FINANCIAL INFORMATION EXCHANGE PROTOCOL (FIX)

Version 4.4 with Errata 20030618

VOLUME 2 – FIX SESSION PROTOCOL

Includes Errata adjustments as of June 18, 2003

Errata Purpose:

This document includes a list of minor adjustments to the FIX 4.4 Specification document due to typographical errors or ambiguities. The nature and scope of Errata adjustments do not introduce new functionality, additional fields, new values for existing fields, or new messages. **Regretably some functionality was introduced in FIX 4.4 which contained errors that required a new value or field on a specific message in order to make the intended functionality implementable. Any such exceptions to the "do not introduce", "additional fields", or "new messages" Errata rules were kept to a minimum using the "required to make the intended functionality implementable" rationale.** The list of items has been reviewed and approved by the FIX Technical Committee and Steering Committees. Implementers of FIX version 4.4 should refer to this document to ensure the most consistent implementation and clearest understanding of the FIX protocol.

The specific adjustments made to the original FIX version 4.4 specification as a result of the Errata can be seen and printed via Microsoft Word's revision feature of this document. A separate document with an itemized list of changes is available via the FIX website.

~~April 30, 2003~~June 18, 2003

Contents – Volume 2

COMMUNICATION USING THE FIX SESSION PROTOCOL

INTRODUCTION

FIX was written to be independent of any specific communications protocol (X.25, asynch, TCP/IP, etc.) or physical medium (copper, fiber, satellite, etc.) chosen for electronic data delivery. It should be noted that if an "unreliable" or non-stream protocol is used, the Logon, Logout, and ResendRequest message processing is particularly susceptible to unordered delivery and/or message loss.

The protocol is defined at two levels: session and application. The session level is concerned with the delivery of data while the application level defines business related data content. This document focuses on the delivery of data using the "FIX Session Protocol".

TRANSMITTING FIXML OR OTHER XML-BASED CONTENT

Note that while the FIX Session Protocol is based upon "Tag=Value" syntax for the Standard Header, Standard Trailer, and the Administrative Messages which make up the FIX Session Protocol, it is possible to send FIXML or other XML-based content ("payload") via the FIX Session Protocol. The FIXML or other XML-based content is enclosed in a traditional "Tag=Value" FIX standard header via the standard header's XmlDataLen and XmlData fields and followed by the "Tag=Value" FIX standard trailer. This allows a FIX Engine (software which implements the FIX Session Protocol) to transmit FIXML or other XML-based content via the robust, real-time asynchronous transport which has been in use for many years. The generic MsgType field (tag 35) value of "n" (lowercase n) for "XML message (e.g. non-FIX MsgType)" can be used when transmitting XML content which is not defined with a FIX MsgType.

FIX MESSAGE DELIVERY

The following section summarizes general specifications for transmitting FIX messages.

Sequence Numbers:

All FIX messages are identified by a unique sequence number. Sequence numbers are initialized at the start of each FIX session (see Session Protocol section) starting at 1 (one) and increment throughout the session. Monitoring sequence numbers will enable parties to identify and react to missed messages and to gracefully synchronize applications when reconnecting during a FIX session.

Each session will establish an independent incoming and outgoing sequence series; participants will maintain a sequence series to assign to outgoing messages and a separate series to monitor for sequence gaps on incoming messages.

Heartbeats:

During periods of message inactivity, FIX applications will generate *Heartbeat* messages at regular time intervals. The heartbeat monitors the status of the communication link and identifies incoming sequence number gaps. The Heartbeat Interval is declared by the session initiator using the HeartBtInt field in the *Logon* message. The heartbeat interval timer should be reset after every message is transmitted (not just heartbeats). The HeartBtInt value should be agreed upon by the two firms and specified by the Logon initiator and echoed back by the Logon acceptor. Note that the same HeartBtInt value is used by both sides, the Logon "initiator" and Logon "acceptor".

Ordered Message Processing:

The FIX protocol assumes complete ordered delivery of messages between parties. Implementers should consider this when designing message gap fill processes. Two options exist for dealing with gaps, either request all messages subsequent to the last message received or ask for the specific message missed while maintaining an ordered list of all newer messages. For example, if the receiver misses the second of five messages, the application could ignore messages 3 through 5 and generate a resend request for messages 2 through 5, or, preferably 2 through 0 (where 0 represents infinity). Another option would involve saving messages 3 through 5 and resending only message 2. In both cases, messages 3 through 5 should not be processed before message 2.

Possible Duplicates:

When a FIX engine is unsure if a message was successfully received at its intended destination or when responding to a resend request, a possible duplicate message is generated. The message will be a retransmission (with the same sequence number) of the application data in question with the PossDupFlag included and set to "Y" in the header. It is the receiving application's responsibility to handle the message (i.e. treat as a new message or discard as appropriate). All messages created as the result of a resend request will contain the PossDupFlag field set to “Y”. Messages lacking the PossDupFlag field or with the PossDupFlag field set to “N” should be treated as original transmissions. *Note: When retransmitting a message with the PossDupFlag set to Y, it is always necessary to recalculate the CheckSum value. The only fields that can change in a possible duplicate message are the CheckSum, OrigSendingTime, SendingTime, BodyLength and PossDupFlag. Fields related to encryption (SecureDataLen and SecureData) may also require recasting.*

Possible Resends:

Ambiguous application level messages may be resent with the PossResend flag set. This is useful when an order remains unacknowledged for an inordinate length of time and the end-user suspects it had never been sent. The receiving application must recognize this flag and interrogate internal fields (order number, etc.) to determine if this order has been previously received. *Note: The possible resend message will contain exactly the same body data but will have the PossResend flag and will have a new sequence number. In addition the CheckSum field will require recalculation and fields related to encryption (SecureDataLen and SecureData) may also require recasting.*

Data Integrity:

The integrity of message data content can be verified in two ways: verification of message length and a simple checksum of characters.

The message length is indicated in the BodyLength field and is verified by counting the number of characters in the message following the BodyLength field up to, and including, the delimiter immediately preceding the CheckSum tag (“10=”).

The CheckSum integrity check is calculated by summing the binary value of each character from the “8” of “8=“ up to and including the <SOH> character immediately preceding the CheckSum tag field and comparing the least significant eight bits of the calculated value to the CheckSum value (see ***“CheckSum Calculation”*** for a complete description).

Message Acknowledgment:

The FIX session protocol is based on an optimistic model; normal delivery of data is assumed (i.e. no acknowledgment of individual messages) with errors in delivery identified by message sequence number gaps. Each message is identified by a unique sequence number. It is the receiving application's responsibility to monitor incoming sequence numbers to identify message gaps for response with resend request messages.

The FIX protocol does not support individual message acknowledgment. However, a number of application messages require explicit application level acceptance or rejection. Orders, cancel requests, cancel/replace requests, allocations, etc. require specific application level responses, executions can be rejected with the DK message but do not require explicit acceptance. See “Volume 1 - Business Message Reject” for details regarding the appropriate response message to specific application level messages.

Encryption:

The exchange of sensitive data across public carrier networks may make it advisable to employ data encryption techniques to mask the application messages.

The choice of encryption method will be determined by mutual agreement of the two parties involved in the connection.

Any field within a message can be encrypted and included in the SecureData field, however, certain explicitly identified fields must be transmitted unencrypted. The clear (unencrypted) fields can be repeated within the SecureData field to serve as an integrity check of the clear data.

When encryption is employed, it is recommended but not required that all fields within the message body be encrypted. If repeating groups are used within a message and encryption is applied to part of the repeating group, then the entire repeating group must be encrypted.

Embedded in the protocol are fields, which enable the implementation of a public key signature and encryption methodology, straight DES encryption and clear text. The previously agreed upon encryption methodology is declared in the *Logon* message. (For more detail on implementation of various encryption techniques see the application notes section on the FIX Web Site.)

SESSION PROTOCOL

A FIX session is defined as a bi-directional stream of ordered messages between two parties within a continuous sequence number series. A single FIX session can exist across multiple sequential (not concurrent) physical connections. Parties can connect and disconnect multiple times while maintaining a single FIX session. Connecting parties must bi-laterally agree as to when sessions are to be started/stopped based upon individual system and time zone requirements. Resetting the inbound and outbound sequence numbers back to 1, for whatever reason, constitutes the beginning of a new FIX session.

It is recommended that a new FIX session be established once within each 24 hour period. It is possible to maintain 24 hour connectivity and establish a new set of sequence numbers by sending a Logon message with the ResetSeqNumFlag set.

The FIX session protocol is based on an optimistic model. Normal delivery of data is assumed (i.e. no communication level acknowledgment of individual messages) with errors in delivery identified by message sequence number gaps. This section provides details on the implementation of the FIX session layer and dealing with message sequence gaps.

The following terms are used throughout this section:

- **Valid FIX Message** is a message that is properly formed according to this specification and contains a valid body length and checksum field
- **Initiator** establishes the telecommunications link and initiates the session via transmission of the initial *Logon* message.
- **Acceptor** is the receiving party of the FIX session. This party has responsibility to perform first level authentication and formally declare the connection request "accepted" through transmission of an acknowledgment *Logon* message.
- **FIX Connection is comprised of three parts: logon, message exchange, and logout.**
- **FIX Session is comprised of one or more FIX Connections, meaning that a FIX Session spans multiple logins.**

Logon -

Establishing a FIX connection involves three distinct operations: creation of a telecommunications level link, authentication/acceptance of the initiator by the acceptor and message synchronization (initialization). The sequence of connection follows:

- The session initiator establishes a telecommunication link with the session acceptor.
- The initiator sends a *Logon* message. The acceptor will authenticate the identity of the initiator by examining the *Logon* message. The *Logon* message will contain the data necessary to support the previously agreed upon authentication method. If the initiator is successfully authenticated, the acceptor responds with a *Logon* message. If authentication fails, the session acceptor should shut down the connection after optionally sending a Logout message to indicate the reason of failure. Sending a Logout in this case is not required because doing so would consume a sequence number for that session, which in some cases may be problematic. The session initiator may begin to send messages immediately following the *Logon* message, however, the acceptor may not be ready to receive them. The initiator must wait for the confirming *Logon* message from the acceptor before declaring the session fully established.

