FIX Performance Session Layer Technical Specification

Version 1.1 – Draft Standard – April 18, 2019

THIS DOCUMENT IS A DRAFT STANDARD FOR A PROPOSED FIX TECHNICAL STANDARD. A DRAFT STANDARD HAS BEEN APPROVED BY THE GLOBAL TECHNICAL COMMITTEE AS THE FINAL STEP IN CREATING A NEW FIX TECHNICAL STANDARD. POTENTIAL ADOPTERS ARE STRONGLY ENCOURAGED TO BEGIN WORKING WITH THE DRAFT STANDARD AND TO PROVIDE FEEDBACK TO THE GLOBAL TECHNICAL COMMITTEE AND THE WORKING GROUP THAT SUBMITTED THE PROPOSAL. THE FEEDBACK TO THE DRAFT STANDARD WILL DETERMINE WHEN TWO INTEROPERABLE IMPLEMENTATIONS HAVE BEEN ESTABLISHED AND THE DRAFT STANDARD CAN BE PROMOTED TO BECOME A NEW FIX TECHNICAL STANDARD.

DISCLAIMER

THE INFORMATION CONTAINED HEREIN AND THE FINANCIAL INFORMATION EXCHANGE PROTOCOL (COLLECTIVELY, THE "FIX PROTOCOL") ARE PROVIDED "AS IS" AND NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL MAKES ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO THE FIX PROTOCOL (OR THE RESULTS TO BE OBTAINED BY THE USE THEREOF) OR ANY OTHER MATTER AND EACH SUCH PERSON AND ENTITY SPECIFICALLY DISCLAIMS ANY WARRANTY OF ORIGINALITY, ACCURACY, COMPLETENESS, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SUCH PERSONS AND ENTITIES DO NOT WARRANT THAT THE FIX PROTOCOL WILL CONFORM TO ANY DESCRIPTION THEREOF OR BE FREE OF ERRORS. THE ENTIRE RISK OF ANY USE OF THE FIX PROTOCOL IS ASSUMED BY THE USER.

NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL SHALL HAVE ANY LIABILITY FOR DAMAGES OF ANY KIND ARISING IN ANY MANNER OUT OF OR IN CONNECTION WITH ANY USER'S USE OF (OR ANY INABILITY TO USE) THE FIX PROTOCOL, WHETHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL (INCLUDING, WITHOUT LIMITATION, LOSS OF DATA, LOSS OF USE, CLAIMS OF THIRD PARTIES OR LOST PROFITS OR REVENUES OR OTHER ECONOMIC LOSS), WHETHER IN TORT (INCLUDING NEGLIGENCE AND STRICT LIABILITY), CONTRACT OR OTHERWISE, WHETHER OR NOT ANY SUCH PERSON OR ENTITY HAS BEEN ADVISED OF, OR OTHERWISE MIGHT HAVE ANTICIPATED THE POSSIBILITY OF, SUCH DAMAGES.

DRAFT OR NOT RATIFIED PROPOSALS (REFER TO PROPOSAL STATUS AND/OR SUBMISSION STATUS ON COVER PAGE) ARE PROVIDED "AS IS" TO INTERESTED PARTIES FOR DISCUSSION ONLY. PARTIES THAT CHOOSE TO IMPLEMENT THIS DRAFT PROPOSAL DO SO AT THEIR OWN RISK. IT IS A DRAFT DOCUMENT AND MAY BE UPDATED, REPLACED, OR MADE OBSOLETE BY OTHER DOCUMENTS AT ANY TIME. THE FIX GLOBAL TECHNICAL COMMITTEE WILL NOT ALLOW EARLY IMPLEMENTATION TO CONSTRAIN ITS ABILITY TO MAKE CHANGES TO THIS SPECIFICATION PRIOR TO FINAL RELEASE. IT IS INAPPROPRIATE TO USE FIX WORKING DRAFTS AS REFERENCE MATERIAL OR TO CITE THEM AS OTHER THAN “WORKS IN PROGRESS”. THE FIX GLOBAL TECHNICAL COMMITTEE WILL ISSUE, UPON COMPLETION OF REVIEW AND RATIFICATION, AN OFFICIAL STATUS ("APPROVED") OF/FOR THE PROPOSAL AND A RELEASE NUMBER.

No proprietary or ownership interest of any kind is granted with respect to the FIX Protocol (or any rights therein).

