CF969 - Introduction to the FIX Protocol

Steve Phelps

CCFEA

February 26, 2014

Overview

- Acronym for _Financial Information eXchange Protocol
- Open standard
- FIX is the defacto standard in electronic trading
- Platform independent
  - Libraries for most languages and architectures
  - Network independent
- `http://www.fixprotocol.org`

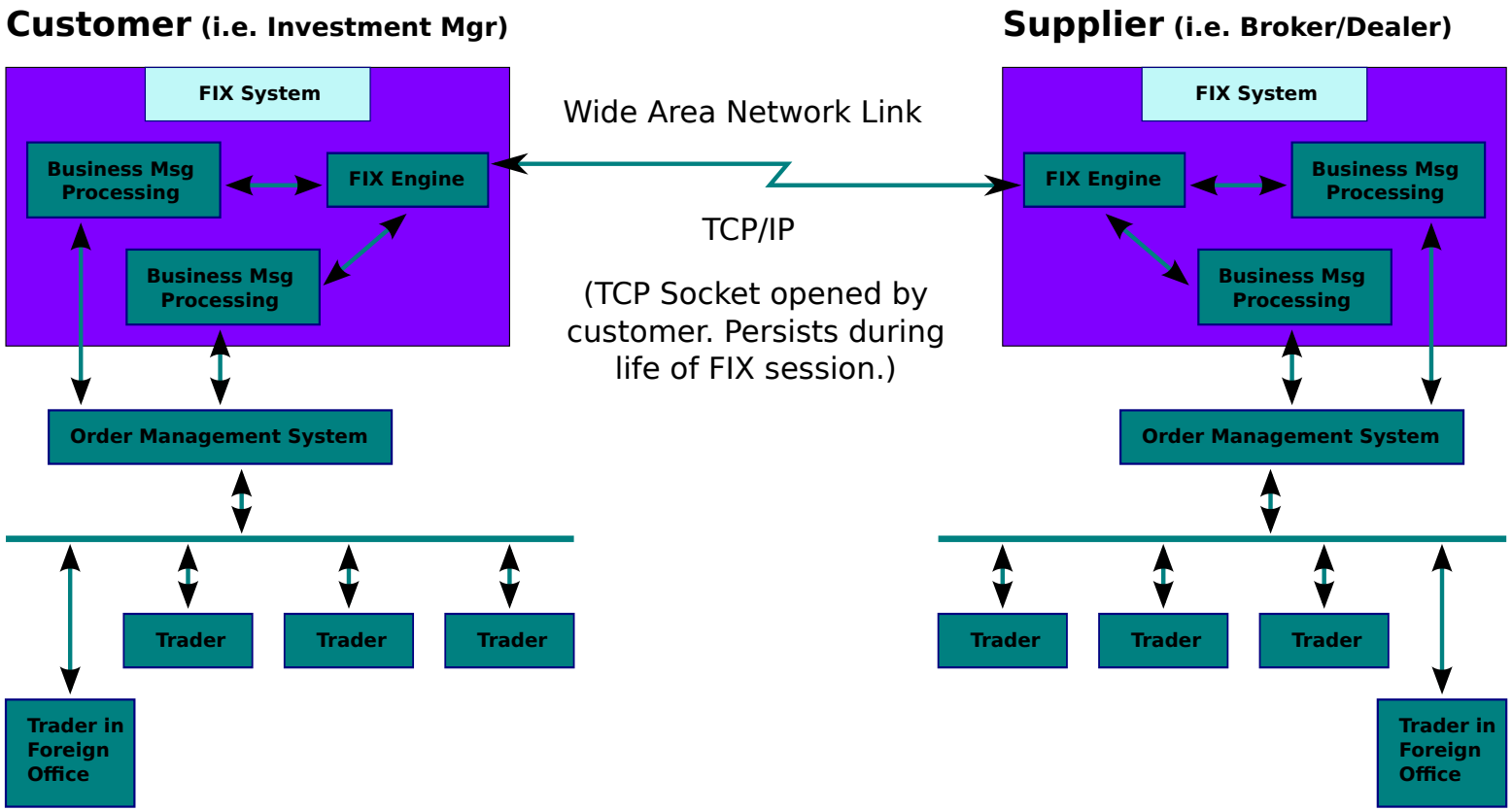
History

- FIX version 2.7 released in January 1995
- Version 3.0 of FIX released in September 1995
- FIX 4.0 in 1997
- FIX 4.1 in 1998
- FIX 4.2 released in March 2000
  - Contained extensions for user defined tags (UDTs)
- Current version 5.0 SP2 in August 2011
- Many different *dialects* for e.g. low-latency trading
  `http://www.onixs.biz/fix-dialect.html`

What it does

- Provides a session and transport network protocol for sending and receiving information about financial events
- Allows for automated trading
- Provides a common standard allowing different financial systems to communicate
- Nearly all *order management systems* offer FIX connectivity
- Many back-testing frameworks use FIX
- Low-level high-frequency tick data is sometimes shipped in FIX format

Overview


Customer (i.e. Investment Mgr)
FIX System
Business Msg Processing
FIX Engine
Business Msg Processing
Order Management System
Trader
Trader
Trader
Trader in Foreign Office
Wide Area Network Link
TCP/IP
(TCP Socket opened by customer. Persists during life of FIX session.)
Supplier (i.e. Broker/Dealer)