  After the initiator has been authenticated, the acceptor will respond immediately with a confirming *Logon* message. Depending on the encryption method being used for that session, this *Logon* message may or may not contain the same session encryption key. The initiator side will use the *Logon* message being returned from the acceptor as confirmation that a FIX session has been established. If the session acceptor has chosen to change the session encryption key, the session initiator must send a third *Logon* back to the other side in order to acknowledge the key change request. This also allows the session acceptor to know when the session initiator has started to encrypt using the new session key. Both parties are responsible for infinite loop detection and prevention during this phase of the session.

- **After authentication, the initiator and acceptor must synchronize their messages through interrogation of the *MsgSeqNum* field before sending any queued or new messages.** A comparison of the *MsgSeqNum* in the *Logon* message to the internally monitored next expected sequence number will indicate any message gaps. Likewise, the initiator can detect gaps by comparing the acknowledgment *Logon* message's *MsgSeqNum* to the next expected value. The section on message recovery later in this document deals with message gap handling.

- It is recommended to wait a short period of time following the Logon or to send a TestRequest and wait for a response to it before sending queued or new messages in order to allow both sides to handle resend request processing. Failure to do this could result in a ResendRequest message being issued by one's counterparty for each queued or new message sent. (see "Logon Message NextExpectedMsgSeqNum Processing" for an alternative approach)

- It is also recommended that an engine should store out of sequence messages in a temporary queue and process them in order when the gap is closed. This prevents generating resend requests for n->m, n->m+1, n->m+2, ... which can result in many resent PossDupFlag=Y messages.

- When using the ResetSeqNumFlag to maintain 24 hour connectivity and establish a new set of sequence numbers, the process should be as follows. Both sides should agree on a reset time and the party that will be the initiator of the process. Note that the initiator of the ResetSeqNum process may be different than the initiator of the Logon process. One side will initiate the process by sending a TestRequest and wait for a Heartbeat in response to ensure of no sequence number gaps. Once the Heartbeat has been received, the initiator should send a Logon with ResetSeqNumFlag set to Y and with MsgSeqNum of 1. The acceptor should respond with a Logon with ResetSeqNumFlag set to Y and with MsgSeqNum of 1. At this point new messages from either side should continue with MsgSeqNum of 2. It should be noted that once the initiator sends the Logon with the ResetSeqNumFlag set, the acceptor must obey this request and the message with the last sequence number transmitted "yesterday" may no longer be available. The connection should be shutdown and manual intervention taken if this process is initiated but not followed properly.

Message exchange -

After completion of the initialization process, normal message exchange begins. The formats for all valid messages are detailed in the sections 'Administrative Messages' and 'Application Messages'.

Logout -

Normal termination of the message exchange session will be completed via the exchange of *Logout* messages. Termination by other means should be considered an abnormal condition and dealt with as an error. Session termination without receiving a Logout should treat the counterparty as logged out.

It is recommended that before sending the Logout message, a TestRequest should be issued to force a Heartbeat from the other side. This helps to ensure that there are no sequence number gaps.

Before actually closing the session, the Logout initiator should wait for the opposite side to respond with a confirming Logout message. This gives the acceptor a chance to perform any Gap Fill operations that may be necessary. Once the messages from the ResendRequest have been received, the acceptor should issue the Logout. The session may be terminated if the acceptor does not respond in an appropriate timeframe.

Note: Logging out does not affect the state of any orders. All active orders will continue to be eligible for execution after logout.

Message Recovery -

During initialization, or in the middle of a FIX session, message gaps may occur which are detected via the tracking of incoming sequence numbers. The following section provides details on how to recover messages.

As previously stated, each FIX participant must maintain two sequence numbers for each FIX session, one each for incoming and outgoing messages which are initialized at '1' at the beginning of the FIX session. Each message is assigned a unique (by connection) sequence number, which is incremented after each message. Likewise, every message received has a unique sequence number and the incoming sequence counter is incremented after each message.

When the incoming sequence number does not match the expected number corrective processing is required. Note that the SeqReset-Reset message (used only to recover from a disaster scenario vs. normal resend request processing) is an exception to this rule as it should be processed without regards to its MsgSeqNum. **If the incoming message has a sequence number less than expected and the PossDupFlag is not set, it indicates a serious error. It is strongly recommended that the session be terminated and manual intervention be initiated.** If the incoming sequence number is greater than expected, it indicates that messages were missed and retransmission of the messages is requested via the *Resend Request* (see the earlier section, *Ordered Message Processing).*

Note: For the purposes of the following paragraphs *requester* refers to the party requesting the resend and *resender* refers to the party responding to the request. The process of resending and synchronizing messages is referred as "gap filling".

Upon receipt of a *Resend Request,* the resender can respond in one of three ways:

1. retransmit the requested messages (in order) with the original sequence numbers and *PossDupFlag* set to "Y" except for the administrative messages (listed below) which are not to be resent and which require a *SeqReset-GapFill (#2)*
2. issue a *SeqReset-GapFill with PossDupFlag set to "Y"* message to replace the retransmission of administrative and application messages
3. issue a *SeqReset-Reset with PossDupFlag set to "Y"* to force sequence number synchronization

The normal course of action involves a combination of #1 and #2. Note that #3 should be used ONLY to recover from a disaster situation which cannot be otherwise recovered via "Gap Fill" mode

During the gap fill process, certain administrative messages should not be retransmitted. Instead, a special *SeqReset-GapFill* message is generated. **The administrative messages which are not to be resent are: *Logon, Logout, ResendRequest, Heartbeat, TestRequest* and *SeqReset-Reset and SeqReset-GapFill.*** The *SeqReset-GapFill* can also be used to skip application messages that the sender chooses not to retransmit (e.g. aged orders). **This leaves Reject as the only administrative message which can be resent.**

All FIX implementations must monitor incoming messages to detect inadvertently retransmitted administrative messages (*PossDupFlag* flag set indicating a resend). When received, these messages should be processed for sequence number integrity only; the business/application processing of these message should be skipped (e.g. do not initiate gap fill processing based on a resent *ResendRequest).*

If there are consecutive administrative messages to be resent, it is suggested that only one *SeqReset-GapFill* message be sent in their place. The sequence number of the *SeqReset-GapFill* message is the next expected outbound sequence number. **The *NewSeqNo* field of the GapFill message contains the sequence number of the highest administrative message in this group plus 1.** For example, during a Resend operation there are 7 sequential administrative messages waiting to be resent. They start with sequence number 9 and end with sequence number 15. Instead of transmitting 7 Gap Fill messages (which is perfectly legal, but not network friendly), a *SeqReset-GapFill* message may be sent. **The sequence number of the Gap Fill message is set to 9 because the remote side is expecting that as the next sequence number.** The *NewSeqNo* field of the GapFill message contains the number 16, because that will be the sequence number of the next message to be transmitted.

Sequence number checking is a vital part of FIX session management. However, a discrepancy in the sequence number stream is handled differently for certain classes of FIX messages. The table below lists the actions to be taken when the incoming sequence number is greater than the expected incoming sequence number.

NOTE: **In *ALL* cases except the Sequence Reset - Reset message, the FIX session should be terminated if the incoming sequence number is less than expected and the PossDupFlag is not set. A *Logout* message with some descriptive text should be sent to the other side before closing the session.**

Response by Message Type

Message Type	**Action to Be Taken on Sequence # mismatch**
Logon	Must always be the first message transmitted. Authenticate and accept the connection. After sending a *Logon* confirmation back, send a *ResendRequest* if a message gap was detected in the *Logon* sequence number.
Logout	If a message gap was detected, issue a *ResendRequest* to retrieve all missing messages followed by a *Logout* message which serves as a confirmation of the logout request. **DO NOT** terminate the session. The initiator of the *Logout* sequence has responsibility to terminate the session. This allows the *Logout* initiator to respond to any *ResendRequest* message. If this side was the initiator of the *Logout* sequence, then this is a *Logout* confirmation and the session should be immediately terminated upon receipt. The only exception to the "do not terminate the session" rule is for an invalid Logon attempt. The session acceptor has the right to send a Logout message and terminate the session immediately. This minimizes the threat of unauthorized connection attempts.
ResendRequest	Perform the Resend processing first, followed by a *ResendRequest* of your own in order to fill the incoming message gap.
SeqReset-Reset	Ignore the incoming sequence number. The *NewSeqNo* field of the *SeqReset* message will contain the sequence number of the next message to be transmitted.
SeqReset-GapFill	Send a *ResendRequest* back. Gap Fill messages behave similar to a *SeqReset* message. However, it is important to insure that no messages have been inadvertently skipped over. This means that *GapFill* messages must be received in sequence. An out of sequence *GapFill* is an abnormal condition
All Other Messages	Perform Gap Fill operations.

Logon Message NextExpectedMsgSeqNum Processing -

The NextExpectedMsgSeqNum (789) field has been added in FIX 4.4 to the Logon message to support a proposed new way to resynchronize a FIX session. If accepted in the next major release of the protocol, its use will become mandatory. Pairs of counterparties may choose to adopt the new method in advance, but for now use of the field and implementation of the logic remains optional.

NextExpectedMsgSeqNum (789) is used as follows:

In its Logon request the session initiator supplies in NextExpectedMsgSeqNum (789) the value next expected from the session acceptor in MsgSeqNum (34). The outgoing header MsgSeqNum (34) of the Logon request is assigned the next-to-be-assigned sequence number as usual.

The session acceptor validates the Logon request including that NextExpectedMsgSeqNum (789) does not represent a gap. It then constructs its Logon response with NextExpectedMsgSeqNum (789) containing the value next expected from the session initiator in MsgSeqNum (34) having incremented the number above the Logon request if that was the sequence expected. The outgoing header MsgSeqNum (34) is constructed as usual.

The session initiator waits to begin sending application messages until it receives the Logon response. When it is received the initiator validates the response including that NextExpectedMsgSeqNum (789) does not represent a gap.

Both sides react to NextExpectedMsgSeqNum (789) from its counterparty thus:

- If equal to the next-to-be-assigned sequence, proceed sending new messages beginning with that number.
- If lower than the next-to-be-assigned sequence, "recover" (see "Message Recovery") all messages from the the last message delivered prior to this Logon through the specified NextExpectedMsgSeqNum (789) sending them in order; then Gap Fill over the sequence number used in Logon and proceed sending newly queued messages with a sequence number one higher than the original Logon.
- If higher than the next-to-be-assigned sequence, send Logout to abort the session.