Contents

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 Trading Community. FIXP is a derived work. The origin and basis for FIXP are the FIX session layer protocols and those designed and implemented by NASDAQ OMX, i.e. 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.

1.1 FIXP features

- Very lightweight session layer with no restrictions on the application layer
- Encoding independent supporting binary protocols
- Transport independent supporting both stream, datagram, and message oriented protocols
- Point-to-point as well as multicast patterns, sharing common primitives
- Negotiable delivery guarantees that may be asymmetrical

1.2 Authors

Name	Affiliation	Contact	Role
Anders Furuhed	Goldman Sachs	anders.furuhed@gs.com	Protocol Designer
David Rosenborg	Goldman Sachs	david.rosenborg@gs.com	Protocol Designer
Rolf Andersson	Goldman Sachs	rolf.andersson@gs.com	Contributor
Jim Northey	LaSalle Technology	jim.northey@fintechstandards.us	Global Technical Committee co-chair
Júlio L R Monteiro	formerly B3	juliolrmonteiro@gmail.com	Editor, Working Group convener
Aditya Kapur	CME Group, Inc	Aditya.kapur@cmegroup.com	Contributor
Don Mendelson	Silver Flash LLC	donmendelson@silver-flash.net	Working Group Lead
Li Zhu	Shanghai Stock Exchange	lzhu@sse.com.cn	Contributor

1.2.1 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	v1.0 Draft Standard	SOFH	Optional usage at presentation layer	Yes
FIX message specifications	5.0 SP2	FIX 5.0 SP2	Presentation and application layers	No
FIX Simple Binary Encoding	Version 1.0	Simple Binary Encoding	Optional usage at presentation layer	No
Encoding FIX Using ASN.1	v1.0 Draft Standard	ASN.1	Optional usage at presentation layer	No
Encoding FIX Using GPB	v1.0 Draft Standard	GPB	Optional usage at presentation layer	No
FIX-over-TLS (FIXS)	v1.0 Draft Standard	FIXS	Security guidelines	Yes

1.2.2 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 6455 WebSocket Protocol	N/A	http://tools.ietf.org/html/rfc6455	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 Specification terms

These key words in this document are to be interpreted as described in Internet Engineering Task Force RFC2119. These terms indicate an absolute requirement for implementations of the standard: "**must**", or "**required**".

This term indicates an absolute prohibition: "**must not**".

These terms indicate that a feature is allowed by the standard but not required: "**may**", "**optional**". An implementation that does not provide an optional feature must be prepared to interoperate with one that does.

These terms give guidance, recommendation or best practices: "**should**" or "**recommended**". A recommended choice among alternatives is described as "**preferred**".

These terms give guidance that a practice is not recommended: "**should not**" or "**not recommended**".

1.4 Definitions

Term	Definition
Client	Initiator of session
Credentials	User identification for authentication
Flow	A unidirectional stream of messages. Each flow has one producer and one or more consumers.
Idempotence	Idempotence means that an operation that is applied multiple times does not change the outcome, the result, after the first time
Multicast	A method of sending datagrams from one producer to multiple consumers
IETF	Internet Engineering Task Force
Server	Acceptor of session
Session	A dialog for exchanging application messages between peers. An established point-to-point session consists of a pair of flows, one in each direction between peers. A multicast session consists of a single flow from the producer to multiple consumers.
TCP	Transmission Control Protocol is a set of IETF standards for a reliable stream of data exchanged between peers. Since it is connection oriented, it incorporates some features of a session protocol.
TLS	Transport Layer Security is a set of IETF standards to provide security to a session. TLS is a successor to Secure Sockets Layer (SSL).
UDP	User Datagram Protocol is a connectionless transmission for delivering packets of data. Any rules for a long-lived exchange of messages must be supplied by a session protocol.
WebSocket	An IETF protocol that consists of an opening handshake followed by basic message framing, layered over TCP. May be used with TLS.

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, Idempotent (operations guaranteed to be applied only once), and None (for a one-way flow of messages).

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 Security Mechanisms

FIXP does not specify its own security features. Rather, the FIX Trading Community separately issues security requirements and recommendations that may apply to FIXP and other FIX session protocols. Due to the ever-changing nature of information security, the requirements and recommendations are likely to be updated periodically. In general, it is recommended that FIX traffic be protected by using

proven controls specified by the FIX Trading Community. See the FIX-over-TLS (FIXS) standard (reference listed in section 1).

