FIX Performance Session Layer

Release Candidate 1

Technical Proposal

September 8, 2014

v0.15

Proposal Status: Public Review

For Global Technical Committee Governance Internal Use Only

Submission Date	September 18, 2014	Control Number	
Submission Status	Submitted	Ratified Date	
Primary Contact Person	Don Mendelson	Release Identifier	

DISCLAIMER

Table of Contents

Table of Figures

Document History

Revision	Date	Author	Revision Comments
0.1	2013-03-25	Jim N	
0.2	2013-04-13	Jim N (editor), Anders Furuhed (submitter), David Rosenberg (submitter)	Significant refactoring based upon prototyping and development work performed by Pantor Engineering to achieve a single session layer protocol that can address reliable, unreliable message flow, for both transactional and information distribution (market data) applications. Renamed FIXNS to FIXP to recognize this next level of refactoring and design. The original concept of message type identification has been optimized. Sequence numbers are now fully implicit, with explicit framing using a Sequence message.
0.3	2013-05-20	Jim N	Corrected some typographical errors.
0.4	2013-08-12	Jim N	Removed Challenge response, Removed data type information instead specified ranges. Revised datatype section during review on 2013-08-12, included reference to RFC 4122.
0.5	2013-08-20	Jim N	Added sequence diagrams
0.6	2013-12-03	Julio M	Updated to comply with XMIT alpha 8.
0.7	2014-04-07	Julio M	Updated to comply with XMIT alpha 13. Still needs diagram updates.
0.8	2014-04-24	Julio M	Updated to comply with XMIT alpha 14. Still needs diagram updates.
0.9	2014-05-28	Jim N, Aditya Kapur, Julio M.	RC1 Candidate, renamed “Sequenced” flow type to “Recoverable”, diagram updates, inclusion of use case scenarios.
0.10	2014-06-11	Don Mendelson	Section 2.1 (Introduction): explain encoding independence for both session and application layers Chapter 3.1 naming convention. Chapter 3.3: rename to section 3.5, explain differences between initiator and acceptor session (is there an asymmetry?)

Revision	Date	Author	Revision Comments
			Chapter 3.4.1: Note to Open Framing Header in chapter 1.3 3.6.1 Explained flow types 3.8 Reasons for terminating a session 3.12 Applied only used for non-standard application responses
0.11	2014-07-18	Don Mendelson	Removed mention of reference implementation. Updated references and removed unpublished documents. Added explanation of reject codes.
0.12	2014-08-13	Don Mendelson	Recommendation of authentication protocol deferred. Added statement that only application messages are sequenced. Explained when a Sequence must be sent. Explained session lifetime. Added negotiation reject code for duplicate ID. Stated than RetransmitRequest should only be sent for recoverable flows and NotApplied is only for idempotent flows. Duplicate requests on an idempotent flow are silently dropped. Appendix added with rules of engagement checklist.
0.13	2014-08-13	Don Mendelson	Clarified distinction between termination and finalization of flow. Added session state machine diagram. Added recommendation about timeout leniency (was a comment). Added discussion of heartbeats and silence is termination. Added recommendation about Nagle.
0.14	2014-09-04	Don Mendelson	Updated from XMIT version alpha15.
0.15	2014-09-08	Don Mendelson	Corrected description of NextSeqNo on establishment and added other explanations discussed on the call today.

1 Introduction

FIX Performance Session Layer (FIXP) is a "lightweight point-to-point protocol" introduced to provide an open industry standard for high performance computing requirements currently encountered by the FIX Community. FIXP is a derived work. The origin and basis for FIXP are the FIX session layer protocols and the designed and implemented by NASDAQOMX, SoupTCP, SoupBinTCP, and UFO (UDP for Orders). Every attempt was made to keep FIXP as close to the functionality and behavior of SoupBinTCP and UFO as possible. Extensions and refactoring were performed as incremental improvements. Every attempt was made to limit the FIXP to establishing and maintaining a communication session between two end points in a reliable manner, regardless of the reliability of the underlying transport.

FIXP features

- Binary protocol
- Very simple lightweight point-to-point session layer for reliable communication.
- Communication protocol independent
- Encoding independent
- Restricted to initiating, maintaining, and reestablishing a session.

The idea to provide an open standard high performance session layer with SoupBinTCP as its source came from two simultaneous sources.

1. The BVMF (Brazil) began investigating SoupBinTCP as a lightweight and simple alternative for market data delivery due to issues with Multicast IP infrastructure at member firms. The idea to align packet types to existing FIX message types was created during a meeting prior to the start of the High Performance Working Group.
2. Pantor Engineering prototyped a solution for high performance computing that used FAST datatypes (without field operators) carried over a SoupBinTCP session. Anders Furuhed presented the concept at the FIX Nordic event.

1.1 Authors

Name	Affiliation	Contact	Role
Anders Furuhed	Pantor Engineering	anders@pantor.com	Protocol Designer
David Rosenborg	Pantor Engineering	david.rosenborg@pantor.com	Protocol Designer
Rolf Andersson	Pantor Engineering	rolf@pantor.com	Contributor, GTC Governance Board member
Jim Northey	LaSalle Technology	jimn@lasalletech.com	Editor, Working group convener
Julio Monteiro	BVMF Bovespa	jmonteiro@bvmf.com.br	Editor, Working Group convener
Aditya Kapur	CME Group, Inc	Aditya.kapur@cmegroup.com	Working Group Participant – provided document editing and input on exchange adoption

Don Mendelson	CME Group, Inc.	Don.Mendelson@cmegroup.com	Working Group Participant and regular contributor
Li Zhu	Shanghai Stock Exchange	lzhu@sse.com.cn	Working Group Participant and regular contributor

1.2 Relevant and Related Standards

1.2.1 Sources

These standards were sources for concepts but are non-normative for FIXP.

Reference	Version	Relevance	Normative
UFO (UDP for Orders) NASDAQ OMX	Version 1.0, July 3, 2008	Basis for high performance session layer.	No
SoupBinTCP NASDAQ OMX	3.00	Basis for high performance session layer.	No
FIXT Session Layer Specification	1.1	The previous FIX session layer specification	No
XMIT	alpha15	High performance session protocol design by Pantor Engineering	No

1.2.2 Related FIX Standards

The FIX Simple Open Framing Header standard governs how messages are delimited and has a specific relationship mentioned in this specification. FIXP interoperates with the other FIX standards at application and presentation layers, but it is not dependent on them. Therefore, they are considered non-normative for FIXP.

Related Standard	Version	Reference location	Relationship	Normative
Simple Open Framing Header	RC1		Optional usage at presentation layer	Yes
FIX message specifications	5.0 SP 2		Presentation and application layers	No
FIX Simple Binary Encoding	RC2		Optional usage at presentation layer	No
Encoding FIX Using ASN.1	Draft Standard		Optional usage at presentation layer	No
Encoding FIX Using GPB	RC2		Optional usage at presentation layer	No

1.2.3 Dependencies on Other Specifications

FIXP is dependent on several industry standards. Implementations of FIXP must conform to these standards to interoperate. Therefore, they are normative for FIXP. Other protocols may be used by agreement between counterparties.