Neither side should generate a ResendRequest based on MsgSeqNum (34) of the incoming Logon message but should expect any gaps to be filled automatically. If a gap is produced by the Logon message MsgSeqNum (34), the receive logic should expect the gap to be filled automatically prior to receiving any messages with sequences above the gap.

Standard Message header

Each administrative or application message is preceded by a standard header. The header identifies the message type, length, destination, sequence number, origination point and time.

Two fields help with resending messages. The PossDupFlag is set to Y when resending a message as the result of a session level event (i.e. the retransmission of a message reusing a sequence number). The PossResend is set to Y when reissuing a message with a new sequence number (e.g. resending an order). The receiving application should process these messages as follows:

> PossDupFlag - if a message with this sequence number has been previously received, ignore message, if not, process normally.
>
> PossResend - forward message to application and determine if previously received (i.e. verify order id and parameters).

Message Routing Details – One Firm-to-One Firm (point-to-point)

The following table provides examples regarding the use of SenderCompID, TargetCompID, DeliverToCompID, and OnBehalfOfCompID when using a single point-to-point FIX session between two firms. Assumption (A=sellside, B =buyside):

	SenderCompID	**OnBehalfOfCompID**	**TargetCompID**	**DeliverToCompID**
A to B directly	A		B	
B to A directly	B		A	

Message Routing Details – Third Party Message Routing

The FIX Session Protocol supports the ability for a single FIX session to represent multiple counterpaties. This can be in a 1-to-many, many-to-1, or 1-to-1 fashion. In addition, some third parties may be connected to other third parties effectively forming a “chain” of “hops” between the original message initiator and the final message receiver. The SenderCompID, OnBehalfOfCompID, TargetCompID, and DeliverToCompID fields are used for routing purposes.

When a third party sends a message on behalf of another firm (using OnBehalfOfCompID), that third party may optionally add their details to the NoHops repeating group. This repeating group builds a “history” of third parties through which the original message was re-transmitted. The NoHops repeating group is NOT used to facilitate routing, rather it provides an audit trail of third party involvement to the receiver of a message. An audit trail of intermediary involvement may be a requirement of some regulatory bodies or counterparties. When a third party forwards a message on to the next hop (may be the end point or another third party), that third party can add its hop details to the NoHops repeating group (e.g. its SenderCompID as HopCompID, its SendingTime as HopSendingTime, and the received message’s MsgSeqNum or some other reference as HopRefID).

Note that if OnBehalfOfCompID or DeliverToCompID message source identification/routing is used for a FIX session, then it must be used on all Application messages transmitted via that session accordingly (Reject the message if not).

The following table provides examples regarding the use of SenderCompID, TargetCompID, DeliverToCompID, and OnBehalfOfCompID when using a single FIX session to represent multiple firms. Assumption (A=sellside, B and C=buyside, Q=third party):

	SenderCompID	**OnBehalfOf CompID**	**TargetCompID**	**DeliverTo CompID**	**HopCompID**	**HopSendingTime**
Send from A to B via Q						
1) A sends to Q	A		Q	B		

2)	Q sends to B	Q	A	B		Q	A’s SendingTime
B responds to A via Q							
1)	B sends to Q	B		Q	A		
2)	Q sends to A	Q	B	A		Q	B’s SendingTime
Send from A to B *AND* C via Q							
1)	A sends to Q	A		Q	B		
2)	Q sends to B	Q	A	B		Q	A’s SendingTime
3)	A sends to Q	A		Q	C		
4)	Q sends to C	Q	A	C		Q	A’s SendingTime
B *AND* C send to A via Q							
1)	B sends to Q	B		Q	A		
2)	Q sends to A	Q	B	A		Q	B’s SendingTime
3)	C sends to Q	C		Q	A		
4)	Q sends to A	Q	C	A		Q	C’s SendingTime

Note that some fields (e.g. ClOrdID on a New Order Single) must be unique for all orders on a given FIX session. Thus when using OnBehalfOfCompID (or DeliverToCompID) addressing, a recommended approach is to prepend OnBehalfOfCompID (or DeliverToCompID) to the original value. Thus if A sends Q ClOrdID value of "123", then Q could specify ClOrdID of "A-123" when sending the message to C to ensure uniqueness.

The standard message header format is as follows:

Standard Message Header

Tag	*Field Name*	*Req'd*	*Comments*
8	BeginString	Y	FIX.4.4 (*Always unencrypted, must be first field in message*)
9	BodyLength	Y	(*Always unencrypted, must be second field in message*)
35	MsgType	Y	(*Always unencrypted, must be third field in message*)
49	SenderCompID	Y	(*Always unencrypted*)
56	TargetCompID	Y	(*Always unencrypted*)
115	OnBehalfOfCompID	N	Trading partner company ID used when sending messages via a third party *(Can be embedded within encrypted data section.)*
128	DeliverToCompID	N	Trading partner company ID used when sending messages via a third party *(Can be embedded within encrypted data section.)*
90	SecureDataLen	N	Required to identify length of encrypted section of message. (*Always unencrypted*)

91	SecureData	N	Required when message body is encrypted. Always immediately follows SecureDataLen field.
34	MsgSeqNum	Y	*(Can be embedded within encrypted data section.)*
50	SenderSubID	N	*(Can be embedded within encrypted data section.)*
142	SenderLocationID	N	Sender's LocationID (i.e. geographic location and/or desk) *(Can be embedded within encrypted data section.)*
57	TargetSubID	N	“ADMIN” reserved for administrative messages not intended for a specific user. *(Can be embedded within encrypted data section.)*
143	TargetLocationID	N	Trading partner LocationID (i.e. geographic location and/or desk) *(Can be embedded within encrypted data section.)*
116	OnBehalfOfSubID	N	Trading partner SubID used when delivering messages via a third party. *(Can be embedded within encrypted data section.)*
144	OnBehalfOfLocationID	N	Trading partner LocationID (i.e. geographic location and/or desk) used when delivering messages via a third party. *(Can be embedded within encrypted data section.)*
129	DeliverToSubID	N	Trading partner SubID used when delivering messages via a third party. *(Can be embedded within encrypted data section.)*
145	DeliverToLocationID	N	Trading partner LocationID (i.e. geographic location and/or desk) used when delivering messages via a third party. *(Can be embedded within encrypted data section.)*
43	PossDupFlag	N	Always required for retransmitted messages, whether prompted by the sending system or as the result of a resend request. *(Can be embedded within encrypted data section.)*
97	PossResend	N	Required when message may be duplicate of another message sent under a different sequence number. *(Can be embedded within encrypted data section.)*
52	SendingTime	Y	*(Can be embedded within encrypted data section.)*
122	OrigSendingTime	N	Required for message resent as a result of a ResendRequest. If data is not available set to same value as SendingTime *(Can be embedded within encrypted data section.)*
212	XmlDataLen	N	Required when specifying XmlData to identify the length of a XmlData message block. *(Can be embedded within encrypted data section.)*
213	XmlData	N	Can contain a XML formatted message block (e.g. FIXML). Always immediately follows XmlDataLen field. *(Can be embedded within encrypted data section.)* See Volume 1: FIXML Support
347	MessageEncoding	N	Type of message encoding (non-ASCII characters) used in a message’s “Encoded” fields. Required if any “Encoding” fields are used.
369	LastMsgSeqNumProcessed	N	The last MsgSeqNum value received by the FIX engine and processed by downstream application, such as trading system or order routing system. Can be specified on every message sent. Useful for detecting a backlog with a counterparty.

627	NoHops		N	Number of repeating groups of historical “hop” information. Only applicable if OnBehalfOfCompID is used, however, its use is optional. Note that some market regulations or counterparties may require tracking of message hops.
→	***628***	***HopCompID***	N	Third party firm which delivered a specific message either from the firm which originated the message or from another third party (if multiple “hops” are performed). It is recommended that this value be the SenderCompID (49) of the third party.
→	***629***	***HopSendingTime***	N	Time that HopCompID (628) sent the message. It is recommended that this value be the SendingTime (52) of the message sent by the third party.
→	***630***	***HopRefID***	N	Reference identifier assigned by HopCompID (628) associated with the message sent. It is recommended that this value be the MsgSeqNum (34) of the message sent by the third party.

Standard Message trailer

Each message, administrative or application, is terminated by a standard trailer. The trailer is used to segregate messages and contains the three digit character representation of the Checksum value.

The standard message trailer format is as follows:

Standard Message Trailer

Tag	*Field Name*	*Req'd*	*Comments*
93	SignatureLength	N	Required when trailer contains signature. *Note: Not to be included within SecureData field*
89	Signature	N	*Note: Not to be included within SecureData field*
10	CheckSum	Y	(*Always unencrypted, always last field in message*)

ADMINISTRATIVE MESSAGES

The administrative messages address the utility needs of the protocol. The following section describes each message and provides the message layout.

Administrative messages will be generated from both sides of the connection.

Heartbeat -

The Heartbeat monitors the status of the communication link and identifies when the last of a string of messages was not received.

When either end of a FIX connection has not sent any data for [HeartBtInt] seconds, it will transmit a Heartbeat message. When either end of the connection has not received any data for (HeartBtInt + “some reasonable transmission time”) seconds, it will transmit a Test Request message. If there is still no Heartbeat message received after (HeartBtInt + “some reasonable transmission time”) seconds then the connection should be considered lost and corrective action be initiated. If HeartBtInt is set to zero then no regular heartbeat messages will be generated. Note that a test request message can still be sent independent of the value of the HeartBtInt, which will force a Heartbeat message.

Heartbeats issued as the result of Test Request must contain the TestReqID transmitted in the Test Request message. This is useful to verify that the Heartbeat is the result of the Test Request and not as the result of a regular timeout.

The heartbeat format is as follows:

Heartbeat

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 0
112	TestReqID	N	Required when the heartbeat is the result of a Test Request message.
	Standard Trailer	Y	

Logon -

The logon message authenticates a user establishing a connection to a remote system. The logon message must be the first message sent by the application requesting to initiate a FIX session.