FIX System
FIX Engine
Business Msg Processing
Business Msg Processing
Order Management System
Trader
Trader
Trader
Trader in Foreign Office

Relationship with TCP/IP

- FIX was originally designed to be network independent
- Implements reliable ordered transport
- Also provides session-layer functionality and authentication
- Typically implemented via TCP even though it already implements transport functionality
- Other dialects use e.g. UDP for low-latency

Message Format

- Every FIX message maps set of keys onto corresponding values.
- In FIX, the keys are called *fields*.
- Each field has a unique integer identifier, called a *tag*.
- Each mapping is separated by a delimiter. The delimiter is an ASCII character with code `0x01` which represents `SOH` (Start of header). This character is often rendered using the symbol `^`.

Fields

- Each binding of a field to a value is sent in the following format:

  `<tag>=<value><delimiter>`

- Where `<tag>` is an integer value.
- There are predefined tags corresponding to universal attributes, see e.g.:

`http://www.onixs.biz/fix-dictionary/4.2/fields_by_tag.html`

- FIX allows custom tags (field numbers 5000 to 9999)

Field Categories

Every FIX message contains:

- Message header: contains the message type, length, sender/receive name, sequence number, time stamp
- Message body: contains the attributes of the message
- Message trailer: contains a checksum and optional signature

