FIX Protocol basics

Prof. Dr. Bernd Ulmann

18-OCT-2010

Hochschule fuer Oekonomie und Management, Frankfurt
Introduction
1. Introduction

What is the FIX Protocol?

- Acronym for *Financial Information eXchange* Protocol (not to be confused with *IPFIX*, the *Internet Protocol Flow Information Export*).

- Open standard which defines a message format as well as a communication model.

- This standard has been created by an industry consortium consisting of banks, independent vendors etc.

- The intended audience consists of financial institutions like banks, brokers, dealers, exchanges etc.

- FIX is *the* defacto standard in financial information exchange today.

- FIX is platform independent (concerning systems and networks).

- Central point for all information about FIX is [http://www.fixprotocol.org](http://www.fixprotocol.org).
How has FIX evolved over the years\(^1\)?

- FIX was conceived by the equity trading departments of *Fidelity Management & Research* and *Salomon Brothers*.

- After the feasibility of the concept developed by these two companies became apparent, a FIX committee was established which began work in June 1994.

- In 1995 the first Technical Committee Meeting took place.

- Also in January 1995, FIX was released to the financial community in version 2.7.

- In September 1995 version 3.0 of the FIX Protocol was released.

---

\(^1\)These slides are based on http://fixprotocol.org/implementation-guide/introduction.shtml, 28.10.2010.
The London FIX General Conference was held in June 1996.
- FIX 4.0 was released in 1997.
- FIX 4.1 was released in 1998.
- In June 1998 the FIX Committee structure was formalized.
- In March 1999 the Japanese FIX Committee was formalized.
- In April 1999 the FIX Protocol Ltd. was started.
- The FIX Protocol was extended for fixed income trading by *Putnam Investments* and *Merrill Lynch*. A pilot implementation was running in March 2000.
- FIX 4.2 was released in March 2000 and included four additional tags for fixed income trading.
The extension also included so called *user defined tags, UDTs* for short.

In 2000 the FIX Protocol organization established the *Fixed Income Working Group, FIWG* for short which became known as the *Global Fixed Income Committee (GFIC)*.

The *FIX Protocol Limited (FPL)* entered so called *Statements of Understanding (SoU)* with other organizations like *SWIFT*, the *Bond Market Association* etc.
Who uses the FIX Protocol? The main user groups are these:

**Buy-side firms:** Communication with sell-side firms by means of pre-trade, trade and post-trade messages.

**Sell-side firms:** Communication with buy-side firms via pre-trade, trade and post-trade messages. In addition to that communication with exchanges and OTC markets in general.

**Exchanges:** Receiving trades from their members, sending execution reports etc. back to them.

Currently a wide variety of product classes are supported ranging from equities to fixed income products, derivatives and the like.
Benefits of using FIX

1. Introduction

FIX offers a wide range of benefits in various areas:

- FIX simplifies the implementation of interfaces by using so-called *FIX engines* (there are open source implementations as well as commercial ones).

- FIX increases the efficiency on the dealer side by saving time for price and execution data transmission.

- Since FIX defines message semantics, it reduces the risk of human error in trade entry etc. and makes it possible to detect errors earlier in the business process flow if they occur.

- FIX engines implement all necessary means for logging FIX message, for dealing with dropped connections etc. thus simplifying application development.

- Nearly every broker/dealer/exchange *speaks* FIX.

- Nearly all vendors of *order management systems*, *OMS*, offer FIX connectivity.
The main benefits of FIX from a trader’s perspective are the following ones:

- The use of FIX "has allowed trading volumes to increase without corresponding increases in staffing".
- FIX "allows traders to focus on dialogue and not on clerical tasks".
- FIX "allows an individual to trade more stocks and decrease the chance of errors at the same time".
- "With the increased volatility in the marketplace, FIX enables me to stay on top of my positions in real-time."
- FIX "prevents Nick Leeson-type activities before they can even happen, not just next day on confirmation or on settlement day".