Related Standard	Version	Reference location	Relationship	Normative
RFC 793 Transmission Control Program (TCP)	N/A	http://tools.ietf.org/html/rfc793 and related standards	Uses transport	Yes
RFC 768 User Datagram Protocol (UDP)	N/A	http://tools.ietf.org/html/rfc768	Uses transport	Yes
RFC4122 A Universally Unique Identifier (UUID) URN Namespace	N/A	http://tools.ietf.org/html/rfc4122	Uses	Yes
UTF-8, a transformation format of ISO 10646	N/A	http://tools.ietf.org/html/rfc3629	Uses encoding	Yes
Various authentication protocols	N/A		Optional usage at session layer	No

1.3 Intellectual Property Disclosure

Related Intellection Property	Type of IP (copyright, patent)	IP Owner	Relationship to proposed standard
Blink http://blinkprotocol.org/spec/BlinkSpec-beta3.pdf	Copyright	Pantor Engineering	
XMIT	Copyright	Pantor Engineering	Basis for design of protocol
Soup, SoupBinTCP, UFO (UDP for Orders), and MoldUDP	Copyright	NASDAQOMX	FIXP is intended to provide functionality equivalent to these protocols.

1.4 Definitions

Term	Definition
Client	Initiator of session.
Server	Acceptor of session
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
IPC	Inter-Process Communication
IP MC	IP Multicast
IDEMP	Idempotence means that an operation that is applied multiple times does not change the outcome, the result, after the first time

2 Requirements

2.1 Business Requirements

Create an enhanced session protocol that can provide reliable, highly efficient, exchange of messages to support high performance financial messaging, over a variety of transports.

Protocol shall be fit for purpose for current high message rates, low latency environments in financial markets, but should be to every extent possible applicable to other business domains. There is no reason to limit or couple the session layer to the financial markets / trading business domain without extraordinary reason.

Support common message flow types: Recoverable, Unsequenced and Idempotent (operations guaranteed to be applied only once).

Protocol shall support asymmetric models, such as market participant to market, in addition to peer-to-peer (symmetric). Allow the communication of messages to multiple receivers (broadcast).

The session protocol does not require or recommend a specific authentication protocol. Counterparties are free to agree on user authentication techniques that fit their needs.

2.2 Technical Requirements

2.2.1 Protocol Layering

This standard endeavors to maintain a clear separation of protocol layers, as expressed by the Open Systems Interconnection model (OSI). The responsibilities of a session layer are establishment, termination and restart procedures and rules for the exchange of application messages.

The protocol shall be independent of message encoding (presentation layer), to provide the maximum utility. Encoding independence applies both to session layer messages specified in this document as well as to application messages. It is simpler if session protocol messages are encoded the same way as application messages, but that is not a requirement of this session protocol.

Users are free to specify message encodings by agreement with counterparties. FIX provides Simple Binary Encoding as well as mappings of FIX to other high performance encodings such as ASN.1, and

Google Protocol Buffers. See the list of related standards above. Other recognized encodings may follow in the future.

Of necessity, the session protocol makes some adaptations for transport layer protocols used by the session layer since the capabilities of common transports are quite different. In particular, TCP is connection- and stream-oriented and implements its own reliable delivery mechanisms. Meanwhile, UDP is datagram-oriented and does not guarantee delivery in order. Unfortunately, these characteristics bleed across protocol layers.

2.2.2 Low overhead

Minimum overhead is added to the messages exchanged between peers, using only the strictly necessary control messages.

By agreement between counterparties, a message framing protocol may be used to delimit messages. This relieves the session layer of application message decoding to determine message boundaries. FIX offers the Simple Open Framing Header standard for framing messages encoded with binary wire formats. See standards references above.

3 FIX Performance Session Layer

3.1 Usage and naming conventions

All symbolic names for messages and fields in this protocol should follow the same naming convention as other FIX specifications: alphanumeric characters plus underscores without spaces.

3.2 Datatypes

Datatypes are abstract. Actual encoding of FIXP is left to the implementation.

Logical Type	**Range**	**Native Type**	**Comments**
u8	0..2^8-1		
u16	0..2^16-1		
u32	0..2^32-1		
u64	0..2^64-1		
UUID	RFC 4122 compliant UUID		The requirement is to provide a mechanism that can be self-generated and guaranteed free of collision. Implementers are encouraged to adopt version 4.
String	text		UTF-8, length may need to be specified as part of the encoding.
Bool	true / false	u8	
nanotime	Time in nanoseconds	u64	
DeltaMillisecs	Number of milliseconds	u32	
Object			Unspecified data content Requires some way to determine length

3.3 FIXP Session Messages

Each message starts with a message type as the first field.

3.3.1 Message Type Identification

The session layer messages have high type id values as these messages are relatively uncommon. The potentially more compact lower values are therefore left for other use by applications.
There is one exception: the Sequence message, which is common and therefore is assigned a low type id value. This session layer reserves the type id 0x01 for Sequence and the range #0x10000 to 0x10210 for other messages.
All other values between 0x02 and 0x0FFFF are available for use in identifying application messages. A FIXP Processor shall make the application messages available to the application layer without decoding or processing.

Purpose	Start Hexadecimal	End Hexadecimal
Sequence Message	0x01	
Application Messages	0x02-	0x0FFFF
FIXP Session Messages	0x10000	0x10210

3.4 Message Sequencing

3.4.1 Sequence numbering

Sequence numbering supports ordered delivery and recovery of messages. In FIXP, only application messages are sequenced, not session protocol messages. A Sequence message is used to start a sequenced flow of application messages. Any applications message passed after a Sequence message is implicitly numbered, where the first message after Sequence has the sequence number NextSeqNo.

Sequence				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x01	
NextSeqNo	u64	Y		The sequence number of the next message after the Sequence message.

3.4.2 Message framing

FIXP does not require application messages to have a session layer header. Application messages may have their own presentation layer header, depending on encoding. However, application messages may immediately follow Sequence without any intervening session layer prologue.

Optionally, application messages may be delimited by use of the Simple Open Framing Header. This is most useful if session message encoding is different than application message encoding or if a session carries application messages in multiple encodings. The framing header identifies the encoding of the message that follows and gives its overall length. If it is used, then FIXP need not parse application messages to determine length and keep track of message counts in a flow.

3.4.3 Application message sequencing considerations

An application layer defined on top is obviously free to put any required application level sequencing inside messages.

3.4.4 Datagram oriented protocol considerations

Using a datagram oriented transport like UDP, each datagram carrying a sequenced flow, the Sequence message is key to detecting packet loss and packet reordering and must precede any application messages in the packet.

FIXP provides no mechanism for fragmenting messages across datagrams. In other words, each application message must fit within a single datagram on UDP.

3.4.5 Multiplexed session considerations

If sessions are multiplexed over a transport, they are framed independently. When multiplexing, the Context message expands Sequence to also specify the session being sequenced.
If flows are multiplexed over a transport, the transport does not imply the session. Context is used to set the session for the remainder of the current datagram (in a datagram oriented transport) or until a new Context is passed. In a sequenced flow, Context can take the role of Sequence by including NextSeqNo (optimizes away the Sequence that would otherwise follow).

Context				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10050	
SessionId	UUID	Y		Session Identifier
NextSeqNo	u64	N		The sequence number of the next message after the Context message.

3.4.6 Sequence context switches

A change in session context ends the sequence of messages implicitly and the sender must pass a Sequence or Context message again before starting to send sequenced messages. A Sequence message must be sent if the session is not multiplexed and Context must be sent if it is multiplexed.
Changes of session context include:

- Interleaving of new, real-time messages and retransmitted messages.
- Switching from one multiplexed session to another when sharing a transport.