The HeartBtInt (108) field is used to declare the timeout interval for generating heartbeats (same value used by both sides). The HeartBtInt value should be agreed upon by the two firms and specified by the Logon initiator and echoed back by the Logon acceptor.

Upon receipt of a Logon message, the session acceptor will authenticate the party requesting connection and issue a Logon message as acknowledgment that the connection request has been accepted. The acknowledgment Logon can also be used by the initiator to validate that the connection was established with the correct party.

The session acceptor must be prepared to immediately begin processing messages after receipt of the Logon. The session initiator can choose to begin transmission of FIX messages before receipt of the confirmation Logon, however it is recommended that normal message delivery wait until after the return Logon is received to accommodate encryption key negotiation.

The confirmation Logon can be used for encryption key negotiation. If a session key is deemed to be weak, a stronger session key can be suggested by returning a Logon message with a new key. This is only valid for encryption protocols that allow for key negotiation. (See the FIX Web Site's Application notes for more information on a method for encryption and key passing.)

The Logon message can be used to specify the MaxMessageSize supported (e.g. can be used to control fragmentation rules for very large messages which support fragmentation). It can also be used to specify the MsgTypes supported for both sending and receiving.

The logon format is as follows:

Logon

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = A
98	EncryptMethod	Y	(*Always unencrypted*)
108	HeartBtInt	Y	Note same value used by both sides
95	RawDataLength	N	Required for some authentication methods
96	RawData	N	Required for some authentication methods
141	ResetSeqNumFlag	N	Indicates both sides of a FIX session should reset sequence numbers
789	NextExpectedMsgSeqNum	N	Optional, alternative via counterparty bi-lateral agreement message gap detection and recovery approach (see "Logon Message NextExpectedMsgSeqNum Processing" section)
383	MaxMessageSize	N	Can be used to specify the maximum number of bytes supported for messages received
384	NoMsgTypes	N	Specifies the number of repeating RefMsgTypes specified
→	***372*** ***RefMsgType***	N	Specifies a specific, supported MsgType. Required if NoMsgTypes is > 0. Should be specified from the point of view of the sender of the Logon message
→	***385*** ***MsgDirection***	N	Indicates direction (send vs. receive) of a supported MsgType. Required if NoMsgTypes is > 0. Should be specified from the point of view of the sender of the Logon message
464	TestMessageIndicator	N	Can be used to specify that this FIX session will be sending and receiving "test" vs. "production" messages.

553	Username	N	
554	Password	N	Note: minimal security exists without transport-level encryption.
	Standard Trailer	Y	

Test Request -

The test request message forces a heartbeat from the opposing application. The test request message checks sequence numbers or verifies communication line status. The opposite application responds to the Test Request with a Heartbeat containing the TestReqID.

The TestReqID verifies that the opposite application is generating the heartbeat as the result of Test Request and not a normal timeout. The opposite application includes the TestReqID in the resulting Heartbeat. Any string can be used as the TestReqID (one suggestion is to use a timestamp string).

The test request format is as follows:

Test Request

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 1
112	TestReqID	Y	
	Standard Trailer	Y	

Resend Request -

The resend request is sent by the receiving application to initiate the retransmission of messages. This function is utilized if a sequence number gap is detected, if the receiving application lost a message, or as a function of the initialization process.

The resend request can be used to request a single message, a range of messages or all messages subsequent to a particular message.

Note: the sending application may wish to consider the message type when resending messages; e.g. if a new order is in the resend series and a significant time period has elapsed since its original inception, the sender may not wish to retransmit the order given the potential for changed market conditions. (The Sequence Reset-GapFill message is used to skip messages that a sender does not wish to resend.)

Note: it is imperative that the receiving application process messages in sequence order, e.g. if message number 7 is missed and 8-9 received, the application should ignore 8 and 9 and ask for a resend of 7-9, or, preferably, 7-0 (0 represents infinity). This latter approach is strongly recommended to recover from out of sequence conditions as it allows for faster recovery in the presence of certain race conditions when both sides are simultaneously attempting to recover a gap.

- To request a single message: BeginSeqNo = EndSeqNo
- To request a range of messages: BeginSeqNo = first message of range, EndSeqNo = last message of range
- To request all messages subsequent to a particular message: BeginSeqNo = first message of range, EndSeqNo = 0 (represents infinity) .

The resend request format is as follows:

Resend Request

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 2
7	BeginSeqNo	Y	
16	EndSeqNo	Y	
	Standard Trailer	Y	

Reject (session-level) -

The reject message should be issued when a message is received but cannot be properly processed due to a session-level rule violation. An example of when a reject may be appropriate would be the receipt of a message with invalid basic data (e.g. MsgType=&) which successfully passes de-encryption, CheckSum and BodyLength checks. As a rule, messages should be forwarded to the trading application for business level rejections whenever possible.

Rejected messages should be logged and the incoming sequence number incremented.

Note: The receiving application should disregard any message that is garbled, cannot be parsed or fails a data integrity check. Processing of the next valid FIX message will cause detection of a sequence gap and a Resend Request will be generated. Logic should be included in the FIX engine to recognize the possible infinite resend loop, which may be encountered in this situation.

Generation and receipt of a Reject message indicates a serious error that may be the result of faulty logic in either the sending or receiving application.

If the sending application chooses to retransmit the rejected message, it should be assigned a new sequence number and sent with PossResend=Y.

Whenever possible, it is strongly recommended that the cause of the failure be described in the Text field (e.g. INVALID DATA - FIELD 35).

If an application-level message received fulfills session-level rules, it should then be processed at a business message-level. If this processing detects a rule violation, a business-level reject should be issued. Many business-level messages have specific "reject" messages, which should be used. All others can be rejected at a business-level via the Business Message Reject message. See Volume 1: "Business Message Reject" message.

Note that in the event a business message is received, fulfills session-level rules, however, the message cannot be communicated to the business-level processing system, a Business Message Reject with BusinessRejectReason = "Application not available at this time" should be issued.

Scenarios for session-level Reject:

SessionRejectReason
0 = Invalid tag number
1 = Required tag missing
2 = Tag not defined for this message type
3 = Undefined Tag
4 = Tag specified without a value
5 = Value is incorrect (out of range) for this tag
6 = Incorrect data format for value
7 = Decryption problem
8 = Signature problem
9 = CompID problem
10 = SendingTime accuracy problem
11 = Invalid MsgType
12 = XML Validation error
13 = Tag appears more than once
14 = Tag specified out of required order
15 = Repeating group fields out of order

16 = Incorrect NumInGroup count for repeating group
17 = Non “data” value includes field delimiter (SOH character)
99 = Other
(Note other session-level rule violations may exist in which case SessionRejectReason of Other may be used and further information may be in Text field.)

The reject format is as follows:

Reject

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 3
45	RefSeqNum	Y	MsgSeqNum of rejected message
371	RefTagID	N	The tag number of the FIX field being referenced.
372	RefMsgType	N	The MsgType of the FIX message being referenced.
373	SessionRejectReason	N	Code to identify reason for a session-level Reject message.
58	Text	N	Where possible, message to explain reason for rejection
354	EncodedTextLen	N	Must be set if EncodedText field is specified and must immediately precede it.
355	EncodedText	N	Encoded (non-ASCII characters) representation of the Text field in the encoded format specified via the MessageEncoding field.
	Standard Trailer	Y	

Sequence Reset (Gap Fill) -

The Sequence Reset message has two modes: **Gap Fill mode** and **Reset mode**.

Gap Fill mode

Gap Fill mode is used in response to a Resend Request when one or more messages must be skipped over for the following reasons:

- During normal resend processing, the sending application may choose not to send a message (e.g. an aged order).
- During normal resend processing, a number of administrative messages are skipped and not resent (such as Heart Beats, Test Requests).

Gap Fill mode is indicated by GapFillFlag (tag 123) field = "Y".

If the GapFillFlag field is present (and equal to "Y"), the MsgSeqNum should conform to standard message sequencing rules (i.e. the MsgSeqNum of the Sequence Reset GapFill mode message should represent the beginning MsgSeqNum in the GapFill range because the remote side is expecting that next message sequence number).

Reset mode

Reset mode involves specifying an arbitrarily higher new sequence number to be expected by the receiver of the Sequence Reset-Reset message, and is used to reestablish a FIX session after an unrecoverable application failure.

Reset mode is indicated by the GapFillFlag (tag 123) field = "N" or if the field is omitted.

If the GapFillFlag field is not present (or set to N), it can be assumed that the purpose of the Sequence Reset message is to recover from an out-of-sequence condition. In Sequence Reset - Reset mode, the MsgSeqNum in the header should be ignored (i.e. the receipt of a Sequence Reset - Reset mode message with an out of sequence MsgSeqNum should not generate resend requests**)**. ***Sequence Reset – Reset should NOT be used as a normal response to a Resend Request (use Sequence Reset – Gap Fill mode). The Sequence Reset – Reset should ONLY be used to recover from a disaster situation which cannot be recovered via the use of Sequence Reset – Gap Fill. Note that the use of Sequence Reset – Reset may result in the possibility of lost messages.***

Rules for processing all Sequence Reset messages

The sending application will initiate the Sequence Reset. **The message in all situations specifies NewSeqNo to reset to as the value of the next sequence number** to be expected by the message receipient **immediately following the messages and/or sequence numbers being skipped**.

The Sequence Reset can only increase the sequence number. If a sequence reset is received attempting to decrease the next expected sequence number the message should be rejected and treated as a serious error. It is possible to have multiple Resend Requests issued in a row (i.e. 5 to 10 followed by 5 to 11). If sequence number 8, 10, and 11 represent application messages while the 5-7 and 9 represent administrative messages, the series of messages as result of the Resend Request may appear as Sequence Reset-GapFill mode with NewSeqNo of 8, message 8, Sequence Reset-GapFill with NewSeqNo of 10, and message 10. This could then followed by Sequence Reset-GapFill with NewSeqNo of 8, message 8, Sequence Reset-GapFill with NewSeqNo of 10, message 10, and message 11. One must be careful to ignore the duplicate Sequence Reset-GapFill mode which is attempting to lower the next expected sequence number. This can be detected by checking to see if its MsgSeqNum is less than expected. If so, the Sequence Reset-GapFill mode is a duplicate and should be discarded.