The FIX Trading Community is in the process of specifying how to authenticate parties using TLS and optionally using FIX credentials. FIX credentials can be used to authenticate a client after an underlying TLS session has been established. FIXP supports this use case by providing a field for credentials in the FIXP session negotiation handshake.

2.2.3 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 Common Features

3.1 Usage and Naming Conventions

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

3.2 Data Types

Data types used in this standard are abstract. The terminology used to define them are to be interpreted as described in international standard ISO/IEC 11404 Information technology -- General-Purpose Datatypes .

It defines a set of datatypes, independent of any particular programming language specification or implementation, that is rich enough so that any common datatype in a standard programming language or service package can be mapped to some datatype in the set.

Actual wire format of FIXP is left to the presentation layer implementation.

FIXP Type	Description	General Purpose Type	Properties
u8	Unsigned number	Integer	Ordered, exact, numeric, bounded. Range $0..2^8-1$
u16	Unsigned number	Integer	Ordered, exact, numeric, bounded. Range $0..2^{16}-1$
u32	Unsigned number	Integer	Ordered, exact, numeric, bounded. Range $0..2^{32}-1$
u64	Unsigned number	Integer	Ordered, exact, numeric, bounded. Range $0..2^{64}-1$
UUID	RFC 4122 version 4 compliant unique identifier	Octet string	Fixed size 16.
String	Text	Character string	Unordered, exact, non-numeric, denumerable. Parameterized by character set.
nanotime	Time in nanoseconds	Date-and-Time	Ordered, exact, numeric, bounded. Time-unit = nanosecond. Same range as u64.
DeltaMillisecs	Number of milliseconds	Time interval	Ordered, exact, numeric, bounded. Time-unit = millisecond. Same range as u32.
Object	Unspecified data content	Octet string	Unordered, exact, non-numeric, denumerable.

FIXP Type	Description	General Purpose Type	Properties
Enumeration	A finite set of values. Error and message type identifiers are enumerated by symbolic name in this specification.	State	Unordered, exact, non-numeric. The value space of a state datatype is the set comprising exactly the named values in the state-value-list, each of which is designated by a unique state-literal.

3.3 FIXP Session Messages

The FIXP protocol defines several messages that are used to establish and tear down sessions and control sequencing of messages for delivery. Message layouts are specified in this document by symbolic names and the abstract data types listed above. Wire format details are provided by supplements to this specification for each of the supported FIX encodings.

Those supplements also explain how to distinguish session messages from application messages in that specific encoding. FIXP does not require that application messages be in the same encoding as session messages. With the use of Simple Open Framing Header to identify the encoding of the following message, it is even possible to mix wire formats in a session. However, a common encoding for all messages likely permits simpler implementation.

3.3.1 Message Type Identification

Message types are listed in this document as an enumeration of symbolic names. Each FIX encoding tells how message type identifiers are encoded on the wire.

See section 4 below for an enumeration of message types.

3.3.2 Fields

Exact wire format is determined by a presentation layer protocol (message encoding). However, fields should be encoded in the same order that they are listed in this specification.

3.3.3 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.

Message-oriented protocols such as WebSocket obviate the need for additional framing protocol.

3.4 Session Properties

3.4.1 Session Identification

Each session must be 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 session identifier must be unique, not only amongst currently active sessions, but for all time. Reusing a session ID is a protocol violation.

3.4.2 User Identification

The FIX Trading Community is in the process of specifying how to authenticate counterparties. This is expected to primarily using TLS and, optionally, using TLS in conjunction with FIX credentials. FIX credentials can be used after a TLS transport has been established, whilst its FIXP session is being established. In any event, the security features will be specified outside of FIXP, but may make use of FIXP credentials.

FIXP does not dictate the format of user credentials. They are agreed between counterparties and should be documented in rules of engagement. The Credentials field in FIXP is of datatype Object (opaque data) so no restriction on its contents is imposed by the protocol.

3.4.3 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.

3.4.4 Flow Types

Each stream of application messages in one direction on a FIXP session is called a flow. FIXP supports configurable delivery guarantees for each flow. A bidirectional session may have asymmetrical flows.

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

- **Recoverable**: Guarantees exactly-once message delivery. If gaps are detected, then missed messages may be 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.
- **None**: No application messages should be sent in one direction of a session. If ClientFlow is None, then application messages flow only from server to client.