\(^{2}\text{Cf. [Hong Kong 00][p. 25 ff.].}\)
A typical FIX setup is shown below:\(^3\):

\[\text{FIX System Connectivity}\]

Customer (i.e. Investment Mgr)

- FIX System
  - Business Msg Processing
  - FIX Engine
  - Order Management System
  - Trader
  - Trader
  - Trader

Supplier (i.e. Broker/Dealer)

- FIX System
  - Business Msg Processing
  - FIX Engine
  - Business Msg Processing
  - Order Management System
  - Trader
  - Trader
  - Trader

Wide Area Network Link

TCP/IP

(TCP Socket opened by customer, persists during life of FIX session)

---

Message structure and fields
Although FIX messages always have a similar structure, the number and type of possible fields depends strongly on the version of the FIX Protocol which is used.

At the time of this writing the latest version is FIX 5.0 although most current production systems use older versions 4.x (4.4 being the last).

FIX messages are grouped into two categories:

**Admin messages:** Connection establishment and termination, heartbeat messages etc. (logon, logoff, heartbeat, test request, resend request, reject, sequence reset, . . . )

**Application messages:** Business related message transferring trade data etc. (advertisement, indication of interest, news, execution report, order cancel, new order, quote and many, many more)
Every FIX message is built according to the following structure:

**Message header:** Contains the message type, length, sender/receiver name, sequence number, time stamp, ...

**Message body:** Contains session/application specific data.

**Message trailer:** Contains a message checksum and an optional signature.

The message consists of so called *FIX fields* which look like this:

\[ <\text{tag}>=<\text{value}><\text{delimiter}> \]

The tag is a numerical identifier, the value is a string (which must be of correct type and length for this particular tag) and the delimiter is SOH (*Start of header* – ASCII 0x01) which is written as \(^1\) in printed messages.

Although FIX supports literally hundreds of predefined tags, there is room for custom defined tags (field numbers 5000 to 9999).
The following example of a *New Order Single* message is taken from [FIXML 00][p. 8]:

8=FIX.4.1^9=0235^35=D^34=10^43=N^49=VENDOR^50=CUSTOMER^56=BR0KER^52=19980930-09:25:58^1=XQCCFUND^11=10^21=1^55=EK^48=277461109^22=1^54=1^38=10000^40=2^44=76.750000^59=0^10=165

**Header:** 8 (version), 9 (body length), 35 (MsgType), 34 (MsgSeqNum), 43 (PossDupFlag), 49 (SenderCompID), 115 (OnBehalfOfCompID), 56 (TargetCompID), 52 (time stamp)

**Body:** 1 (Account), 11 (ClOrdID), 21 (HandInst), 55 (Symbol), 48 (SecurityID), 22 (IDSource), 54 (Side), 38 (OrderQty), 49 (OrdType), 44 (Price), 59 (TimeInForce)

**Trailer:** 10 (Checksum)
Message types

The FIX Protocol supports many different message types – some of these are shown in the following:

0: Heartbeat
1: Test request
2: Resend request
3: Reject
4: Sequence reset
5: Logout
6: Indication of interest
7: Advertisement
8: Execution report
9: Order cancel reject
A: Logon
B: News
C: Email
D: Order single
E: Order list
The very same message may also be formatted using FIXML (unfortunately FIXML is not in widespread use)\(^4\):

```
<?xml version='1.0'?><!DOCTYPE FIXML SYSTEM 'fixmlmain.dtd'>
<FIXML>
  <FIXMLMessage>
    <Header>
      ...
    </Header>
    <ApplicationMessage>
      <Order>
        <ClOrdID>12345</ClOrdID>
        <HandlInst Value="1"/>
        <Instrument>
          <Security>
            <Symbol>EK</Symbol>
          </Security>
        </Instrument>
        <Side Value="1"/>
        <OrderQuantity>
          <OrderQty>10000</OrderQty>
        </OrderQuantity>
        <OrderType>
          <MarketOrder OrdType="1"/>
        </OrderType>
        <Currency Value="USD"/>
      </Order>
    </ApplicationMessage>
  </FIXMLMessage>
</FIXML>
```