~~The sequence reset message is used by the sending application to reset the incoming sequence number on the opposing side. This message has two modes: "Sequence Reset-Gap Fill" when GapFillFlag is 'Y' and "Sequence Reset-Reset" when GapFillFlag is N or not present. The "Sequence Reset-Reset" mode should~~ ~~**ONLY**~~ ~~be used to recover from a disaster situation which cannot be otherwise recovered via "Gap Fill" mode. The sequence reset message can be used in the following situations:~~

- ~~During normal resend processing, the sending application may choose not to send a message (e.g. an aged order). The Sequence Reset – Gap Fill is used to mark the place of that message.~~

- ~~During normal resend processing, a number of administrative messages are not resent, the Sequence Reset – Gap Fill message is used to fill the sequence gap created.~~

- ~~In the event of an application failure, it may be necessary to force synchronization of sequence numbers on the sending and receiving sides via the use of Sequence Reset - Reset~~

~~The sending application will initiate the sequence reset.~~ ~~**The message in all situations specifies NewSeqNo to reset as the value of the next sequence number immediately following the messages and/or sequence numbers being skipped**.~~

~~If the GapFillFlag field is not present (or set to N), it can be assumed that the purpose of the sequence reset message is to recover from an out-of-sequence condition. The MsgSeqNum in the header should be ignored (i.e. the receipt of a Sequence Reset - Reset message with an out of sequence MsgSeqNum should not generate resend requests)***. Sequence Reset – Reset should NOT be used as a normal response to a Resend Request (use Sequence Reset – Gap Fill). The Sequence Reset – Reset should ONLY be used to recover from a disaster situation which cannot be recovered via the use of Sequence Reset – Gap Fill. Note that the use of Sequence Reset – Reset may result in the possibility of lost messages***~~

~~If the GapFillFlag field is present (and equal to Y), the MsgSeqNum should conform to standard message sequencing rules (i.e. the MsgSeqNum of the Sequence Reset-GapFill message should represent the beginning MsgSeqNum in the GapFill range because the remote side is expecting that next message sequence number).~~

~~The sequence reset can only increase the sequence number. If a sequence reset is received attempting to decrease the next expected sequence number the message should be rejected and treated as a serious error. It is possible to have multiple ResendRequests issued in a row (i.e. 5 to 10 followed by 5 to 11). If sequence number 8, 10, and 11 represent application messages while the 5-7 and 9 represent administrative messages, the series of messages as result of the Resend Request may appear as SeqReset-GapFill with NewSeqNo of 8, message 8, SeqReset-GapFill with NewSeqNo of 10, and message 10. This could then followed by SeqReset-GapFill with NewSeqNo of 8, message 8, SeqReset-GapFill with NewSeqNo of 10, message 10, and message 11. One must be careful to ignore the duplicate SeqReset-GapFill which is attempting to lower the next expected sequence number. This can be detected by checking to see if its MsgSeqNum is less than expected. If so, the SeqReset-GapFill is a duplicate and should be discarded.~~

The Sequence Reset format is as follows:

Sequence Reset

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 4
123	GapFillFlag	N	
36	NewSeqNo	Y	
	Standard Trailer	Y	

Logout -

The logout message initiates or confirms the termination of a FIX session. Disconnection without the exchange of logout messages should be interpreted as an abnormal condition.

Before actually closing the session, the logout initiator should wait for the opposite side to respond with a confirming logout message. This gives the remote end a chance to perform any Gap Fill operations that may be necessary. The session may be terminated if the remote side does not respond in an appropriate timeframe.

After sending the Logout message, the logout initiator should not send any messages unless requested to do so by the logout acceptor via a ResendRequest.

The logout format is as follows:

Logout

Tag	*Field Name*	*Req'd*	*Comments*
	Standard Header	Y	MsgType = 5
58	Text	N	
354	EncodedTextLen	N	Must be set if EncodedText field is specified and must immediately precede it.
355	EncodedText	N	Encoded (non-ASCII characters) representation of the Text field in the encoded format specified via the MessageEncoding field.
	Standard Trailer	Y	

CheckSum Calculation

The checksum of a FIX message is calculated by summing every byte of the message up to but not including the checksum field itself. This checksum is then transformed into a modulo 256 number for transmission and comparison. The checksum is calculated after all encryption is completed, i.e. the message as transmitted between parties is processed.

For transmission, the checksum must be sent as printable characters, so the checksum is transformed into three ASCII digits.

For example, if the checksum has been calculated to be 274 then the modulo 256 value is 18 (256 + 18 = 274). This value would be transmitted a |10=018| where "10="is the tag for the checksum field.

A sample code fragment to generate the checksum field is as follows:

```
char *GenerateCheckSum( char *buf, long bufLen )
{
    static char tmpBuf[ 4 ];
    long idx;
    unsigned int cks;

    for( idx = 0L, cks = 0; idx < bufLen; cks += (unsigned int)buf[ idx++ ] );
    sprintf( tmpBuf, “%03d”, (unsigned int)( cks % 256 ) );
    return( tmpBuf );
}
```

FIX Session Using a Multicast Transport

Justification

The FIX Protocol is made up of a session layer, an application layer and a field reference (data dictionary). The latter two have usefulness independent of the FIX session. Furthermore, because the FIX session is meant for point-to-point communication, it does not lend itself to publish/subscribe models well (e.g. providing market data to numerous receivers). This application note defines how a FIX messages can be distributed over a Multicast transport (e.g. IP Multicast). This note does not go into details of a particular multicast technology, but rather discusses how a modified FIX session can be implemented over it.

Overview of Session Layer

To ensure proper detection of gaps in messages in a multicast environment, except as noted below there should only be one publisher of a particular subject. For the purpose of this document, “subject” refers to how or what a receiver listens for and publisher sends information on (e.g. a subject name, a TCP listen port, etc). The characteristics and features of a particular multicast technology as well as the implementation (e.g. the primary publisher using one subject to accept resend requests from mulitple primary receivers publishing the request) will control whether or not messages sent by a particular publisher can be expected to be delivered in order. There can be any number of subscribers or receivers of a particular subject.

In order for a message gap detecting session layer to exist, the sequence number that is placed on each message should be assigned on a per subject basis. The consequence of not assigning on a per subject basis is that the receivers (who might not subscribe to all subjects) would end up detecting gaps where gaps really do not exist for that subject. Thus, receivers must maintain expected sequence numbers on a per subject basis.

Unlike the standard FIX session layer (in FIX 4.0 and above), all of the administrative messages will not consume a sequence number (as was the case in FIX 3.0). The type of data transmitted in a multicast environment is one-way (from publisher to subscriber) and thus race conditions that are possible in a standard two-way FIX 3.0 session are not applicable here. As such the requirement for administrative messages consuming a sequence number (FIX.4.0 and greater) can be relaxed. In this case administrative messages should contain the MsgSeqNum of the “next” message to be sent but not “consume” or increment this number (this is consistent with FIX 3.0). Any business-level message which is sent should increment the next outbound sequence number to allow receivers to detect gaps and perform gap processing.

The TargetCompID on all messages should be set to some predefined value (assigned by the publisher) such as the subject the message is being published out under. The SenderCompID on all messages should be set to some predefined, static value assigned by the publisher which identifies the publisher.

Logon

Different from the normal FIX session two-way logon, the logon in a pub/sub or multicast environment should be one-way from the publisher to the subscribers. It is used to notify subscribers that the publisher’s session has begun.

Heartbeats

Heartbeats are used as keep-alive packets in periods of application message inactivity. These messages should only be transmitted by the publisher.

Resend Requests

If a receiver detects a sequence gap in the messages being sent by the publisher, it can transmit a Resend Request on a separate well defined Resend Request subject. Upon receiving a Resend Request, the publisher can choose to respond immediately, to schedule responses or to not satisfy the request at all. Since Rejects messages are not used in a multicast environment, the publisher should ignore invalid resend requests (e.g. seq num too high or duplicates). If when responding to a Resend Request, the publisher wishes to skip over one or more messages, it should transmit a SequenceReset/GapFill message to indicate that the gap should be artificially filled. Because the publisher may not respond immediately, care should be taken on the receiving application to decide if the need for ordered delivery should delay processing of subsequent messages.

If multiple application subjects are being published, either an individual Resend Request subject should be defined for each application subject or Resend Requests should somehow identify the desired application subject via the TargetSubID field in order for the application subject publisher to identify which subject the request is for. In addition, it is also possible to establish a separate subject for receivers to listen for responses to Resend Requests on rather than the primary message subject.

Some multicast technologies (e.g. IP Multicast) may not allow for the ability of a publisher to respond to a specific subscriber's resend request without also providing that same response to **all** subscribers via the "primary" subject distribution channel. Subscribers must be capable of disgarding messages which contain lower than expected sequence numbers.

Rejects

Session level rejects messages are not applicable or used in this model.

Logout

Different from the FIX session two-way logout, the logout in a multicast environment should be one-way from the publisher to the subscribers. It is used to notify subscribers that the session has ended.

Additional Implementation Details

Additional implementation details concerning an actual business implementation of a multicast environment should be documented and agreed to by the parties involved.

FIX Session-level Test Cases and Expected Behaviors

Applicability

This document was last revised September 20, 2002 at which time FIX version 4.3 with Errata 20020930 was the latest version of the FIX Protocol. Note that future amendments to this document may be found on the FIX website and any version of this document published on a later date takes precedence over this version of the document. This document is applicable to all versions of FIX 4.X (4.0, 4.1, 4.2, 4.3, and 4.4) except where explicitly indicated.

When to send a Logout vs. when to just disconnect

In general a Logout message should always be sent prior to shutting down a connection. If the Logout is being sent due to an error condition, the Text field of the Logout should provide a descriptive reason, so that operational support of the remote FIX system can diagnosis the problem. There are exceptions, when it is recommended that a Logout message not be sent, these include:

- If during a logon either the SenderCompID, TargetCompID or IP address of the session initiator is invalid, it is recommended that the session be immediately terminated and no Logout message sent. This login attempt might be an unauthorized attempt to break into your system; hence one does not want to divulge any information about one's FIX system, such as: which SenderCompID/TargetCompID values are valid or which version of FIX is supported.
- If during a Logon one receives a second connection attempt while a valid FIX session is already underway for that same SenderCompID, it is recommended that the session acceptor immediately terminate the second connection attempt and not send a Logout message. Sending a Logout message runs the risk of interfering with and possibly adversely affecting the current active FIX connection. For example, in some FIX system implementations, sending a Logout message might consume a sequence number that would cause an out of sequence condition for the established FIX session.