3.5 Session Properties

3.5.1 Session identification

Each session is identified by a unique Session ID encoded as a UUID version 4 (RFC 4122) assigned by the client. The benefit of using an UUID is that it is effortless to allocate in a distributed system. It is also simple and efficient to hash and therefore easy to look up at the endpoints. The downside is a larger size overhead. The identifier however does not appear in the stream except once at the start of each datagram, when using UDP, or when sessions are multiplexed, regardless of the underlying transport that is used. For a non-multiplexed TCP session, the identifier therefore appears only once during the lifetime of the TCP session. A UUID is intended to be unique, not only amongst currently active sessions, but for all time. Reusing a session ID is a protocol violation.

3.5.2 Session lifetime

A logical session is established between counterparties and lasts until information flows between them are complete. Commonly, such flows are concurrent with daily trading sessions, but no set time limit is imposed by this protocol. Rather, timings for session start and end are set by agreement between counterparties.

A logical session is identified by a session ID, as described above, until its information flows are finalized. After finalization, the old session ID is no longer valid, and messages are no longer recoverable. Counterparties may subsequently start a new session under a different ID.

A logical session is bound to a transport, but a session may outlive a transport connection. The binding to a transport may be terminated intentionally or may be triggered by an error condition. However, a client may reconnect and bind the existing session to the new transport. When re-establishing an existing session, the original session ID continues to be used, and recoverable messages that were lost by disconnection may be recovered .

3.5.3 *Summary of messages that control lifetime*

Logical sessions are created by using the Negotiation message. The session ID is sent in the Negotiation message and that ID is used for the lifetime of the session.

After negotiation is complete, the client sends an Establish message to reach the established state. Once established, exchange of application messages may proceed. The established state is concurrent with the lifetime of a connection-oriented transport such as TCP. A client can re-establish a previous session after reconnecting without any further negotiation. Thus, Establish binds the session to the new transport instance.

To signal a counterparty that a disconnection is about to occur, a Terminate message is sent. This unbinds the transport from the session, but it does not end a logical session.

A session that has a recoverable flow may be re-established by sending Establish with the same session ID, and an exchange of messages may continue until all business transactions are finished.

A logical session is ended by sending a FinishedSending message. Thereafter, no more application messages should be sent. The counterparty responds with FinishedReceiving when it has processed the last message, and then the transport is terminated for the final time for that session. Once a flow is finalized and the transport is unbound, a session ID is no longer valid and messages previously sent on that session are no longer recoverable.

3.6 *Session Initiation and Negotiation*

A negotiation dialog is provided to support a session negotiation protocol that is used for a client to declare what id it will be using, without having to go out of band. There is no concept of resetting a session. Instead of starting over a session, a new session is negotiated - a SessionId in UUID form is cheap.
The optional session negotiation is expected to occur at session initiation.

3.6.1 *Flow Type*

The negotiation protocol identifies the types of message flow that can occur, Recoverable, Unsequenced or Idempotent.

```
FlowType =  Recoverable | Unsequenced | Idempotent
```

From highest to lowest delivery guarantee, the flow types are:

- **Recoverable**: Guarantees exactly-once message delivery. If gaps are detected, then missed messages are recovered by retransmission.
- **Idempotent**: Guarantees at-most-once delivery. If gaps are detected, the sender is notified, but recovery is under control of the application, if it is done at all.

- **Unsequenced**: Makes no delivery guarantees (best-effort). This choice is appropriate if guarantees are unnecessary or if recovery is provided at the application layer or through a different communication channel.

By agreement between counterparties, only certain of these flow types may be supported for a particular service.

3.6.2 *Initiate Session Negotiation*

Negotiate message is sent from client to server.

Negotiate				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10000	
Timestamp	nanotime	Y		Time of request
SessionId	UUID	Y		Session Identifier
ClientFlow	FlowType	Y		Type of flow over the session
Credentials	Object	N		Optional credentials to authenticate session initiator. Format and protocol for authentication to be determined by agreement between counterparties.

3.6.3 *Accept Session Negotiation*

When a session is accepted by a server, it sends a NegotiationResponse in response to a Negotiate message.

NegotiationResponse				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10001	
RequestTimestamp	nanotime	Y		Matches Negotiate.Timestamp
SessionId	UUID	Y		Session Identifier
ServerFlow	FlowType	Y		Type of flow over the session

3.6.4 *Reject Session Negotiation*

When a session cannot be created, a server sends NegotiationReject to the client, giving the reason for the rejection. No further messages should be sent, and the transport should be terminated.

```
NegotiationRejectCode = Credentials | Unspecified | FlowTypeNotSupported | DuplicateId
```

Rejection reasons

- Credentials: failed authentication because identity is not recognized, keys are invalid, or the user is not authorized to use a particular service.
- FlowTypeNotSupported: server does not support requested client flow type.
- DuplicateId: session ID is non-unique.
- Unspecified: Any other reason that the server cannot create a session.

If negotiation is re-attempted after rejection, a new session ID should be generated.

NegotiationReject				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10002	
RequestTimestamp	nanotime	Y		Matches Negotiate.Timestamp
SessionId	UUID	Y		Session Identifier
Code	NegotiationRejectCode	Y		
Reason	string	N		Reject reason details

3.6.5 *Session Negotiation Sequence Diagram*

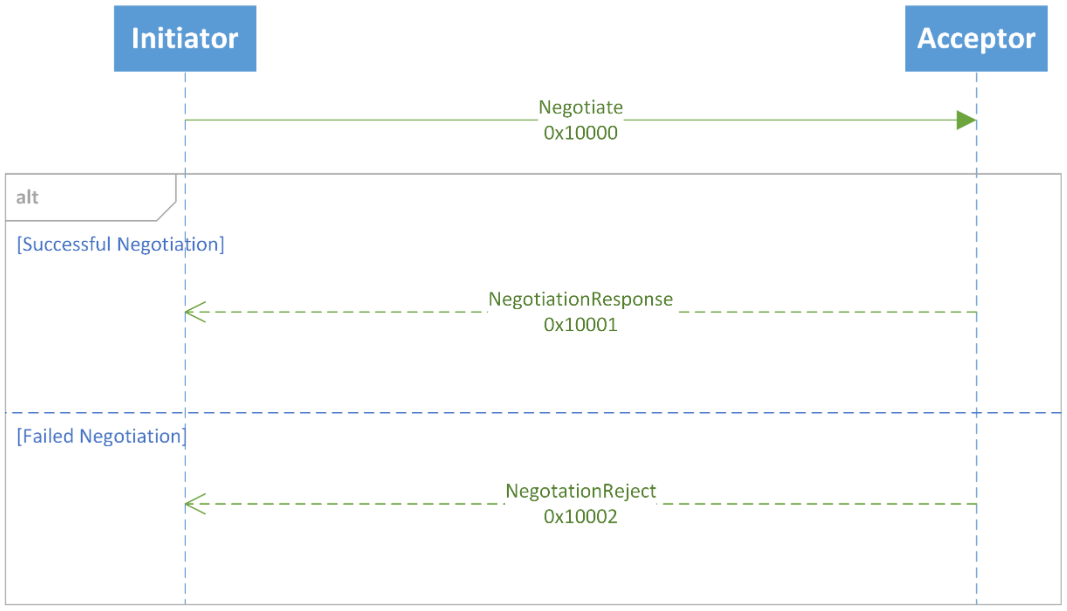

Figure 1 Session Negotiation Sequence Diagram

3.7 *Session Establishment*

Establish attempts to bind the specified logical session to the transport that the message is passed over. The response to Establish is either EstablishmentAck or EstablishmentReject.

3.7.1 *Establish*

The client sends Establish message to the server and awaits acknowledgement.

There is no specific timeout value for the wait defined in this protocol. Experience should be a guide to determine an abnormal wait after which a server is considered unresponsive. Then establishment may be retried or out-of-band inquiry may be made to determine application readiness.