\(^4\) Cf. [FIXML 00][p. 10].
FIXML can even be embedded into *standard* FIX messages to facilitate migration and maintain backward compatibility:

```xml
... 49=BROKER 56=HUB 128=INST 212=245 213=<FIXML>
    <Header>
      ...
    </Header>
    <Indication>
      ...
    </Indication>
  </FIXML>
...```

\(^5\)Cf. [FIXML 00][p. 12].
FIXML or not FIXML?

2. Message structure

+:
- FIXML can be parsed with any XML parser.
- FIXML allows automatic message validation.
- FIXML is more human readable compared with traditional FIX.
- FIXML can be processed by various middlewares etc.

−:
- FIXML needs much more bandwidth than the traditional FIX Protocol.
- Parsing (and especially validating) FIXML takes way more CPU resources than processing traditional FIX messages.
- Still only a few parties use FIXML.
Further reading

2. Message structure

- [Shaik 10] contains a wealth of practical examples of FIX messages ranging from single orders to order cancel/replace etc.

- Some example heartbeat and IOI FIX messages can be found at http://fixprotocol.org/specifications/TechResources-Examples
Network
Communication using the FIX Protocol has the following properties:

- It is session based and thus point to point from a logical perspective.

- Communication partners act according to one of these roles:
  
  **Initiator:** Initiates a communication by sending a *logon* message and ends it with a *logout* message – this is the *client*.

  **Acceptor:** Receives a login request, validates it and establishes the connection – this is the *server*.

- The FIX protocol implements a session layer.

- Messages are identified by sequence numbers. These are used for resend or reject requests and the like.

- Normally new sessions start with sequence number 1.
Choosing a network

FIX is rather network agnostic in that it supports a wide variety of network models. Basically there are four network architectures to choose from:

**Leased line:** In this case a leased line is employed between two counter parts – the big advantage is that this line is truly private. On the other hand this solution becomes quickly expensive when many leased lines and/or high data rates/low latencies are necessary.

**Internet:** This is the cheapest solution from a technical point of view. The main disadvantages are stability, bandwidth and latency issues – all of these parameters can and will vary extremely. Another disadvantage is the need for encryption (SSL, AES etc.).

**P2P-VPN:** Quite like the Internet variant but the encryption is normally done at the router level.

**Hub-and-Spoke:** Here a central FIX engine will be employed in the hub from which all the users connect.
FAST
FAST is the acronym for *FIX Adapted for STreaming* and describes a protocol which has been developed for the one-way exchange of data between a sender and one or multiple receivers.

To quote [FAST 06][p. 5 f.]:

- "**FAST, at its core, is a data compression algorithm which when properly implemented will significantly reduce bandwidth requirements and latency between sender and receiver.**"

- "**FAST ist an extension of the base FIX specification [. . .]. FAST exists as a stand alone specification which can be used within either broadcast or point-to-point transports.**"

FAST performs the compression at two levels:

- Field encoding (controlled by templates which specify the message structure)
- Transfer encoding
The following picture shows the levels FAST operates on:

\[\text{Cf. [FAST 06][p. 6].}\]
A typical area for using FAST is broadcast communication where one sender distributes data to a multitude of receivers.

This is normally implemented using so called *multicasts* and UDP as the transport mechanism.