3.4.4.1 Flow Restrictions

All the flow types listed above are possible for a point-to-point session. Only one of the flows may be None, meaning that although the transport supports bidirectional transmissions, application messages flow in only one direction. By agreement between counterparties, only certain of these flow types may be supported for a particular service.

A multicast session only supports one flow from producer to consumers, and it is restricted to the Idempotent type, possibly with out-of-band recovery.

3.5 Message Sequencing

3.5.1 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 (or Context message described below) must be 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.

Sending a Sequence or Context message on an Unsequenced or None flow is a protocol violation.

Sequence

Sequence message must be used only in a Recoverable or Idempotent flow on a non-multiplexed transport.

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

3.5.2 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.

3.5.3 Multiplexed session considerations

If sessions are multiplexed over a transport, they should be framed independently. If a framing header is used, the same framing protocol must be used for all sessions on a multiplexed transport. There would be no practical way to delimit messages with mixed framing policies.

If flows are multiplexed over a transport, the transport does not imply the session. When multiplexing, the Context message expands Sequence to also specify the session being sequenced. 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 supersedes the role of Sequence by including NextSeqNo (optimizes away the Sequence that would otherwise follow).

Context

Context message must be used in a Recoverable or Idempotent flow on a multiplexed transport.

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

3.5.4 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.

3.5.5 Application Layer Sequencing

Application-layer sequencing may be used on an Unsequenced flow as an alternative to FIXP session-layer message sequencing. If used, each application message body must contain an identifier used to sequence messages, and the application provider must specify rules for out-of-order delivery and recovery.

3.6 In-band Template Delivery

FIXP is independent of the wire format of session and application messages. However, some message encodings are controlled by templates that must be shared between peers in order to interoperate. Therefore, FIXP provides a means to deliver templates or message schemas.

All FIX encodings that use a template or message schema are supported. They are identified by the same code registered for Simple Open Framing Header (SOFH).

Templates may be delivered either over a point-to-point or multicast session. MessageTemplate may be sent at any time. For a multicast, it is recommended to resend templates at intervals to support late joiners. It is assumed to apply to all sessions on a transport in the case of multiplexing.

MessageTemplate

Field name	**Type**	**Required**	**Value**	**Description**
MessageType	Enum	Y	MessageTemplate	
EncodingType	u32	Y		Identifier registered for SOFH
EffectiveTime	nanotime	N		Date-time that the template becomes effective. If not present, effective immediately.
Version	Object	N		Version and format description. Version may also be embedded in the template itself, depending on protocol.
Template	Object	Y		Content of the template or message schema

4 Point-to-Point Session Protocol

A point-to-point session between a client and server must be conducted over a bidirectional transport, such as TCP or UDP unicast. Point-to-point protocol is designed for private flows of information between organizations. Optionally, multiple sessions belonging to an organization may be multiplexed over a shared transport.

4.1 Summary of Messages that Control Lifetime

A logical session must be created by using a Negotiation message. The session ID must be sent in the Negotiation message and that ID is used for the lifetime of the session.

After negotiation is complete, the client must send 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 may re-establish a previous session after reconnecting without any further negotiation. Thus, Establish binds the session to the new transport instance.

To signal a peer that a disconnection is about to occur, a Terminate message should be 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 should be ended by sending a FinishedSending message. Thereafter, no more application messages should be sent. The peer must respond with FinishedReceiving when it has processed the last message, and then the transport must be 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.

4.2 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 negotiation dialog declares the types of message flow in each direction of a session.

4.2.1 Initiate Session Negotiation

Negotiate message is sent from client to server.

Negotiate

FlowType = Recoverable | Unsequenced | Idempotent | None

Field name	Type	Required	Value	Description
MessageType	Enum	Y	Negotiate	
SessionId	UUID	Y		Session Identifier

Field name	Type	Required	Value	Description
Timestamp	nanotime	Y		Time of request
ClientFlow	FlowType Enum	Y		Type of flow from client to server
Credentials	Object	N		Optional credentials to identify the client. Format to be determined by agreement between counterparties.

4.2.2 Accept Session Negotiation

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

To support mutual authentication, a server may return a Credentials field to the NegotiationResponse message. It conveys identification of the server back to the client. As for the Credentials field in the Negotiate message, the format should be determined by agreement of counterparties.