Establish				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10010	
Timestamp	nanotime	Y		Time of request
SessionId	UUID	Y		Session Identifier
KeepaliveInterval	DeltaMillisecs	Y		The longest time in milliseconds the initiator will remain silent before sending a keep alive message
NextSeqNo	u64	N		For re-establishment of a recoverable server flow only, the next application sequence number to be produced by the client.
Credentials	object	N		Credentials for session authorization

Counterparties may agree on a valid range for KeepaliveInterval.
The server should evaluate NextSeqNo to determine whether it missed any messages after re-establishment of a recoverable flow. If so, it may immediately send a RetransmitRequest after sending EstablishAck.

3.7.2 *Establish Acknowledgment*

Used to indicate the acceptor acknowledges the session. If the communication flow from this endpoint is recoverable, it will fill the NextSeqNo field, allowing the initiator to start requesting the replay of messages that it has not received.

EstablishmentAck				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10011	
SessionId	UUID	Y		SessionId is included only for robustness, as matching RequestTimestamp is enough
RequestTimestamp	nanotime	Y		Matches Establish.Timestamp
KeepaliveInterval	DeltaMillisecs	Y		The longest time in milliseconds the acceptor will wait before sending a keep alive message
NextSeqNo	u64	N		For a recoverable server flow only, the next application sequence number to be produced by the server.

The client should evaluate NextSeqNo to determine whether it missed any messages after re-establishment of a recoverable flow. If so, it may immediately send a RetransmitRequest .

3.7.3 *Establish Reject*

EstablishmentRejectCode = Unnegotiated | AlreadyEstablished | SessionBlocked | KeepaliveInterval | Credentials | Unspecified

Rejection reasons:

- Unnegotiated: Establish request was not preceded by a Negotiation or session was finalized, requiring renegotiation.
- AlreadyEstablished: EstablishmentAck was already sent; Establish was redundant.
- SessionBlocked: user is not authorized
- KeepaliveInterval: value is out of accepted range.
- Credentials: failed authentication because identity is not recognized, keys are invalid, or the user is not authorized to use a particular service.
- Unspecified: Any other reason that the server cannot establish a session.

EstablishmentReject				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10012	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
RequestTimestamp	nanotime	Y		Matches Establish.Timestamp
Code	EstablishmentRejectCode	Y		
Reason	string	N		Reject reason details

3.7.4 *Session Establishment Sequence Diagram*

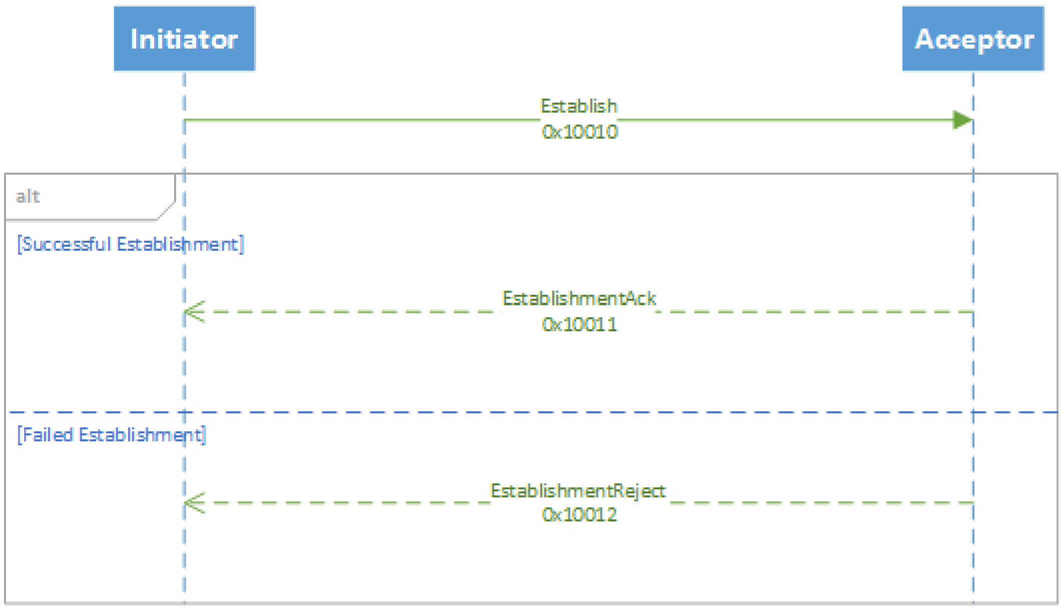

Figure 2 Session Establishment Sequence Diagram

3.8 *Transport termination*

Terminate is a signal to the counterparty that this side is dropping the binding between the logical session and the underlying transport. A session may terminate its transport if there are no more messages to send but it intends to restart at a later time. Additionally, a transport should be terminated if an unrecoverable error occurs in message parsing or framing, or if the counterparty violates this protocol.

An established session becomes terminated (stops being established) for the following reasons:

- One of the peers sends of receives a Terminate message;
- The transport level session is disconnected;
- The keep-alive interval expired and no keep-alive message received. It is recommended to allow some leniency in timeout to allow for slight mismatches of timers between parties.

TerminationCode = Finished | UnspecifiedError | ReRequestOutOfBounds | ReRequestInProgress

Terminate				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10015	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
Code	TerminationCode	Y		
Reason	string	N		

3.8.1 *Terminate Session Sequence Diagram*

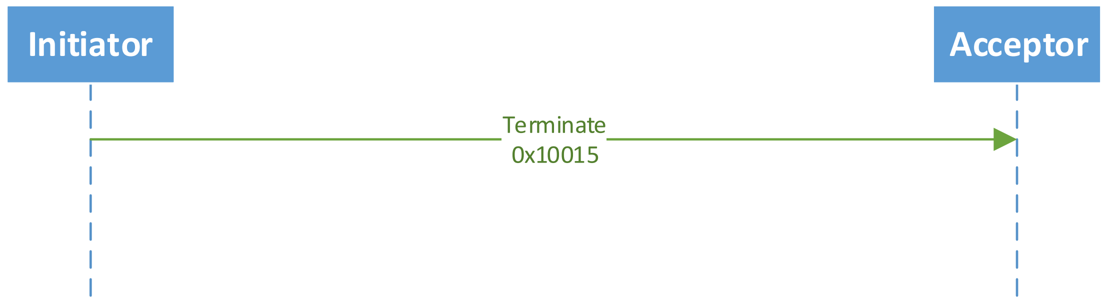

Figure 3 Termination Session Sequence Diagram

3.9 *Session heartbeat*

Each party must send a heartbeat message during each interval in which no application messages were sent. A client’s heartbeat timing is governed by the KeepaliveInterval value it sent in the Establish message, and a server is governed by the value it sent in EstablishAck.
Each party should check whether it has received any message from its counterparty in the expected interval. Silence is taken as evidence that the transport is no longer valid, and the session is terminated in that event.
For recoverable or idempotent flows, the gap detection can be achieved by sending Sequence messages respecting the keepalive interval.

For unsequenced flows, there is the UnsequencedHeartbeat message to detect that a logical session has disappeared or that there is a problem with the transport, allowing the peer to terminate session state timely and to potentially reestablish the session.

UnsequencedHeartbeat				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10020	

When a session is being finalized, but the FinishedReceiving message has not yet been received, then FinishedSending message is used as the heartbeat.

On TCP, it is recommended that Nagle algorithm be disabled to prevent the transmission of heartbeats and other messages from being delayed, potentially causing unnecessary session termination.