Since UDP is a connection less communication model it is important for FAST to be employed on a frame-by-frame basis, thus data sent and received should never span multiple frames.
FIX engines
An actual implementation of the FIX Protocol is called a *FIX engine*. There is a wide variety of FIX engines available on the market – some of which are open source while the majority are rather expensive and sometimes only usable with other systems from a particular vendor.

For small applications it might even be an option to develop your own FIX engine – the protocol is quite simple and if you can live without semantics checking, writing your own FIX engine could result in a remarkable short time to market (the author would, of course, use Perl for the implementation of such a small FIX engine).

Selecting a FIX engine is not a simple task and involves looking at the following criteria\(^7\) at least:

FIX engine selection criteria

- What FIX versions are supported?
- Are all tags you need supported?
- Is the FIX engine able to run multiple FIX Protocol versions?
- Does it support your asset class?
- How about stability, extendability etc.?
- How much throughput is possible? Which memory/CPU requirements does the engine have?
- Does the engine run in your preferred hardware/software environment (this includes necessary databases, too)?
- Do you need a FIX engine that is capable of taking care of business logic or will you implement the business logic for your connection(s) at the application level?
- Does the vendor of your business system offer an own FIX engine?
- What about support? Are the sources available?
- Are there monitoring utilities included?
Anatomy of a FIX engine

The following picture\(^8\) shows the anatomy of a typical FIX engine:

\(^8\) Cf. [Northey 04][p. 8].
A FIX engine implements the FIX session level protocol and takes care of things like logon, log out, sequence numbering, message resend requests etc. A typical login scenario looks like this⁹:

⁹ Cf. [Northey 04][p. 12].
At logon time the sequence number is checked - if it is too high, something has been lost and has to be resent etc.\(^{10}\):

\(^{10}\) Cf. [Northey 04][p. 16].
Dealing with a sequence number being too low results in closing the connection\textsuperscript{11}:

\begin{itemize}
  \item Login $\text{MsgType=\texttt{A}}$, $\text{TargetCompID=CBOEFIX001}$
  \item $\text{SenderCompID=M001}$, $\text{MsgSeqNum=1}$
  \item Valid $\text{SenderCompID}$
  \item Sequence Number $< \text{Expected Value}$
  \item Logout $\text{MsgType=\texttt{5}}$, $\text{TargetCompID=M001}$
  \item $\text{SenderCompID=CBOEFIX001}$, $\text{MsgSeqNum=1}$
  \item Text = $\text{Incoming Sequence Number} < \text{Expected} = 5$
\end{itemize}

\textsuperscript{11} Cf. [Northey 04][p. 17].
During an established session heartbeat packets are exchanged in regular (definable) intervals to make sure that the connection is still valid\textsuperscript{12}:

\textsuperscript{12}\textit{Cf. [Northey 04][p. 20].}
In addition to this test requests are supported:\(^1_3\):

\(^{13}\) Cf. [Northey 04][p. 23 f.]
Handling a logout request works like this\textsuperscript{14}:

\begin{itemize}
\item It is considered bad form to close the connection prior to receiving the logout confirmation.
\item Wait a brief period of time (heartbeat interval) for the other side to send a logout – this is done in case the other side needs to do some processing.
\item Both sides disconnect by closing the socket.
\end{itemize}

\textsuperscript{14} Cf. [Northey 04][p. 27].
Available FIX engines

The following list of FIX engines is far from being complete – nearly every software vendor in the financial sector has its own FIX engine – the following entries just serve as examples:

**FIX Antenna C++:**

http://b2bits.com/trading_solutions/fix_engine_cpp.html
(also available for Java and .NET)

**QuickFIX:** A free production quality FIX engine implementation
(see http://www.quickfixengine.org/).

**QuickFIX/J:** Quite the same as QuickFIX but completely Java based
(see http://www.quickfixj.org/).

**VersaFix:** An open source .NET based FIX engine, implemented in C#
(see http://sourceforge.net/projects/versafix/).

**UL FIX:** Yet another free FIX implementation (cf.
After you have chosen a FIX engine to be used and integrated it into your existing environment, you have to test it prior to use it in a production environment.

Most communication partners like Bloomberg etc. require the following tests to be performed successfully with their support before you will be allowed to communicate with them via FIX:

**Connectivity testing:** Basic connectivity, reconnect in case of errors etc. (∼10%)

**Session level testing:** Are the generated messages valid and compatible with your communication partner etc. (∼10%)

In addition to that you will need more tests inhouse:

**FIX/OMS (order management system) tests:** ∼40%

**Integration testing:** ∼40%
Session level tests normally include the following scenarios:\(^{15}\):

- Stopping heartbeat on client and/or host to simulate a flaky connection.
- Send messages with sequence numbers being too high and too low and check the response of the FIX engines.
- Check how the FIX engines deal with messages they have missed (or think they have missed).
- …

\(^{15}\)Cf. [Johnson, Rhodes 01][p. 32].
Typical test scenarios from the business level viewpoint are the following\(^{16}\):

**Orders:**

- Test all required parameters and their domains (numerical values, strings containing special characters, date/time values, \ldots).
- The same tests are required for optional parameters.
- Test optional order types like Stop or Stop Limit etc.

**Cancels:**

- Simple cancel.
- Cancel after a Partially Filled.
- Partially filled while pending cancel.
- Unsolicited cancels etc.

\(^{16}\)Cf. [Johnson, Rhodes 01][p. 30].
FIX tools
FIX tools

**FIXForge FIX Dictionary:**
http://www.onixs.biz/tools/fixdictionary/

**FIXimate FIX Dictionary:** This is the ultimate resource for digging deeper into FIX message of various versions (see http://www.fixprotocol.org/FIXimate3.0/).

**FIXopaedia:** http://btobits.com/fixopaedia/index.html

**FIXwiki:** http://fixwiki.org/fixwiki/FIXwiki

**Mini-FIX:** This is a handy Windows tool which implements a simple FIX client/server with a simple GUI. Mini-FIX has been proven to be very valuable during development of FIX based interfaces etc. (see http://69.64.38.175/).

**ValidFIX:** This is a free online tool for inspecting FIX messages as well as logs produced by FIX applications (see http://www.validfix.com).
Browse FIX Dictionary:

<table>
<thead>
<tr>
<th></th>
<th>Messages by</th>
<th>Fields by</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIX 4.0</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 4.1</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 4.2</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 4.3</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 4.4</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 5.0</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 5.0.SP1</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
<tr>
<td>FIX 5.0.SP2</td>
<td>(MsgType)</td>
<td>Name</td>
</tr>
</tbody>
</table>

*Note:* You must have JavaScript enabled in your web-browser to use all dictionary features.

For more information on Onix Solutions or any of our products please e-mail info@onixs.biz.
### FIX.4.4 Message

**ExecutionReport [type '8']**