NegotiationResponse

FlowType = Recoverable | Unsequenced | Idempotent | None

Field name	Type	Required	Value	Description
MessageType	Enum	Y	NegotiationResponse	
SessionId	UUID	Y		Session Identifier
RequestTimestamp	nanotime	Y		Matches Negotiate.Timestamp
ServerFlow	FlowType Enum	Y		Type of flow from server to client
Credentials	Object	N		Optional credentials to identify the server. Format to be determined by agreement between counterparties.

4.2.3 Reject Session Negotiation

When a session cannot be created, a server must send 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, 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	Enum	Y	NegotiationReject	
SessionId	UUID	Y		Session Identifier
RequestTimestamp	nanotime	Y		Matches Negotiate.Timestamp
Code	NegotiationRejectCode Enum	Y		
Reason	string	N		Reject reason details

4.2.4 Session Negotiation Sequence Diagram

4.3 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.

4.3.1 Establish

The client must send Establish message to the server and await 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	Enum	Y	Establish	
SessionId	UUID	Y		Session Identifier
Timestamp	nanotime	Y		Time of request
KeepaliveInterval	DeltaMillisecs	Y		The longest time in milliseconds the client may 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		Optional credentials to identify the client.

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.

4.3.2 Establish Acknowledgment

Used to indicate the acceptor acknowledges the session. If the communication flow from this endpoint is recoverable, it should 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	Enum	Y	EstablishmentAck	
SessionId	UUID	Y		SessionId is included only for robustness, as matching RequestTimestamp is enough
RequestTimestamp	nanotime	Y		Must match Establish.Timestamp
KeepaliveInterval	DeltaMillisecs	Y		The longest time in milliseconds the server may wait before sending a keep alive message

Field name	Type	Required	Value	Description
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 .

4.3.3 Establish Reject

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

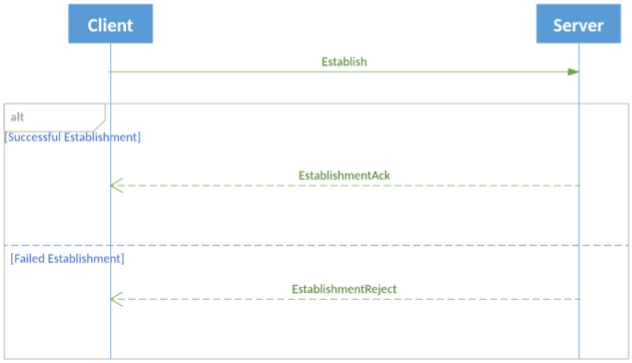
Rejection reasons:

- 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 because identity is not recognized, 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	Enum	Y	EstablishmentReject	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
RequestTimestamp	nanotime	Y		Must match Establish.Timestamp
Code	EstablishmentRejectCode Enum	Y		
Reason	string	N		Reject reason details

4.3.4 Session Establishment Sequence Diagram



4.4 Transport Termination

Terminate is a signal to the peer that a party intends to drop the binding between the logical session and the underlying transport. Either peer may terminate its transport if there are no more messages to send but it expects to re-establish the logical session at a later time.

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

- One of the peers receives a Terminate message (or Close frame on WebSocket)..
- The transport was abruptly 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.
- The peer violated this protocol. A specific example of protocol violation is to send a RetransmitRequest while another one is in progress.
- Additionally, a transport should be terminated if an unrecoverable error occurs in message parsing or framing.

No other messages may be sent on the session after sending a Terminate message. Any messages sent after Terminate are a protocol violation and should be ignored.

TerminationCode = Finished | UnspecifiedError | ReRequestOutOfBounds | ReRequestInProgress

Terminate

Field name	Type	Required	Value	Description
MessageType	Enum	Y	Terminate	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
Code	TerminationCode Enum	Y		
Reason	string	N		Reject reason details

4.4.1 Terminate Response

On a point-to-point session, the party that initiated termination should then wait for a response from its peer to permit in-flight messages to be processed. Upon receiving a Terminate message, the receiver must respond with a Terminate message. The Terminate response must be the last message sent.

If the peer is unresponsive to Terminate for a heartbeat interval, then the initiator of termination should consider the session terminated anyway.

4.4.2 Closing the Transport

On a non-multiplexed transport, when the party that initiated termination receives the Terminate response from its peer, it then should close the transport immediately.