3.10 Resynchronization

When receiving a recoverable message flow, a peer may request sequenced messages to be retransmitted by sending a *RetransmitRequest* message, which is answered by one or more *Retransmission* messages (or with a *Terminate* message if the request is invalid).
Sending a RetransmitRequest to the sender of an idempotent or unsequenced flow is a protocol violation. In that case, the session should be terminated.

RestransmitRequest				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10021	
SessionId	UUID	Y		
Timestamp	nanotime	Y		
FromSeqNo	u64	Y		
Count	u32	Y		

Retransmission implies that the subsequent messages are sequenced without requiring that a Sequence message is passed. In a datagram oriented transport, Retransmission is passed in every single retransmission datagram.

Restransmission				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10022	
SessionId	UUID	Y		Defeats the need for Context when multiplexing
NextSeqNo	u64	Y		
RequestTimestamp	nanotime	Y		
Count	u32	Y		

In a datagram transport without builtin flow control, like UDP, resending many messages in a short time may result in packet loss. *RetransmitRequest* and *Retransmission* messages must therefore be

exchanged in a way that the correspondence is rate paced. Unless the underlying transport provides guaranteed delivery, the sender must limit how many messages are retransmitted per request to what can be sent in a single datagram.
In addition to resetting the implicit sequence number, the *Retransmission* message signals that the request has been or is being processed and also informs the consumer whether it needs to issue a follow up *RetransmitRequest* message.
The sender will terminate the session with the **ReRequestInProgress** code if it sees a premature retransmit request. If the RetransmitRequest is used to request messages that are not available a Terminate is issued with the **ReRequestOutOfBounds** code.

3.10.1 Retransmission Sequence Diagram

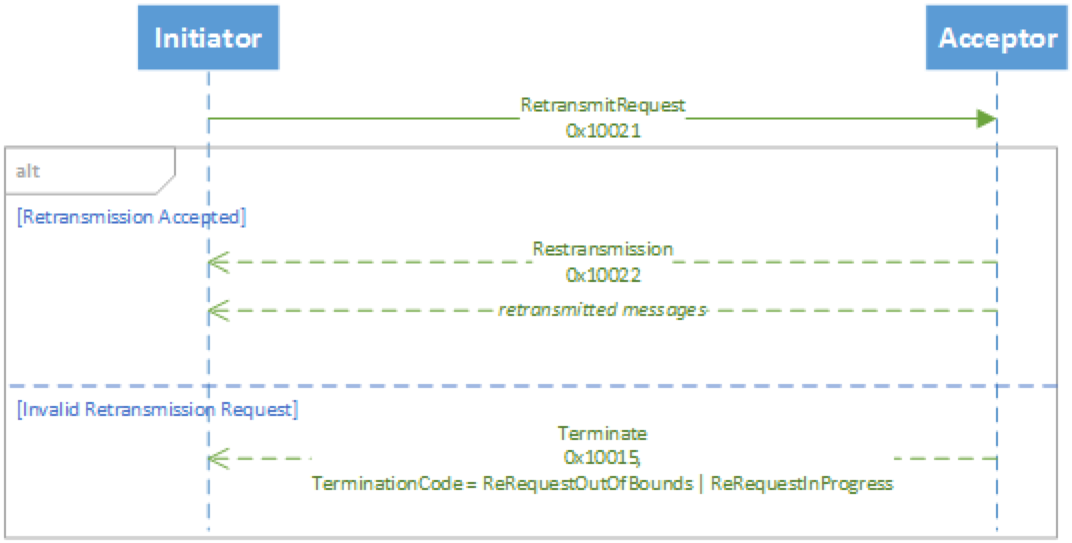
Initiator
Acceptor
RetransmitRequest
0x10021
alt
[Retransmission Accepted]
Restransmission
0x10022
retransmitted messages
[Invalid Retransmission Request]
Terminate
0x10015
TerminationCode = ReRequestOutOfBounds | ReRequestInProgress

Figure 4 Retransmission Sequence Diagram

3.11 Finalizing a Session

Finalization is a handshake that ends a logical session when there are no more messages to exchange.

3.11.1 Finish Sending

A FinishedSending message is sent to begin finalizing a logical session when the last application message in a flow has been sent.

The sender of this message awaits a FinishedReceiving response. It the wait takes longer than KeepaliveInterval for the flow, it should send FinishedSending messages as heartbeats until finalization is complete.

FinishedSending				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10025	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
LastSeqNo	u64	N		Populated for an idempotent or recoverable flow

The counterparty should evaluate LastSeqNo to determine whether it has processed the flow to the end. If received on a recoverable flow, the counterparty may send a RetransmitRequest to recover any missed messages before acknowledging finalization of the flow. On an idempotent flow, it should send NotApplied to notify the sender of the gap.

3.11.2 Finish Receiving

Upon processing the last application message indicated by the FinishedSending message (possibly received on a retransmission), a FinishedReceving message is sent in response.

When a FinishedReceiving has been received by the party that initiated the finalization handshake, a Terminate message is sent to unbind the transport. At that point, the session is considered finalized, and its session ID is no longer valid.

FinishedReceiving				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10026	
SessionId	UUID	Y		SessionId is redundant and included only for robustness

3.11.3 Terminating a Recoverable Session Sequence Diagram

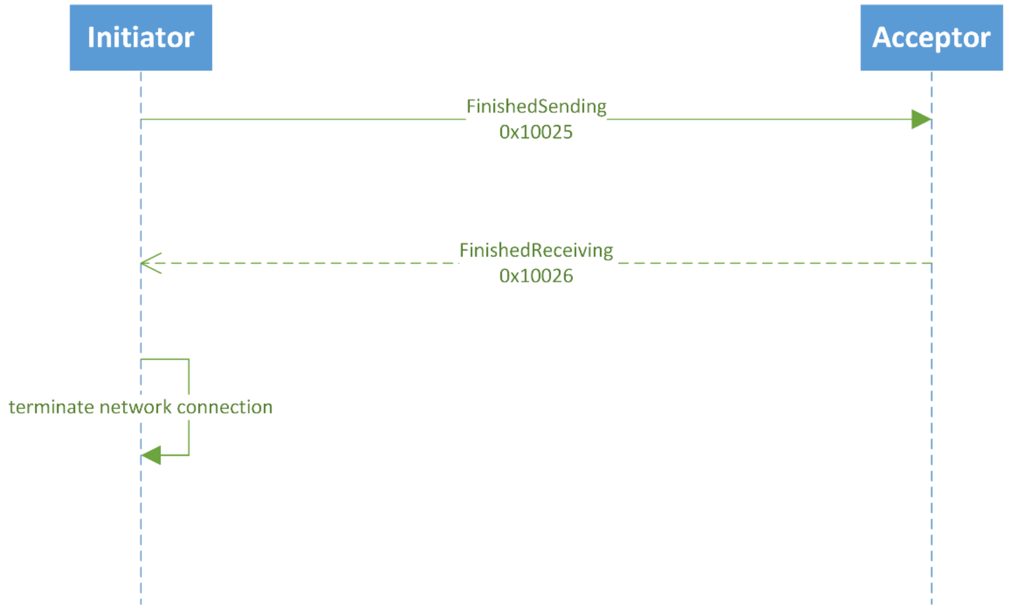

Figure 5 Recoverable Session Termination Sequence Diagram

3.12 Idempotent Flow

When using the idempotent flow, the protocol ensures that each application message is an idempotent operation that will be guaranteed applied only once.