In all other cases, if sending a Logout does not create risk or violate security, a Logout message should be sent with a descriptive text message.

When to send a Session Reject vs. when to ignore the message

The following excerpt is taken from the Reject message definition within the FIX Protocol specification:

> *Note: The receiving application should disregard any message that is garbled, cannot be parsed or fails a data integrity check. Processing of the next valid FIX message will cause detection of a sequence gap and a Resend Request will be generated. Logic should be included in the FIX engine to recognize the possible infinite resend loop, which may be encountered in this situation.*

The FIX Protocol takes the optimistic view; it presumes that a garbled message is received due to a transmission error rather than a FIX system problem. Therefore, if a Resend Request is sent the garbled message will be retransmitted correctly. If a message is not considered garbled then it is recommended that a session level Reject message be sent.

What constitutes a garbled message

- BeginString (tag #8) is not the first tag in a message or is not of the format 8=FIX.n.m.
- BodyLength (tag #9) is not the second tag in a message or does not contain the correct byte count.
- MsgType (tag #35) is not the third tag in a message.
- Checksum (tag #10) is not the last tag or contains an incorrect value.

If the MsgSeqNum(tag #34) is missing a logout message should be sent terminating the FIX Connection, as this indicates a serious application error that is likely only circumvented by software modification.

FIX Session-level State Matrix

Precedence	State	Initiator	Acceptor	Description
1	Disconnected-No Connection Today	Y	Y	Currently disconnected, have not attempted to establish a connection “today”, and no MsgSeqNum have been consumed (next connection “today” will start at *MsgSeqNum* of 1)
2	Disconnected-Connection Today	Y	Y	Currently disconnected, have attempted to establish a connection “today” and thus MsgSeqNum have been consumed (next connection “today” will start at *MsgSeqNum* of (last + 1))
3	Detect Broken Network Connection	Y	Y	While connected, detect a broken network connection (e.g. TCP socket closed). Disconnect the network connection and “shutdown” configuration for this session.
4	Awaiting Connection	N	Y	Session acceptor Logon awaiting network connection from counterparty
5	Initiate Connection	Y	N	Session initiator Logon establishing network connection with counterparty
6	Network Connection Established	Y	Y	Network connection established between both parties
7	Initiation **Logon** Sent	Y	N	Session initiator Logon send **Logon** message. *** Exception: 24hr sessions.
8	Initiation **Logon** Received	N	Y	Session acceptor Logon receive counterparty’s **Logon** message. *** Exception: 24hr sessions.
9	Initiation **Logon** Response	N	Y	Session acceptor Logon respond to counterparty’s **Logon** message with **Logon** message to handshake
10	Handle **ResendRequest**	Y	Y	Receive and respond to counterparty’s **ResendRequest** sending requested messages and/or **SequenceReset-Gap Fill** messages for the range of *MsgSeqNum* requested. Updated to include rejecting Resend Request received with MsgSeqNum that is <= LastSeqNum processed.
11	Receive *MsgSeqNum* Too High	Y	Y	Receive too high of *MsgSeqNum* from counterparty, queue message, and send **ResendRequest**

12	Awaiting/Processing Response to **ResendRequest**	Y	Y	Process requested *MsgSeqNum PossDupFlag*=Y resent messages and/or **SequenceReset-Gap Fill** messages from counterparty. Queue incoming messages with *MsgSeqNum* too high
13	No messages received in Interval	Y	Y	No inbound messages (non-garbled) received in (HeartBeatInt + “reasonable period of time”), send **TestRequest**
14	Awaiting/Processing Response to **TestRequest**	Y	Y	Process inbound messages. Reset heart beat interval-related timer when ANY inbound message (non-garbled) is received
15	Receive **Logout** message	Y	Y	Receive **Logout** message from counterparty initiating logout/disconnect. If *MsgSeqNum* too high, send **ResendRequest**. If sent, wait a reasonable period of time for complete response to **ResendRequest**. Note that depending upon the reason for the **Logout**, the counterparty may be unable to fulfill the request. Send **Logout** message as response and wait a reasonable period of time for counterpaty to disconnect the network connection. Note counterparty may send a **ResendRequest** message if **Logout** message response has *MsgSeqNum* too high and then re-initiate the **Logout** process.
16	Initiate **Logout** Process	Y	Y	Identify condition or reason to gracefully disconnect (e.g. end of “day”, no response after multiple **TestRequest** messages, too low *MsgSeqNum*, etc.). Send **Logout** message to counterparty. Wait a reasonable period of time for **Logout** response. During this time handle “new” inbound messages and/or **ResendRequest** if possible. Note that some logout/termination conditions (e.g. loss of database/message safe-store) may require immediate termination of the network connection following the initial send of the **Logout** message. Disconnect the network connection and “shutdown” configuration for this session.
17	Active/Normal Session	Y	Y	Network connection established, **Logon** message exchange successfully completed, inbound and outbound *MsgSeqNum* are in sequence as expected, and **Heartbeat** or other messages are received within (*HeartBeatInt* + “reasonable period of time”).
18	Waiting for **Logon** ack	Y	N	Session initiator waiting for session acceptor to send back **Logon** ACK.

FIX Logon Process State Transition Diagram

Session Initiator (e.g. buyside) **Action**	**Session Acceptor (e.g. sellside)** **Action**	**Session Initiator (e.g. buyside)** State	**Session Acceptor (e.g. sellside)** **State**
Start		• Disconnected-No Connection Today • Disconnected-Connection Today	Awaiting Connection
Connect		Initiate Connection (Possible) Detect Broken Network Connection	Awaiting Connection
	Accept Connection	Network Connection Established	Network Connection Established
Initiate **Logon**		Initiation **Logon** Sent	Network Connection Established
	Receive Initiation **Logon**	Initiation **Logon** Sent	Initiation **Logon** Received
	Send Initiation **Logon** Response	Initiation **Logon** Sent	Initiation **Logon** Response (possible) Initiate **Logout** Process (e.g. if *MsgSeqNum* too low) (Possible) Receive *MsgSeqNum* Too High
	(Possible) Send **ResendRequest**		Initiation **Logon** Response (Possible) Receive *MsgSeqNum* Too High
Receive Initiation **Logon** Response		(Possible) Active/Normal Session	Initiation **Logon** Response

		(Possible) Initiate **Logout** Process (e.g. if *MsgSeqNum* too low)	
(Possible) Send **ResendRequest**		(Possible) Active/Normal Session (Possible) Receive *MsgSeqNum* Too High	(Possible) Active/Normal Session (Possible) Handle **ResendRequest**
		Active/Normal Session	Active/Normal Session

FIX Logout Process State Transition Diagram

Logout Initiator: Action	**Logout Acceptor Action**	Logout Initiator State	**Logout Acceptor State**
Start		• Active/Normal Session • No messages received in Interval • Awaiting/Processing Response to **TestRequest**	• Active/Normal Session • No messages received in Interval • Initiation **Logon** Sent • Awaiting/Processing Response to **TestRequest** • Awaiting validation of logon • Receive *MsgSeqNum* Too High • Awaiting/Processing Response to **ResendRequest** • Initiate **Logout** Process • Waiting for **Logon** ack
Send **Logout** message		Logout Pending	
	Receive **Logout** message	Logout Pending	Logout Pending (Possible) Receive *MsgSeqNum* Too High
	Send **Logout** response	Logout Pending	Awaiting Disconnect
	(Possible) Send **ResendRequest**	Logout Pending	(Possible) Awaiting / Processing Response to **ResendRequest**

(Possible) receive **ResendRequest**		(Possible) Awaiting / Processing Response to **ResendRequest**	(Possible) Awaiting Response to **ResendRequest**
Receive **Logout** Response		Disconnected-Connection Today	Awaiting Disconnect
Disconnect		Disconnected-Connection Today	Disconnected-Connection Today

Test cases

These test cases are from the perspective of the FIX system being tested. The FIX system receives the “Condition / Stimulus” and is expected to take the appropriate action as defined by “Expected Behavior”.

Buyside-oriented (session initiator) Logon and session initiation test case

Ref ID	Pre-condition	Test case	Mandatory/ Optional	Condition/Stimulus	Expected Behavior
1B		Connect and Send **Logon** message	Mandatory	a. Establish Network connection	Successfully open TCP socket with counterparty
				b. Send **Logon** message	Send **Logon** message
				c. Valid **Logon** message as response is received	If *MsgSeqNum* is too high then send **Resend Request**
				d. Invalid **Logon** message is received	1. Generate an "error" condition in test output. 2. (Optional) Send **Reject** message with *RefMsgSeqNum* referencing **Logon** message’s *MsgSeqNum* with *Text* referencing error condition 3. Send **Logout** message with *Text* referencing error condition 4. Disconnect
				e. Receive any message other than a **Logon** message.	1. Log an error “first message not a logon” 2. (Optional) Send **Reject** message with *RefMsgSeqNum* referencing message’s *MsgSeqNum* with *Text* referencing error condition 3. Send **Logout** message with *Text* referencing error condition 4. Disconnect

Sellside-oriented (session acceptor) Logon and session initiation test case

Ref ID	Pre-condition	Test case	Mandatory/ Optional	Condition/Stimulus	Expected Behavior
1S		Receive **Logon** message	Mandatory	a. Valid **Logon** message	1. Respond with **Logon** response message 2. If *MsgSeqNum* is too high then send **Resend Request**
				b. **Logon** message received with duplicate identity (e.g. same IP, port, SenderCompID, TargetCompID, etc. as existing connection)	1. Generate an "error" condition in test output. 2. Disconnect without sending a message (note sending a Reject or Logout would consume a MsgSeqNum)
				c. **Logon** message received with unauthenticated/non-configured identity (e.g. invalid SenderCompID, invalid TargetCompID, invalid source IP address, etc. vs. system configuration)	1. Generate an "error" condition in test output. 2. Disconnect without sending a message (note sending a Reject or Logout would consume a MsgSeqNum)
				d. Invalid **Logon** message	1. Generate an "error" condition in test output. 2. (Optional) Send **Reject** message with *RefMsgSeqNum* referencing **Logon** message's *MsgSeqNum* with *Text* referencing error condition 3. Send **Logout** message with *Text* referencing error condition 4. Disconnect

