradeReportTransType (new)
541=20101101^                   MaturityDate
552=1^                          NoSides (repeating group)
54=2^                           Side (sell)
37=1280622950^                  OrderID
453=1^                          NoPartyIDs
448=VAX11780^                   PartyID
452=100^                        PartyRole
1=11^                           Account
571=82^                         TradeReportID
572=0^                          TradeReportRefID
9610=2^                         Custom repeating group (2 elements)
9611=LONG^
9612=1^
9613=50000277777710^
9611=LONG^
9612=2^
9613=<comment>^
9654=DT^                        DirectTrade
9896=479^                       BBPricingNumber
9998=VAXSYS^                    BBFirmNumber
10=091                          Checksum
```

Creating a message skeleton and adding some standard message fields is quite straight forward:

Standard message fields

```
                          standard
1   public create create_trade(tdata data)
2   {
3     QuickFix44.TradeCaptureReport message =
4       new QuickFix44.TradeCaptureReport();
5
6     message.set(new TradeReportID(
7       data.trade_report_id.ToString());
8     message.set(new TradeReportRefID("0"));
9     message.set(new LastQty(Math.Abs(data.last_qty)));
10
11    ...
12  }
                          standard
```

Creating the NoSide group and adding portfolio information:

NoSide group

```
                              NoSide
1    // Create and set the NoSide group:
2    QuickFix44.TradeCaptureReport.NoSides no_sides =
3      new QuickFix44.TradeCaptureReport.NoSides();
4    no_sides.set(new Account(data.account));
5    no_sides.set(new Side(data.buy ? '1' : '2'));
6    no_sides.set(new OrderID(data.order_id.ToString()));
7
8    // Set portfolio information in a sub group:
9    QuickFix44.TradeCaptureReport.NoSides.NoPartyIDs no_party_ids =
10     new QuickFix44.TradeCaptureReport.NoSides.NoPartyIDs();
11   no_party_ids.set(new PartyID(data.portfolio));
12   no_party_ids.set(new PartyRole(100));
13   no_sides.addGroup(no_party_ids);
14   message.addGroup(no_sides);
                              NoSide
```

Creating the customer fields and group and sending the message:

Customer fields

```
                                  NoSide
1    message.setField(9896, data.pricing_number);
2    message.setField(9998, data.firm_number);
3    message.setField(9654, "DT");
4
5    // Create repeating subgroup 9610
6    QuickFix.Group repeating_group = new QuickFix.Group(9610, 1);
7    repeating_group.setField(9611, "LONG");
8    repeating_group.setField(9612, "1");
9    repeating_group.setField(9613, data.wis_cntpy);
10   message.addGroup(repeating_group);
11   repeating_group.setField(9612, "2");
12   repeating_group.setField(9613, data.comment);
13   message.addGroup(repeating_group);
14
15   // Send message to BB:
16   Session.sendToTarget(message, this.session_id);
                                  NoSide
```

Receiving application messages requires appropriate overloads of
QuickFIX's onMessage method[2].
Whenever a message is received for which no overloading of
onMessage exists, an Unsupported Message Type-exception will
be thrown!

The following example shows how a mandatory field can be
extracted from a TradeCaptureReportAck message:

Overriding onMessage

```
                              onMessage
1  public override void onMessage(
2    QuickFix44.TradeCaptureReportAck message, SessionID session)
3  {
4    this.trade_ack_data.trade_report_id =
5      message.getTradeReportID().ToString();
6    ...
7  }
                              onMessage
```

[2]This is not necessary for administrative message since QuickFIX already
handles them automatically.

Reading optional fields and customer specific fields is simple, too:

Customer fields

```
                          customer fields
1  this.trade_ack_data.reject_reason =
2    message.isSetTradeReportRejectReason() ?
3      message.getTradeReportRejectReason().ToString() :
4      "0";
5
6  this.trade_ack_data.bb_ticket_number =
7    message.isSetField(9009) ? message.getField(9009) : null;
                          customer fields
```

Configuration

Configuring QuickFix is rather straight forward:

- QuickFix needs its own configuration file (consisting of name/value pairs grouped in named sections).
- The path to this file is set in the initial call of `SessionSettings(...)`.

The following example shows the essentials of such a configuration file:

```
[DEFAULT]
  ConnectionType=initiator
  ReconnectInterval=10
  SenderCompID=<your ID>
  TargetCompID=<your partner's ID>
  FileLogPath=<some suitable path>
  FileStorePath=<some other suitable path>
[SESSION]
  BeginString=FIX.4.4
  StartTime=00:00:00
  EndTime=23:59:59
  HeartBtInt=30
  SocketConnectPort=<port of the target system>
  SocketConnectHost=<address of the target system>
  DataDictionary=<path to your dictionary>FIX44.xml
  ResetOnLogon=Y
```

In addition to this QuickFix uses an XML based description of the expected FIX message format as the basis for validating messages.

To handle customer defined fields, it is necessary to perform some additions to this file to accomodate for the custom specific fields which were used in the examples before.

The various Bloomberg specific fields make the following extensions necessary:

Adding new fields

```
                        ─── FIX44.xml ───
1  <message name=TradeCaptureReportAck' msgcat='app' msgtype='AR'>
2    <field name='BBTicketNumber' required='N' />
3    <field name='BBNumberOfTickets' required='N' />
4    <field name='BBNochNeTicketNumber' required='N' />
5    <field name='BBFixedIncomeFlag' required='N' />
6    <field name='BBFixedIncomeSubFlag' required='N' />
7    <field name='BBFirmPricingNumber' required='N' />
8    <field name='BBUUID' required='N' />
9    <field name='BBFirmNumber' required='N' />
10    ...
11  </message>
12  ...
13  <fields>
14    ...
15    <field number='9009' name='BBTicketNumber' type='INT' />
16    <field number='9103' name='BBNumberOfTickets' type='NUMINGROUP' />
17    <field number='9104' name='BBNochNeTicketNumber' type='INT' />
18    <field number='9894' name='BBFixedIncomeFlag' type='STRING' />
19    <field number='9895' name='BBFixedIncomeSubFlag' type='STRING' />
20    <field number='9896' name='BBFirmPricingNumber' type='STRING' />
21    <field number='9998' name='BBUUID' type='STRING' />
22    <field number='9999' name='BBFirmNumber' type='STRING' />
23  </fields>
                        ─── FIX44.xml ───
```

Conclusion

Using QuickFix – Prof. Dr. Bernd Ulmann 18-OCT-2010

It may be concluded that

- QuickFix is a very powerful FIX engine that is very well production worthy,

- developing inhouse applications using QuickFix is quite easy (although the generation of repeating groups is sometimes a bit tricky – adding the portfolio information took some time),

- QuickFix mastered all connectivity testing without any problem at all – dropping connections, sending messages with erroneous sequence numbers etc. were no problems at all.

- The author would like to thank Dr. Reinhard Steffens for his support and proof reading of these slides.
- The author can be reached at `ulmann@vaxman.de`.

Thank you for your interest!