To guarantee idempotence, a unique sequential identifier has to be allocated to each operation to be carried out. The response flow must identify which operations that have been carried out, and is sequenced. The lack of acknowledgment of an operation triggers the operation to be reattempted (at least once semantics). The lack of acknowledgment can be triggered by the acknowledgment of a later operation or by the expiration of a timer. The side carrying out an operation must filter out operations with a duplicate identifier (at most once semantics). If a transaction has already been applied, a duplicate request should be silently dropped. The combination of at-most-once and at-least-once semantics provide exactly-once semantics, making any operation tagged with a unique id to be idempotent.

The sequence number is implicit and is defined using a Sequence message. The first message after Sequence has the sequence number NextSeqNo. The same lifetime rules apply for the implicit sequence number in the idempotent flow, as for the implicit sequence number in the recoverable flow.
Unless the recoverable server return flow identifies the result of operations at the application level, implementers may opt to use the following *Applied* or *NotApplied* messages to return the status of the operation.

3.12.1 Applied

This is an optional application response for non-standard messages. Standard FIX semantics provide application layer acknowledgements to requests, e.g. Execution Report in response to New Order Single. The principle is to use application specific acknowledgement messages where possible; use the Applied message where an application level acknowledgement message does not exist.

Since Applied is an application message, it will be reliably delivered if returned on a recoverable flow.

Applied				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10201	
FromSeqNo	u64	Y		The first applied sequence number
Count	u32	Y		How many messages have been applied

3.12.2 NotApplied

NotApplied response is given if a receiver recognizes a gap in sequence numbers on an idempotent flow. The sender of the missed requests then has a responsibility to make a decision about recovery at an application layer. It may decide to resend the transactions with new sequence numbers, to send different transactions, or to do nothing.

Like Applied, the NotApplied message is handled as an application message. That is, it consumes a sequence number.

Sending NotApplied for a recoverable or unsequenced flow is a protocol violation. On a recoverable flow, RetransmitRequest should be used instead.

NotApplied				
Field name	**Type**	**Required**	**Value**	**Description**
MessageType	u32	Y	0x10202	
FromSeqNo	u64	Y		The first not applied sequence number
Count	u32	Y		How many messages haven´t been applied

3.12.3 Idempotent Flow Sequence Diagram

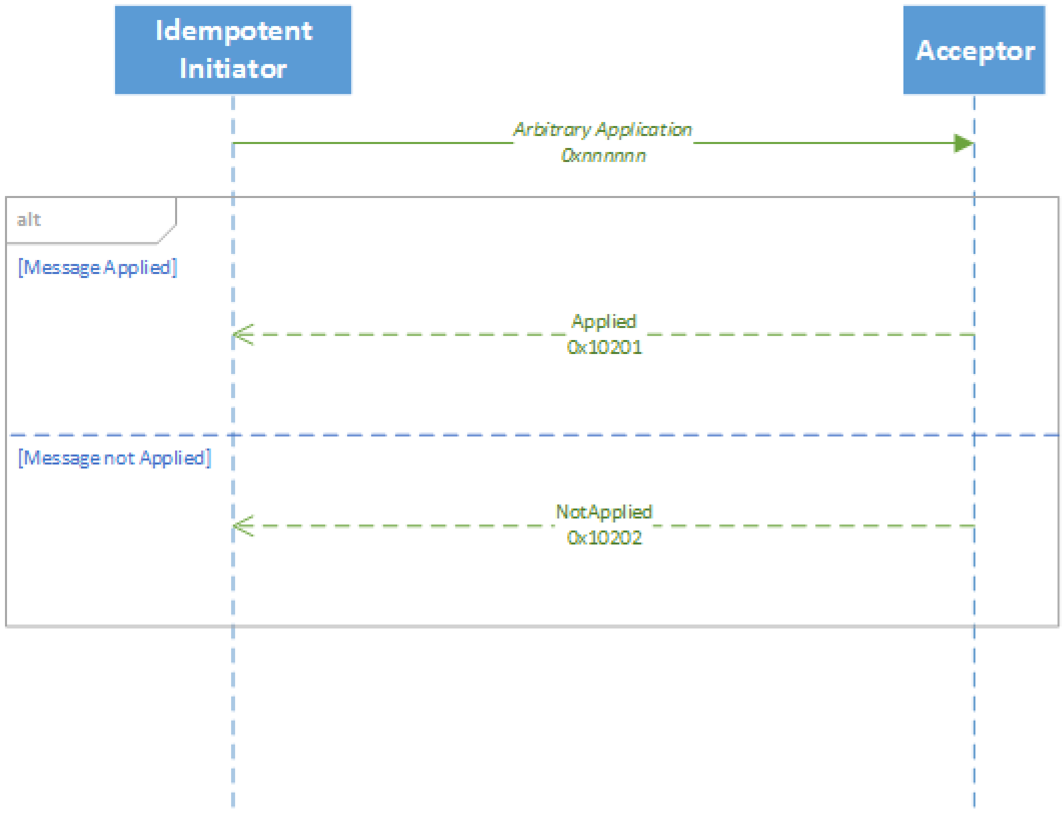

Figure 6 Idempotent Flow sequence diagram

4 Session Messages

Message Name	Message Type	Purpose
Sequence	0x01	Initiate a new session
Negotiate	0x10000	
NegotiationResponse	0x10001	
NegotiationReject	0x10002	
Establish	0x10010	
EstablishmentAck	0x10011	
EstablishmentReject	0x10012	
Terminate	0x10015	
UnsequencedHeartbeat	0x10020	
RetransmitRequest	0x10021	
Retransmission	0x10022	
FinishedSending	0x10025	
FinishedReceiving	0x10026	
Context	0x10050	Identify a context of a session
Applied	0x10201	Returns the status of idempotent operations
NotApplied	0x10202	Returns the status of idempotent operations

5 Appendix A - Usage Examples

5.1 Session Creation/Negotiation

5.1.1 Session negotiation (both Recoverable)

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		0x10000	ABC	T1	--	Recoverable	--	**123**
	Negotiation Response	0x10001	ABC	--	T1	--	Recoverable	--

5.1.2 Session negotiation (both Unsequenced)

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		0x10000	ABC	T1	--	Unsequenced	--	123
	Negotiation Response	0x10001	ABC	--	T1	--	Unsequenced	--

5.1.3 Session negotiation (Client Idempotent and Server Recoverable)

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		0x10000	ABC	T1	--	Idempotent	--	123
	Negotiation Response	0x10001	ABC	--	T1	--	Recoverable	--

5.1.4 Session negotiation (Client Recoverable and Server Unsequenced)

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		0x10000	ABC	T1	--	Recoverable	--	123
	Negotiation Response	0x10001	ABC	--	T1	--	Unsequenced	--

5.1.5 *Session negotiation (Client Unsequenced and Server Recoverable)*

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		0x10000	ABC	T1	--	Unsequenced	--	123
	Negotiation Response	0x10001	ABC	--	T1	--	Recoverable	--

5.1.6 *Session negotiation (rejects)*

5.1.6.1 **Bad credentials**

For example – Valid Credentials are 123 but Negotiate message is sent with Credentials as 456 then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		0x10000	ABC	T1	--	Idempot ent		--	456
	Negotiation Reject	0x10002	ABC	--	T1	--	Bad Credentials	Invalid Password	--

5.1.6.2 **Flow type not supported**

For example – Recoverable flow from Client is not supported but Negotiate message is sent with Client Flow as Recoverable then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		0x10000	ABC	T1	--	Recover able	--	--	123
	Negotiation Reject	0x10002	ABC	--	T1	--	FlowTypeNot Supported	Client Recoverable Flow Prohibited	--