		Receive any message other than a **Logon** message	Mandatory	First message received is not a Logon message.	1. Log an error "first message not a logon" 2. Disconnect

Test cases applicable to all FIX systems

Ref ID	Pre-condition	Test case	Mandatory/ Optional	Condition/Stimulus	Expected Behavior
2		Receive **Message Standard Header**	Mandatory	a. *MsgSeqNum* received as expected	Accept *MsgSeqNum* for the message
				b. *MsgSeqNum* higher than expected	Respond with **Resend Request** message
				c. *MsgSeqNum* lower than expected without *PossDupFlag* set to Y Exception: *SeqReset-Reset*	1. Whenever possible it is recommended that FIX engine attempt to send a Logout message with a text message of “MsgSeqNum too low, expecting X but received Y” 2. (optional) Wait for **Logout** message response (note likely will have inaccurate *MsgSeqNum*) or wait 2 seconds whichever comes first 3. Disconnect 4. Generate an "error" condition in test output.
				d. Garbled message received	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages. 2. Generate a "warning" condition in test output.
				e. *PossDupFlag* set to Y; *OrigSendingTime* specified is less than or equal to *SendingTime* and *MsgSeqNum* lower than expected Note: *OrigSendingTime* should be earlier than *SendingTime* unless the message is being resent within the same second during which it was sent.	1. Check to see if *MsgSeqNum* has already been received. 2. If already received then ignore the message, otherwise accept and process the message.

				f. *PossDupFlag* set to Y; *OrigSendingTime* specified is greater than *SendingTime* and *MsgSeqNum* as expected Note: *OrigSendingTime* should be earlier than *SendingTime* unless the message is being resent within the same second during which it was sent.	*1.* Send **Reject (session-level)** message referencing inaccurate *SendingTime* (>= FIX 4.2: SessionRejectReason = "SendingTime acccuracy problem") *2.* Increment inbound *MsgSeqNum* 3. Optional • Send **Logout** message referencing inaccurate *SendingTime* value • (optional) Wait for **Logout** message response (note likely will have inaccurate *SendingTime*) or wait 2 seconds whichever comes first • Disconnect Generate an "error" condition in test output.
				g. *PossDupFlag* set to Y and *OrigSendingTime* not specified Note: Always set *OrigSendingTime* to the time when the message was originally sent-not the present *SendingTime* and set *PossDupFlag* = “Y” when responding to a **Resend Request**	1. Send **Reject (session-level)** message referencing missing *OrigSendingTime* (>= FIX 4.2: SessionRejectReason = "Required tag missing") 2. Increment inbound *MsgSeqNum*
				h. *BeginString* value received as expected and specified in testing profile and matches *BeginString* on outbound messages.	Accept *BeginString* for the message

i. *BeginString* value (e.g. "FIX.4.2") received did not match value expected and specified in testing profile or does not match *BeginString* on outbound messages.	1. Send **Logout** message referencing incorrect *BeginString* value 2. (optional) Wait for **Logout** message response (note likely will have incorrect BeginString) or wait 2 seconds whichever comes first 3. Disconnect 4. Generate an "error" condition in test output.
j. *SenderCompID* and *TargetCompID* values received as expected and specified in testing profile.	Accept *SenderCompID* and *TargetCompID* for the message
k. *SenderCompID* and *TargetCompID* values received did not match values expected and specified in testing profile.	1. Send **Reject (session-level)** message referencing invalid *SenderCompID* or *TargetCompID* (>= FIX 4.2: SessionRejectReason = "CompID problem") 2. Increment inbound *MsgSeqNum* 3. Send **Logout** message referencing incorrect *SenderCompID* or *TargetCompID* value 4. (optional) Wait for **Logout** message response (note likely will have incorrect *SenderCompID* or *TargetCompID*) or wait 2 seconds whichever comes first 5. Disconnect 6. Generate an "error" condition in test output.
l. *BodyLength* value received is correct.	Accept *BodyLength* for the message

				m. *BodyLength* value received is not correct.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
				n. *SendingTime* value received is specified in UTC (Universal Time Coordinated also known as GMT) and is within a reasonable time (i.e. 2 minutes) of atomic clock-based time.	Accept *SendingTime* for the message
				o. *SendingTime* value received is either not specified in UTC (Universal Time Coordinated also known as GMT) or is not within a reasonable time (i.e. 2 minutes) of atomic clock-based time. Rationale: Verify system clocks on both sides are in sync and that SendingTime must be current time	*1.* Send **Reject (session-level)** message referencing inaccurate *SendingTime* (>= FIX 4.2: SessionRejectReason = "SendingTime acccuracy problem") *2.* Increment inbound *MsgSeqNum* 3. Send **Logout** message referencing inaccurate *SendingTime* value 4. (optional) Wait for **Logout** message response (note likely will have inaccurate *SendingTime*) or wait 2 seconds whichever comes first 5. Disconnect 6. Generate an "error" condition in test output.
				p. *MsgType* value received is valid (defined in spec or classified as user-defined).	Accept *MsgType* for the message

				q. *MsgType* value received is not valid (defined in spec or classified as user-defined).	*1.* Send **Reject (session-level)** message referencing invalid *MsgType* (>= FIX 4.2: SessionRejectReason = "Invalid MsgType") 2. Increment inbound *MsgSeqNum* 3. Generate a "warning" condition in test output.
				r. *MsgType* value received is valid (defined in spec or classified as user-defined) but not supported or registered in testing profile.	*1)* If < FIX 4.2 *a)* Send **Reject (session-level)** message referencing valid but unsupported *MsgType* *2)* If >= FIX 4.2 *a)* Send **Business Message Reject** message referencing valid but unsupported *MsgType* (>= FIX 4.2: BusinessRejectReason = "Unsupported Message Type") 3) Increment inbound *MsgSeqNum* 4) Generate a "warning" condition in test output.
				s. *BeginString*, *BodyLength*, and *MsgType* are first three fields of message.	Accept the message
				t. *BeginString*, *BodyLength*, and *MsgType* are not the first three fields of message.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
3		Receive **Message Standard Trailer**	Mandatory	a. Valid *CheckSum*	Accept Message

				b. Invalid *CheckSum*	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
				c. Garbled message	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
				d. *CheckSum* is last field of message, value has length of 3, and is delimited by <SOH>.	Accept Message
				e. *CheckSum* is not the last field of message, value does not have length of 3, or is not delimited by <SOH>.	1. Consider garbled and ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
4		Send **Heartbeat** message	Mandatory	a. No data sent during preset heartbeat interval (*HeartBeatInt* field)	Send **Heartbeat** message
				b. A **Test Request** message is received	Send **Heartbeat** message with **Test Request** message's *TestReqID*
5		Receive **Heartbeat** message	Mandatory	Valid **Heartbeat** message	Accept **Heartbeat** message
6		Send **Test Request**	Mandatory	No data received during preset heartbeat interval (*HeartBeatInt* field) + "some reasonable period of time" (use 20% of *HeartBeatInt* field)	1. Send **Test Request** message 2. Track and verify that a **Heartbeat** with the same *TestReqID* is received (may not be the next message received)
7		Receive **Reject** message	Mandatory	Valid **Rejec**t message	1. Increment inbound *MsgSeqNum* 2. Continue accepting messages

8		Receive **Resend Request** message	Mandatory	Valid Resend Request	Respond with application level messages and **SequenceReset**-*Gap Fill* for admin messages in requested range according to "Message Recovery" rules.
9		Synchronize sequence numbers	Optional	Application failure	Send **Sequence Reset - Reset** message or manually reset to 1 out-of-band.
10		Receive **Sequence Reset (Gap Fill)**	Mandatory	a. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum > than expect sequence number	Issue **Resend Request** to fill gap between last expected MsgSeqNum & received MsgSeqNum.
				b. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum = to expected sequence number	Set next expected sequence number = NewSeqNo
				c. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum < than expected sequence number and PossDupFlag = “Y”	Ignore message

				d. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo > MsgSeqNum and MsgSeqNum < than expected sequence number and without PossDupFlag = "Y"	1) If possible send a Logout message with text of "MsgSeqNum too low, expecting X received Y," prior to disconnecting FIX session. 2) (optional) Wait for **Logout** message response (note likely will have inaccurate *MsgSeqNum*) or wait 2 seconds whichever comes first 3) Disconnect 4) Generate an "error" condition in test output
				e. Receive **Sequence Reset (Gap Fill)** message with NewSeqNo <= MsgSeqNum and MsgSeqNum = to expected sequence number	Send **Reject (session-level)** message with message "attempt to lower sequnce number, invalid value NewSeqNum=<x>"
11		Receive **Sequence Reset (Reset)**	Mandatory	a. Receive **Sequence Reset (reset)** message with *NewSeqNo* > than expected sequence number	1) Accept the **Sequence Reset (Reset)** message without regards to its *MsgSeqNum* 2) Set expected sequence number equal to NewSeqNo
				b. Receive **Sequence Reset (reset)** message with *NewSeqNo* = to expected sequence number	1) Accept the **Sequence Reset (Reset)** message without regards to its *MsgSeqNum* 2) Generate a "warning" condition in test output.

				c. Receive **Sequence Reset (reset)** message with *NewSeqNo* < than expected sequence number	*1)* Accept the **Sequence Reset (Reset)** message without regards to its *MsgSeqNum* 2) Send **Reject (session-level)** message referencing invalid *MsgType* (>= FIX 4.2: SessionRejectReason = "Value is incorrect (out of range) for this tag") 3) Do NOT Increment inbound *MsgSeqNum* 4) Generate an "error" condition in test output 5) Do NOT lower expected sequence number.
12		Initiate logout process	Mandatory	Initiate Logout	1) Send **Logout** message 2) Wait for counterparty to respond with **Logout** message up to 10 seconds (note may not be received if communications problem exists). If not received, generate a “warning” condition in test output. 3) Disconnect
13		Receive **Logout** message	Mandatory	a. Receive valid **Logout** message in response to a solicited logout process	Disconnect without sending a message
				b. Receive valid **Logout** message unsolicitied	1. Send **Logout** response message 2. Wait for counterparty to disconnect up to 10 seconds. If max exceeded, disconnect and generate an “error” condition in test output.