Example Message

```
8=FIX.4.1^9=0235^35=D^34=10^43=N^49=VENDOR^50=CUSTOMER^
56=BROKER^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

- Tag 35 (MsgType) specifies the type of message
- `http://www.onixs.biz/fix-dictionary/4.2/msgs_by_msg_type.html`

Message Categories

- Message types belong to the following categories

1. Admin messages
   - Session establishment, termination and management
     - ★ logon, logoff, hearbeat, test request, resend request, reject, sequence reset
2. Application messages
   - execution report,
   - new order
   - order cancel etc. . .

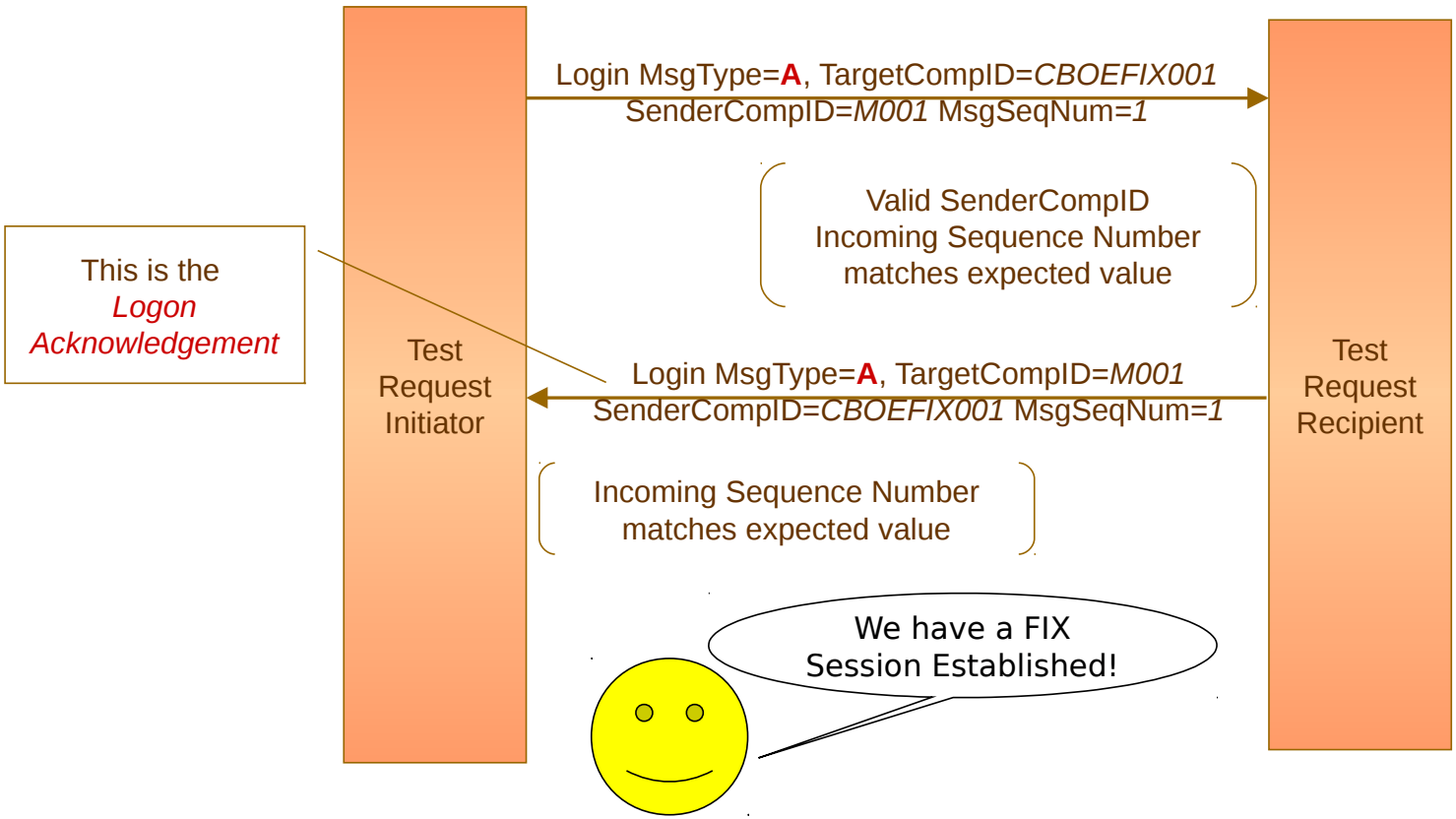
Transport Layer

- Network agnostic, but in practice TCP/IP is the de facto standard.