```
<ExecRpt>
```

The execution report message is used to:

1. Confirm the receipt of an order
2. Confirm changes to an existing order (i.e. accept cancel and replace requests)
3. Relay order status information
4. Relay fill information on working orders
5. Relay fill information on tradeable or restricted tradeable quotes
6. Reject orders
7. Report post-trade fees calculations associated with a trade

<table>
<thead>
<tr>
<th>Tag or Component</th>
<th>Field Name</th>
<th>FIXML name</th>
<th>Req'd</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>StdHeader</td>
<td>@OrderID</td>
<td>✓</td>
<td>StdType = 8, OrderID is required to be unique for each chain of orders.</td>
</tr>
<tr>
<td>37</td>
<td>OrderID</td>
<td>@OrderID</td>
<td>✓</td>
<td>StdType = 8, OrderID is required to be unique for each chain of orders.</td>
</tr>
<tr>
<td>198</td>
<td>SecondaryOrderID</td>
<td>@OrderID2</td>
<td></td>
<td>Can be used to provide order ID used by exchange or executing system.</td>
</tr>
<tr>
<td>526</td>
<td>SecondaryClOrdID</td>
<td>@ClOrdID2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>527</td>
<td>SecondaryExecID</td>
<td>@ExecID2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>ClOrdID</td>
<td>@ClOrdID</td>
<td></td>
<td>Required for executions against electronically submitted orders which were assigned an ID by the institution or intermediary, not required for orders manually entered by the broker or fund manager (for CIV orders).</td>
</tr>
<tr>
<td>42</td>
<td>OrigClOrdID</td>
<td>@OrigID</td>
<td></td>
<td>Conditionally required for response to an electronic Cancel or Cancel/Replace request (ClOrdType=Cancel, Replace, or Cancel). ClOrdID of the previous accepted order (NOT the initial order of the day) when cancelling or replacing an order.</td>
</tr>
<tr>
<td>583</td>
<td>ClOrdIDLinkID</td>
<td>@LinkID</td>
<td></td>
<td></td>
</tr>
<tr>
<td>693</td>
<td>QuoteRespID</td>
<td>@QuoteID</td>
<td></td>
<td>Required if responding to a QuoteResponse message. Echo back the initiator's value specified in the message.</td>
</tr>
<tr>
<td>790</td>
<td>OrdStatusRespID</td>
<td>@StatRespID</td>
<td></td>
<td>Required if responding to and if provided on the Order Status Request message. Echo back the value provided by the requestor.</td>
</tr>
<tr>
<td>584</td>
<td>MassStatusRespID</td>
<td>@StatRespID</td>
<td></td>
<td>Required if responding to and if provided on the Order Mass Status Request. Echo back the value provided by the requestor.</td>
</tr>
<tr>
<td>911</td>
<td>ToNumReports</td>
<td>@ToNumRpts</td>
<td></td>
<td>Can be used when responding to an Order Mass Status Request to identify the total number of Execution Reports which will be returned.</td>
</tr>
<tr>
<td>912</td>
<td>LastRptRequested</td>
<td>@LastRptReqd</td>
<td></td>
<td>Can be used when responding to an Order Mass Status Request to indicate that this is the last Execution Reports which will be returned as a result of the request.</td>
</tr>
<tr>
<td>Component</td>
<td>Parties</td>
<td>Party</td>
<td></td>
<td>Insert here the set of Parties (firm identification) fields defined as &quot;Common Components of Application Messages&quot;</td>
</tr>
<tr>
<td>229</td>
<td>TradeOriginDateTime</td>
<td>@OrigDt</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Welcome to FIXopaedia the fastest, most comprehensive and feature rich FIX reference source. B2BITS' FIXopaedia is built on top of our complete set of FIX Dictionaries and allows examining the elements required for session and application level messages across all versions of the FIX Protocol (FIX 4.0 - 5.0, 5.0 SP1, 5.0 SP2):

- Component Blocks
- Message Name
- Message Type
- Field Name
- Field Type

Please click below to see the full logical specification of every message and tag type for the version of FIX that you require.

Version 2.2.1

- FIX 4.0
- FIX 4.1 (with errata 19990630)
- FIX 4.2 (with errata 20010501)
- FIX 4.3 (with errata 20020820)
- FIX 4.4 (with errata 20030818)
- FIX 5.0
- FIX 5.0 SP1
- FIX 5.0 SP2
- FIXT 1.1

The following FIX engines provide complete support of all these dictionaries:

- FIX Antenna C++
- FIX Antenna .NET
- FIX Antenna Java
- CME FIX/FAST Market Data Adapter
FIXwiki

FIXwiki is a wiki containing data from the FIX specification. There is a FIXwiki page for each FIX message, component, field, value or type.

In addition to definitions taken from the specification, each page also has an area for user comments, clarifications, corrections, examples or suggestions. Please contribute.

FIXwiki was created by John Cameron of Cameron Edge for the benefit of the FIX community. It is completely free.

Quick Start

Log in then type one of the following into the search box on the left:
- a FIX message or component name (eg Confirmation or Parties)
- a FIX field name (eg Quantity)
- a FIX tag number (eg 54)

or if you just want to browse, click on one of the following links:
- FIX Messages
- FIX Components
- FIX Fields
- FIX Values
- FIX Types

Click on the Cameron Edge logo at the top left at any time to return to this main page.

More Detail

Using FIXwiki.
Structure of FIXwiki.
About FIXwiki.
Thank you for your interest.

The author would also like to thank Dr. Reinhard Steffens for his support and proof reading.


[Johnson, Rohdes 01] Sam Johnson, David Rhodes, FIX/FIXML implementation