5.1.6.3 **Invalid session ID**

For example – Session ID does not follow UUID or GUID semantics as per RFC 4122 and Negotiate message is sent with Session ID as all zeros then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		0x10000	000	0	--	Idempot ent	--	--	123
	Negotiation Reject	0x10002	000	--	0	--	Unspecified	Invalid SessionID Format	--

5.1.6.4 **Invalid request timestamp**

For example – Timestamp follows Unix Epoch semantics and is to be sent with nanosecond level precision but Negotiate message is sent with Timestamp as Unix Epoch but expressed as number of seconds then it will be rejected.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Client Flow**	**Code**	**Reason**	**Credentials**
Negotiate		0x10000	ABC	86400	--	Idempotent	--	--	123
	Negotiation Reject	0x10002	ABC	--	86400	--	Unspecified	Invalid Timestamp Format	--

5.1.6.5 **Mismatch of sessionID/RequestTimestamp**

For example – the session identifier and the request timestamp in the NegotiationResponse do not match with the Negotiate message then the acknowledgment MUST be ignored and an internal alert may be generated.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Client Flow**	**Server Flow**	**Credentials**
Negotiate		0x10000	ABC	T1	--	Recoverable	--	123
	Negotiation Response	0x10001	DEF	--	T2	--	Recoverable	--
<Ignore NegotiationResponse message since it contains incorrect Session ID and/or RequestTimestamp and Generate Internal Alert and Possibly Retry>								
Negotiate		0x10000	ABC	T3	--	Recoverable	--	123

5.2 Establishment and Reestablishment

*5.2.1 **Establishment (Recoverable)***

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Client Flow**	**Keep Alive Interval**	**Next SeqNo**	**Server Flow**
Negotiate		0x10000	ABC	T1	--	Recoverable	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	Recoverable
Establish		0x10010	ABC	T2	--	--	10	--	--
	EstablishmentAck	0x10011	ABC	--	T2	--	10	1	--

5.2.2 *Establishment (Unsequenced)*

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Client Flow**	**Keep Alive Interval**	**Next SeqNo**	**Server Flow**
Negotiate		0x10000	ABC	T1	--	Unsequenced	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	Unsequenced
Establish		0x10010	ABC	T2	--	--	10	--	--
	EstablishmentAck	0x10011	ABC	--	T2	--	10	--	--

5.2.3 *Establishment (idempotent)*

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Client Flow**	**Keep Alive Interval**	**Next SeqNo**	**Server Flow**
Negotiate		0x10000	ABC	T1	--	Idempotent	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	Recoverable
Establish		0x10010	ABC	T2	--	--	10	--	--
	EstablishmentAck	0x10011	ABC	--	T2	--	10	1	--

5.2.4 *Establishment rejects*

5.2.4.1 Unnegotiated

For example – Trying to send an Establish message without first Negotiating the session will result in the Establishment message being rejected.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Code**	**Reason**	**Keep Alive Interval**
Establish		0x10010	ABC	T2	--	--	--	10
	Establishment Reject	0x10012	ABC	--	T2	Unnegotiated	Establishment Not Allowed Without Negotiation	--

5.2.4.2 Already established

For example – Trying to send an Establish message when the session itself is already Negotiated and Established will result in the Establishment message being rejected.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Code**	**Reason**	**Keep Alive Interval**
Negotiate		0x10000	ABC	T1	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	ABC	T2	--	--	--	10
	Establish mentAck	0x10011	ABC	--	T2	--	--	10
Establish		0x10010	ABC	T3	--	--	--	10
	Establish mentReject	0x10012	ABC	--	T3	Already Established	Session is Already Established	--

5.2.4.3 **Session blocked**

For example – if a particular Session ID has been blocked for bad behavior and is not allowed to establish a session with the counterparty then also the Establishment message will be rejected.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Code**	**Reason**	**Keep Alive Interval**
Negotiate		0x10000	ABC	T1	--	--	--	--
	NegotiationR esponse	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	ABC	T2	--	--	--	10
	Establishmen tReject	0x10011	ABC	--	T2	Session Blocked	Session Has Been Blocked, Please Contact Market Operations	10

5.2.4.4 **Invalid keep alive interval**

For example – if the bilateral rules of engagement permit a KeepAliveInterval no smaller than 10 milliseconds then an Establishment message sent with a KeepAliveInterval of 1 millisecond will be rejected.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Code**	**Reason**	**Keep Alive Interval**
Negotiate		0x10000	ABC	T1	--	--	--	--
	NegotiationR esponse	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	ABC	T2	--	--	--	1
	Establishmen tReject	0x10011	ABC	--	T2	KeepAlive Interval	Invalid KeepAlive Interval	1

5.2.4.5 **Invalid session ID**

For example – Session ID does not follow UUID or GUID semantics as per RFC 4122 and Establishment message is sent with Session ID as all zeros then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		0x10000	ABC	T1	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	000	T2	--	--	--	10
	Establish mentReject	0x10011	000	--	T2	Unspecified	Invalid Session ID Format	10

5.2.4.6 **Invalid request timestamp**

For example – Timestamp follows Unix Epoch semantics and is to be sent with nanosecond level precision but Establishment message is sent with Timestamp as Unix Epoch but expressed as number of seconds then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		0x10000	ABC	T1	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	ABC	86400	--	--	--	10
	Establish mentReject	0x10011	ABC	--	86400	Unspecified	Invalid Timestamp Format	10

5.2.4.7 **Bad credentials**

For example – Valid Credentials are 123 but Establishment message is sent with Credentials as 456 then it will be rejected.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Code	Reason	Credentials
Negotiate		0x10000	ABC	T1	--	--	--	123
	NegotiationR esponse	0x10001	ABC	--	T1	--	--	--
Establish		0x10010	ABC	T2	--	--	--	456
	Establishmen tReject	0x10011	ABC	--	T2	Bad Credentials	Invalid Password	--

5.2.4.8 **Mismatch of sessionID/RequestTimestamp**

For example – the session identifier and the request timestamp in the EstablishmentAck do not match with the Establishment message then the acknowledgment MUST be ignored and an internal alert may be generated.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		0x10000	ABC	T1	--	Sequenced	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	Recoverable
Establish		0x10010	ABC	T2	--	--	10	--	--
	EstablishmentAck	0x10011	DEF	--	T3	--	10	1	--
<Ignore EstablishmentAck message since it contains incorrect Session ID and/or RequestTimestamp and Generate Internal Alert and Possibly Retry>									
Establish		0x10010	ABC	T4	--	--	10	--	--

5.3 Termination

5.3.1 Time out

When the KeepAliveInterval has expired and no keep alive message is received then the session is terminated and will need to be re-established. The transport level connection is still open (TCP socket) therefore Negotiation is not required.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Code	Reason
Negotiate		0x10000	ABC	T1	--	Recoverable	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	--
Establish		0x10010	ABC	T2	--	--	10	--	--
	EstablishmentAck	0x10011	ABC	--	T2	--	10	--	--
<Time Interval Greater Than Keep Alive Interval Has Lapsed Without Any Message Being Received>									
	Terminate	0x10015	ABC	--	--	--	--	Timed Out	Keep Alive Interval Has Lapsed
Establish		0x10010	ABC	T3	--	--	10	--	--
	EstablishmentAck	0x10011	ABC	--	T3	--	10	--	--