- Encryption provided SSL or a VPN.

Protocol

- FIX is a session-based protocol.
- When using TCP, a TCP connection corresponds to a FIX session.
- There are two end points:
  - Initiator/Client: Initiates a session by sending a *logon* message and terminates it with a *logout* message
  - Acceptor/Server: Authenticates logon and establishes the connection
- Each message has an attached sequence number (on top of TCPs own sequence numbering). As with TCP, there are two sequence numbers:
  - incoming message sequence number
  - outgoing message sequence number
- By default, new sessions start with sequence number 1

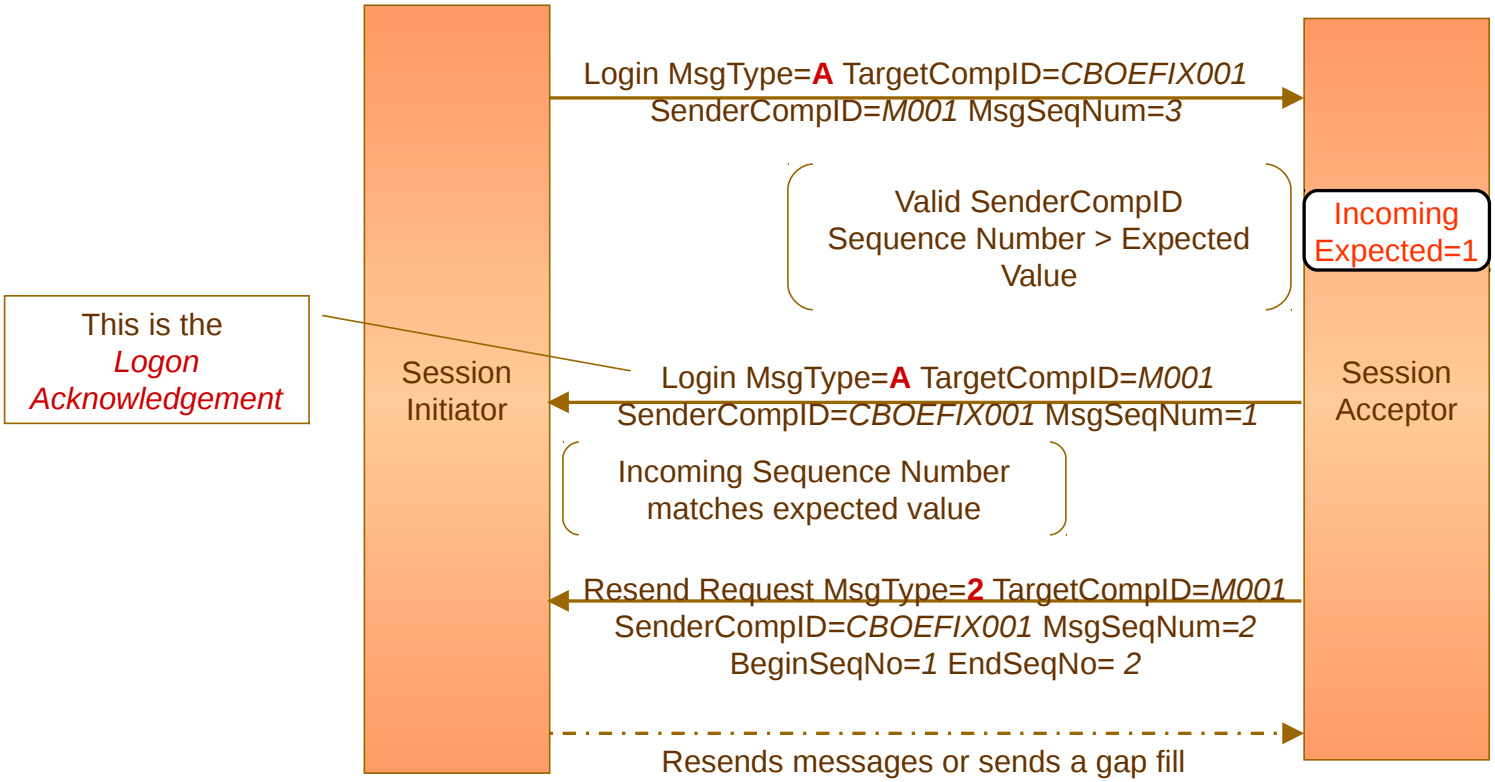
Session Establishment



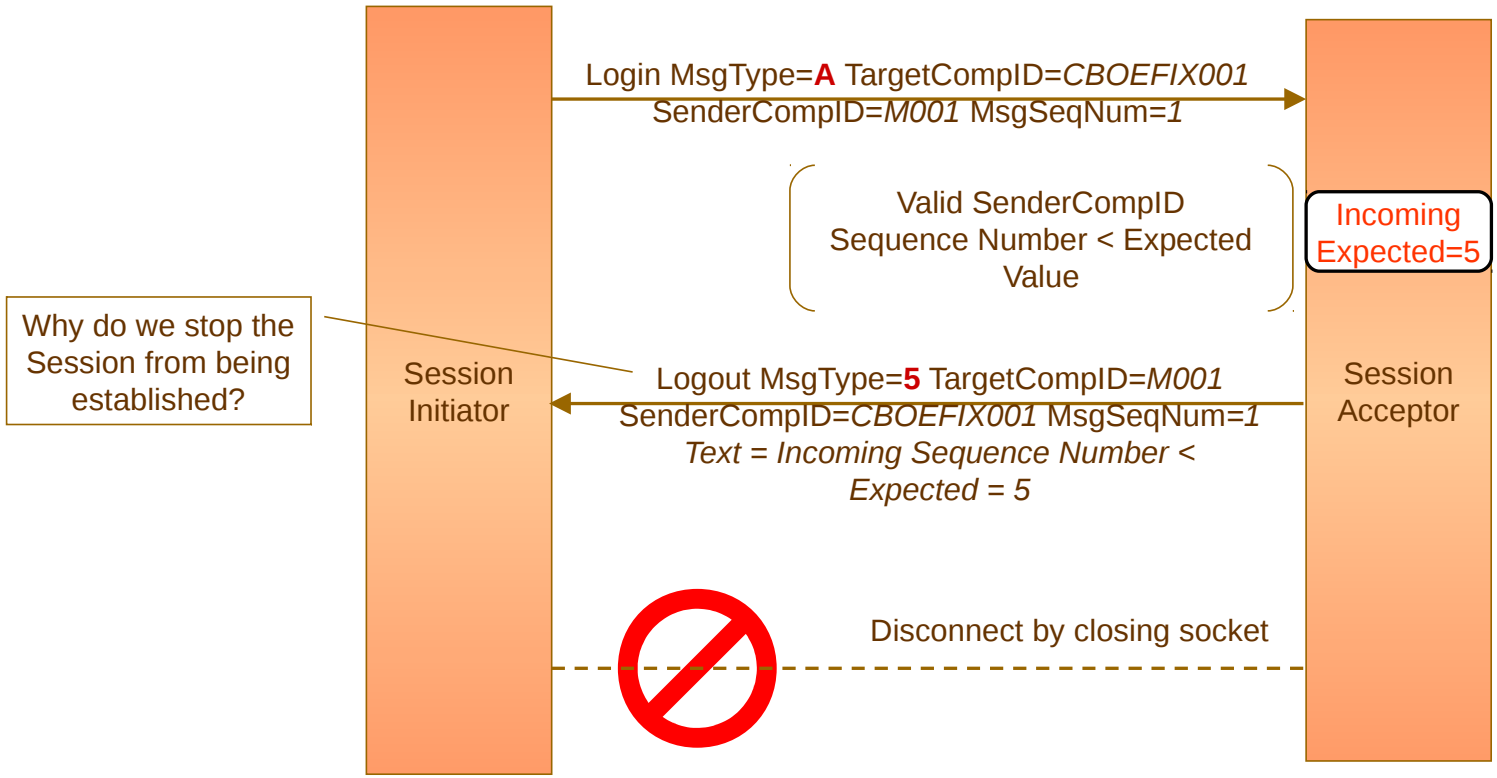