14		Receive application or administrative message	Mandatory	a. Receive field identifier (tag number) not defined in specification. Exception: undefined tag used is specified in testing profile as user-defined.	*1.* Send **Reject (session-level)** message referencing invalid tag number (>= FIX 4.2: SessionRejectReason = "Invalid tag number") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
				b. Receive message with a required field identifier (tag number) missing.	*1.* Send **Reject (session-level)** message referencing required tag missing (>= FIX 4.2: SessionRejectReason = "Required tag missing") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
				c. Receive message with field identifier (tag number) which is defined in the specification but not defined for this message type. Exception: undefined tag used is specified in testing profile as user-defined for this message type.	*1.* Send **Reject (session-level)** message referencing tag not defined for this message type (>= FIX 4.2: SessionRejectReason = "Tag not defined for this message type") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
				d. Receive message with field identifier (tag number) specified but no value (e.g. "55=<SOH>" vs. "55=IBM<SOH>").	*1.* Send **Reject (session-level)** message referencing tag specified without a value (>= FIX 4.2: SessionRejectReason = "Tag specified without a value") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

				e. Receive message with incorrect value (out of range or not part of valid list of enumerated values) for a particular field identifier (tag number). Exception: undefined enumeration values used are specified in testing profile as user-defined.	1. Send **Reject (session-level)** message referencing value is incorrect (out of range or not part of valid list of enumerated values) for this tag (>= FIX 4.2: SessionRejectReason = "Value is incorrect (out of range) for this tag") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
				f. Receive message with a value in an incorrect data format (syntax) for a particular field identifier (tag number).	1. Send **Reject (session-level)** message referencing value is in an incorrect data format for this tag (>= FIX 4.2: SessionRejectReason = "Incorrect data format for value") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
				g. Receive a message in which the following is not true: Standard Header fields appear before Body fields which appear before Standard Trailer fields.	1. Send **Reject (session-level)** message referencing incorrect message structure header+body+trailer (>= FIX 4.3: SessionRejectReason = “Tag specified out of required order”) 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

h. Receive a message in which a field identifier (tag number) which is not part of a repeating group is specified more than once	1. Send **Reject (session-level)** message referencing duplicate field identifier (tag number) (>= FIX 4.3: SessionRejectReason = “Tag appears more than once”) 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
i. Receive a message with repeating groups in which the "count" field value for a repeating group is incorrect.	1. Send **Reject (session-level)** message referencing the incorrect "count" field identifier (tag number) (>= FIX 4.3: SessionRejectReason = “Incorrect NumInGroup count for repeating group”) 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
j. Receive a message with repeating groups in which the order of repeating group fields does not match the specification.	1. Send **Reject (session-level)** message referencing the repeating group with incorrect field ordering (>= FIX 4.3: SessionRejectReason = “Repeating group fields out of order”) 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

				k. Receive a message with a field of a data type other than "data" which contains one or more embedded <SOH> values.	1. Send **Reject (session-level)** message referencing field identifier (tag number) with embedded <SOH> (>= FIX 4.3: SessionRejectReason = “Non “data” value includes field delimiter (SOH character)”) 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output. ?? Discard as valid response/outcome too or Consider garbled and ignore message
				l. Receive a message when application-level processing or system is not available (Optional)	*1)* If < FIX 4.2 *a)* Send **Reject (session-level)** message referencing application message processing is not available *2)* If >= FIX 4.2 *a)* Send **Business Message Reject** message referencing application message processing is not available (>= FIX 4.2: BusinessRejectReason = "Application not available") 3) Increment inbound *MsgSeqNum* 4) Generate a "warning" condition in test output.

				m. Receive a message in which a conditionally required field is missing.	*1)* If < FIX 4.2 *a)* Send **Reject (session-level)** message referencing field identifier (tag number) of the missing conditionally required field(s) *2)* If >= FIX 4.2 *a)* Send **Business Message Reject** message referencing field identifier (tag number) of the missing conditionally required field(s) (>= FIX 4.2: BusinessRejectReason = "Conditionally Required Field Missing") 3) Increment inbound *MsgSeqNum* 4) Generate an "error" condition in test output.
				N. Receive a message in which a field identifier (tag number) appears in both cleartext and encrypted section but has different values.	*1.* Send **Reject (session-level)** message referencing field identifier (tag number) missing from unencrypted section (>= FIX 4.2: SessionRejectReason = " Decryption problem") *2.* Increment inbound *MsgSeqNum* Generate an "error" condition in test output.
15		Send application or administrative messages to test normal and abnormal behavior/response		Send more than one message of the same type with header and body fields ordered differently to verify acceptance. (Exclude those which have restrictions regarding order)	Messages accepted and subsequent messages' *MsgSeqNum* are accepted.

16		Queue outgoing messages	Mandatory	a. Message to send/queue while disconnected	Queue outgoing messages. Note there are two valid approaches: 1) Queue without regards to *MsgSeqNum* a) Store data for messages 2) Queue each message with the next *MsgSeqNum* value a) Store data for messages in such a manner as to use and "consume" the next *MsgSeqNum* Note: SendingTime (Tag#52): must contain the time the message is sent not the time the message was queued.

				b. Re-connect with queued messages	1. Complete logon process (connect, and **Logon** message exchange) 2. Complete *MsgSeqNum* recovery process if applicable. 3. Recommended short delay or **TestRequest/Heartbeat** to verify *MsgSeqNum* recovery completed. 4. Note there are two valid queuing approaches: a) Queue without regards to *MsgSeqNum* i) Send queued messages with new *MsgSeqNum* values (greater than **Logon** message's *MsgSeqNum*) b) Queue each message with the next *MsgSeqNum* value i) (note **Logon** message's *MsgSeqNum* will be greater than the queued messages' *MsgSeqNum)* ii) Counterparty will issue **ResendRequest** requesting the range of missed messages iii) Resend each queued message with *PossDupFlag* set to Y Note: SendingTime (Tag#52): must contain the time the message is sent not the time the message was queued.

17		Support encryption	Optional	a. Receive **Logon** message with valid, supported *EncryptMethod*	1. Accept the message 2. Perform the appropriate decryption and encryption method readiness 3. Respond with **Logon** message with the same *EncryptMethod*
				b. Receive **Logon** message with invalid or unsupported *EncryptMethod*	*1.* Send **Reject (session-level)** message referencing invalid or unsupported *EncryptMethod* value (>= FIX 4.2: SessionRejectReason = "Decryption problem") 2. Increment inbound *MsgSeqNum* 3. Send **Logout** message referencing invalid or unsupported *EncryptMethod* value 4. (optional) Wait for **Logout** message response (note could have decrypt problems) or wait 2 seconds whichever comes first 5. Disconnect 6. Generate an "error" condition in test output.
				c. Receive message with valid *SignatureLength* and *Signature* values.	Accept the message
				d. Receive message with invalid *SignatureLength* value.	*1.* Send **Reject (session-level)** message referencing invalid *SignatureLength* value (>= FIX 4.2: SessionRejectReason = "Signature problem") *2.* Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

			e. Receive message with invalid *Signature* value.	*1.* Send **Reject (session-level)** message referencing invalid *Signature* value (>= FIX 4.2: SessionRejectReason = "Signature problem") *2.* Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output. Or consider decryption error or message out of order, ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages
			f. Receive message with a valid *SecureDataLen* value and a *SecureData* value that can be decrypted into valid, parse-able cleartext.	Accept the message
			g. Receive message with invalid *SecureDataLen* value.	*1.* Consider decryption error or message out of order, ignore message (do not increment inbound *MsgSeqNum*) and continue accepting messages 2. Generate a "warning" condition in test output.
			h. Receive message with a *SecureData* value that cannot be decrypted into valid, parse-able cleartext.	*1.* Send **Reject (session-level)** message referencing invalid *SecureData* value (>= FIX 4.2: SessionRejectReason = " Decryption problem") *2.* Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

				i. Receive message with one or more fields not present in the unencrypted portion of the message that "must be unencrypted" according to the spec.	3. Send **Reject (session-level)** message referencing field identifier (tag number) missing from unencrypted section (>= FIX 4.2: SessionRejectReason = " Decryption problem") 4. Increment inbound *MsgSeqNum* 5. Generate an "error" condition in test output.
				j. Receive message with incorrect handling of "left over" characters (e.g. when length of cleartext prior to encryption is not a multiple of 8) according to the specified *EncryptMethod*.	1. Send **Reject (session-level)** message referencing incorrect handling of "left over" characters during encryption (>= FIX 4.2: SessionRejectReason = " Decryption problem") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.
18		Support third party addressing	Optional	a. Receive messages with *OnBehalfOfCompID* and *DeliverToCompID* values expected as specified in testing profile and with correct usage.	Accept messages
				b. Receive messages with *OnBehalfOfCompID* or *DeliverToCompID* values not specified in testing profile or incorrect usage.	1. Send **Reject (session-level)** message referencing invalid *OnBehalfOfCompID* or *DeliverToCompID* (>= FIX 4.2: SessionRejectReason = "CompID problem") 2. Increment inbound *MsgSeqNum* 3. Generate an "error" condition in test output.

19		Test PossResend handling	Mandatory	a. Receive message with PossResend = “Y” and application-level check of Message specific ID indicates that it has already been seen on this session	1. Ignore the message. 2. Generate a "warning" condition in test output
				b. Receive message with PossResend = “Y” and application-level check of Message specific ID indicates that it has NOT yet been seen on this session	1. Accept and process the message normally.
20		Simultaneous Resend request test	Mandatory	Receive a **Resend Request** message while having sent and awaiting complete set of responses to a **Resend Request** message.	1. Perform resend of requested messages. 2. Send **Resend Request** to request missed messages