5.3.2 Deliberate termination

When the session has been deliberately terminated for example due to invalid RetransmitRequest then it will need to be re-established. The transport level connection is still open (TCP socket) therefore Negotiation is not required.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	From SeqNo	Count	Code	Reason
Negotiate		0x10000	ABC	T1	--	--	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	--	--
Establish		0x10010	ABC	T2	--	--	--	--	--	--
	Establish mentAck	0x10011	ABC	--	T2	1000	--	--	--	--
Retransmit Request		0x10010	ABC	T3	--	--	1	1000	--	--
	Terminate	0x10015	ABC	--	--	--	--	--	ReRequest OutOf Bounds	Resent Request Exceeds Maximum
Establish		0x10010	ABC	T4	--	--	--	--	--	--
	Establish mentAck	0x10011	ABC	--	T4	1000	--	--	--	--

5.3.3 *Disconnection*

When the transport level session itself (TCP socket) has been disconnected then the session needs to be Negotiated and Established.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Code	Reason
Negotiate		0x10000	ABC	T1	--	Recoverable	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	--
Establish		0x10010	ABC	T2	--	--	10	--	--
	Establish mentAck	0x10011	ABC	--	T2	--	10	--	--
<TCP socket connection is disconnected>									
Negotiate		0x10000	ABC	T3	--	Recoverable	--	--	--
	Negotiation Response	0x10001	ABC	--	T3	--	--	--	--
Establish		0x10010	ABC	T4	--	--	10	--	--
	Establish mentAck	0x10011	ABC	--	T4	--	10	--	--

5.4 *Sequence*

5.4.1 *Recoverable flow*

Over TCP – a Client would send a Sequence message at the very beginning of the session upon establishment. The counterparty would not use it initially as it is provided in the EstablishmentAck message.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo
Negotiate		0x10000	ABC	T1	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--
Establish		0x10010	ABC	T2	--	--	--
	EstablishmentAck	0x10011	ABC	--	T2	1000	--
Sequence		0x01	--	--	--	100	--
NewOrder Single		0x02	ABC	T3	--	--	100
	ExecutionReport	0x03	ABC	T4	--	--	1000

5.4.1.1 **Higher sequence number**

The Sequence, Context, EstablishmentAck and Retransmission messages are sequence forming. They turn the message flow into a sequenced mode since they have the next implicit sequence number. Any other Session message makes the flow leave the sequenced mode. If the message is sequence forming then the flow does not leave the sequenced mode, but the message potentially resets the sequence numbering.
For example – here the second Sequence message is increasing the NextSeqNo even though it was sent as a keep alive message within a sequenced flow.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	From SeqNo	Count
Negotiate		0x10000	ABC	T1	--	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	--
Establish		0x10010	ABC	T2	--	--	--	--	--
	Establishment Ack	0x10011	ABC	--	T2	1000	--	--	--
Sequence		0x01	--	--	--	100	--	--	--
NewOrder Single		0x02	ABC	T3	--	--	100	--	--
	Execution Report	0x03	ABC	T4	--	--	1000	--	--
Sequence		0x01	--	--	--	200	--	--	--
NewOrder Single		0x02	ABC	T5	--	--	200	--	--
	NotApplied	0x10202	--	--	--	--	--	101	100
	Execution Report	0x03	ABC	T6	--	--	1001	--	--

5.4.1.2 **Lower sequence number**

This is an example of a Sequence message being sent with a lower than expected NextSeqNo value even though it was sent as a keep alive message within a sequenced flow.

Message Received	**Message Sent**	**Message Type**	**Session ID (UUID)**	**Timestamp**	**Request Timestamp**	**Next SeqNo**	**Implicit SeqNo**	**From SeqNo**	**Count**
Negotiate		0x10000	ABC	T1	--	--	--	--	--
	Negotiation Response	0x10001	ABC	--	T1	--	--	--	--
Establish		0x10010	ABC	T2	--	--	--	--	--
	EstablishmentAck	0x10011	ABC	--	T2	1000	--	--	--
Sequence		0x01	--	--	--	100	--	--	--
NewOrder Single		0x02	ABC	T3	--	--	100	--	--
	ExecutionReport	0x03	ABC	T4	--	--	1000	--	--
Sequence		0x01	--	--	--	50	--	--	--
	NotApplied	0x10202	--	--	--	--	--	50	1

5.5 Multiplexing sessions

5.5.1 Context

5.5.1.1 Context flow

The Context message is needed to convey that a context switch is taking place from one Session ID (ABC) to another (DEF) over the same transport. This way – two FIX sessions (ABC & DEF) could be multiplexed over one TCP connection and there is a one to one relation between the two such that they need to be negotiated and established independently. They will have independent sequence numbering and the EstablishmentAck response will depend on where the particular session is sequence wise. There is no need to send a Context message before an application message if the previous application message was destined for the same session. A Context message has to be sent before an application message if the previous application message was destined for another session. This is an example where a Context message is necessary since the previous message was for a different session.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Next Seq No	Implicit SeqNo
Negotiate		0x10000	ABC	T1	--	--	--
	NegotiationResponse	0x10001	ABC	--	T1	--	--
Establish		0x10010	ABC	T2	--	--	--
	EstablishmentAck	0x10011	ABC	--	T2	1000	--
Negotiate		0x10000	DEF	T3	--	--	--
	NegotiationResponse	0x10001	DEF	--	T3	--	--
Establish		0x10010	DEF	T4	--	--	--
	EstablishmentAck	0x10011	DEF	--	T4	2000	--
Context		0x10050	ABC	--	--	100	--
NewOrder Single		0x02	ABC	T5	--	--	100
	Context	0x10050	ABC	--	--	1000	--
	ExecutionReport	0x03	ABC	T6	--	--	1000
Context		0x10050	DEF	--	--	200	--
NewOrder Single		0x02	DEF	T7	--	--	200
	Context	0x10050	DEF	--	--	2000	--
	ExecutionReport	0x03	DEF	T8	--	--	2000

5.5.1.2 Context flow using sequence

This is an example where a Context message is not necessary since the previous message was for the same session and a Sequence message will suffice.

Message Received	Message Sent	Message Type	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo
Negotiate		0x10000	ABC	T1	--	--	--
	NegotiationResponse	0x10001	ABC	--	T1	--	--
Establish		0x10010	ABC	T2	--	--	--
	EstablishmentAck	0x10011	ABC	--	T2	1000	--
Sequence		0x01	--	--	--	100	--
NewOrder Single		0x02	ABC	T3	--	--	100
	ExecutionReport	0x03	ABC	T4	--	--	1000
Negotiate		0x10000	DEF	T5	--	--	--
	NegotiationResponse	0x10001	DEF	--	T5	--	--
Establish		0x10010	DEF	T6	--	--	--
	EstablishmentAck	0x10011	DEF	--	T6	2000	--
Sequence		0x01	--	--	--	200	--
NewOrder Single		0x02	DEF	T7	--	--	200
	ExecutionReport	0x03	DEF	T8	--	--	2000

6 Appendix B – FIXP Rules of Engagement

This checklist is an aid to specifying a full protocol stack to be used for communication between counterparties

Stack layer	Client	Server
Application Layer		
Application level recovery supported?		
FIX version Service pack Extension packs		
Presentation Layer		
Message encoding Version Schema/templates	☐ Simple Binary Encoding ☐ GPB ☐ ASN.1 ☐ FIX tag-value	
Framing	☐ Simple Open Framing Header ☐ None	
Session Layer		
Supported flow types	☐ Recoverable ☐ Idempotent ☐ Unsequenced	☐ Recoverable ☐ Idempotent ☐ Unsequenced
Security protocols Authentication		
Transport Layer		
Transports supported	☐ TCP ☐ UDP	
Other network protocols		