Resend Requests

- A message with MsgType=2 can be used to request a resend
- It has fields BeginSeqNo (tag 7) and EndSeqNo (tag 16) which specify the messages that you did not receive

Sequencing exceptions



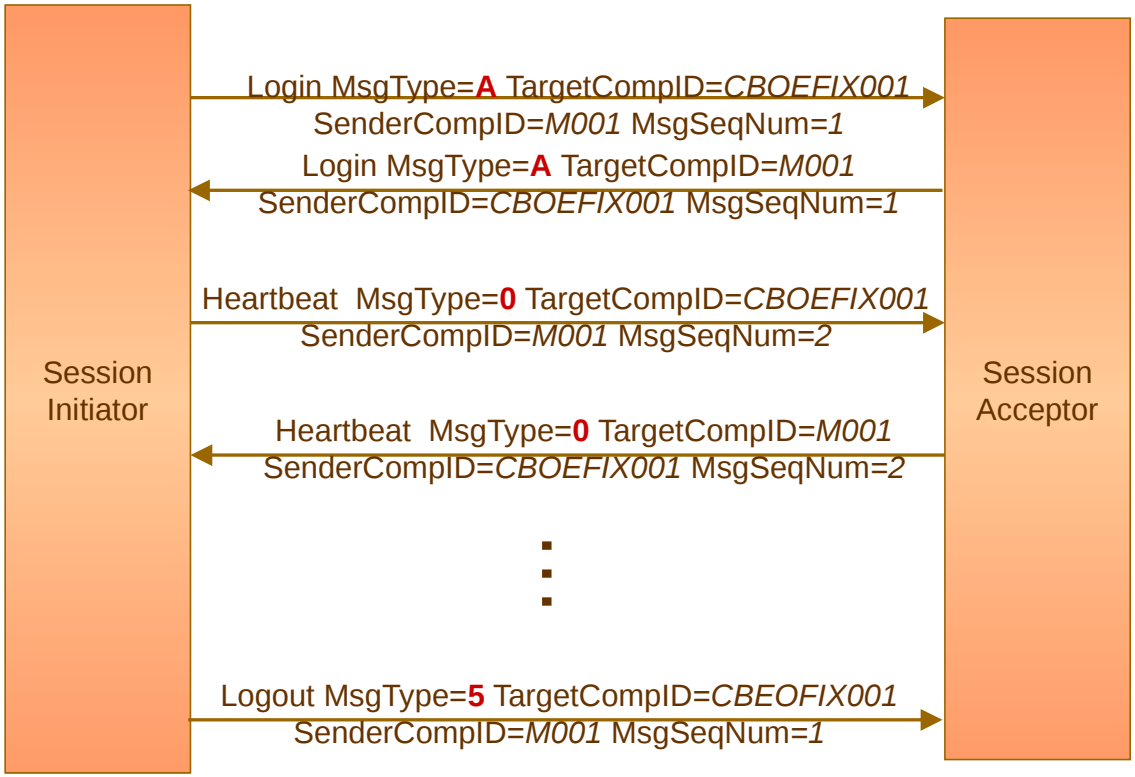
Sequencing exceptions



Heartbeat messages

- A FIX Session will be kept alive provided that regular messages are sent
- Otherwise a timeout will occur
- In order to keep the session alive even if there is no market activity, you can send Heartbeat messages
- MsgType=0

Heartbeat


Session Initiator
Session Acceptor
Login MsgType=A TargetCompID=CBOEFIX001 SenderCompID=M001 MsgSeqNum=1
Login MsgType=A TargetCompID=M001 SenderCompID=CBOEFIX001 MsgSeqNum=1
Heartbeat MsgType=0 TargetCompID=CBOEFIX001 SenderCompID=M001 MsgSeqNum=2
Heartbeat MsgType=0 TargetCompID=M001 SenderCompID=CBOEFIX001 MsgSeqNum=2
Logout MsgType=5 TargetCompID=CBEOFIX001 SenderCompID=M001 MsgSeqNum=1

FIX engines

When developing software to talk another FIX system, you can either:

- Implement the protocol yourself, e.g. using the Java sockets API.
- Use an existing library which provides an API to send FIX messages.
- FIX libraries are called FIX *engines*.
- For example, QuickFix and QuickFix/J
- `http://www.quickfixj.org/`
- As with any other software development project, selection of an appropriate FIX engine will be driven by application requirements.

Anatomy of a FIX engine