On a multiplexed transport, the transport should be closed when the last session on that transport is terminated. When termination is the result of an unexpected transport disconnection, then all sessions on that transport are terminated.

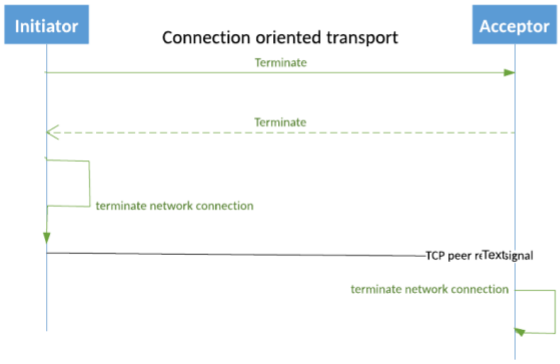
On a connectionless transport such as UDP, the Terminate message informs the peer that message exchange is suspended since there is no disconnection signal in the transport layer.

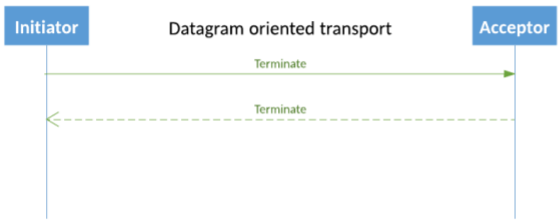
On a connection-oriented transport such as TCP, when the last peer that initiated termination receives a Terminate response, it should disconnect the socket from its end. Both peers then complete the transport close handshake.

4.4.3 WebSocket Termination

On a WebSocket transport, a Close frame is used instead of a Terminate message. See WebSocket Usage below.

4.4.4 Terminate Session Sequence Diagrams





4.5 Session Heartbeat

Each peer must send a heartbeat message during each interval in which no application messages were sent. A party may send a heartbeat before its interval has expired, for example to force its peer to check for a sequence number gap prior to sending a large batch of application messages.

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 peer in the expected interval. Silence is taken as evidence that the transport is no longer valid, and the session should be terminated in that event.

For recoverable or idempotent flows, the gap detection should be achieved by sending Sequence messages respecting the keepalive interval.

For Unsequenced and None (one-way session) 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	Enum	Y	UnsequencedHeartbeat	

When a session is being finalized, but the FinishedReceiving message has not yet been received, then FinishedSending message must be 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.

4.6 Resynchronization

The following sections describe resynchronization of a recoverable flow.

4.6.1 Retransmission Request

When receiving a recoverable message flow, a peer may request sequenced messages to be retransmitted by sending a *RetransmitRequest* message, which should be answered by one or more *Retransmission* messages (or with a *Terminate* message if the request is invalid).

Only one RetransmitRequest is allowed in-flight at a time per session. Another RetransmitRequest must not be sent until a response has been received from a previous request.

The receiver on a recoverable flow should accept messages with a higher sequence number after recognizing a gap. However, the application should queue messages for in-sequence processing after a requested retransmission is received.

Sending a RetransmitRequest to the sender of an Idempotent, Unsequenced or None flow is a protocol violation. In that case, the session must be terminated.

RestransmitRequest

Field name	Type	Required	Value	Description
MessageType	Enum	Y	RestransmitRequest	
SessionId	UUID	Y		
Timestamp	nanotime	Y		Timestamp used as a unique identifier of the request
FromSeqNo	u64	Y		Sequence number of the first message requested
Count	u32	Y		Count of messages requested

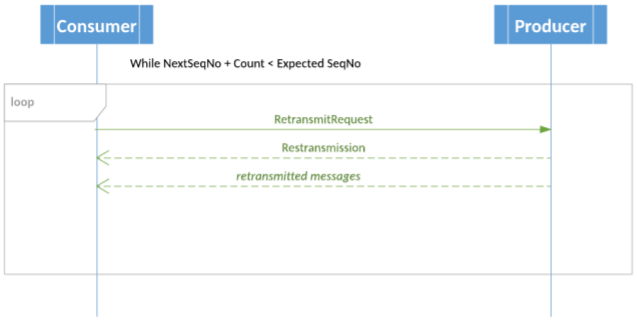
4.6.2 Retransmission Responses

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	Enum	Y	Restransmission	
SessionId	UUID	Y		Defeats the need for Context when multiplexing
RequestTimestamp	nanotime	Y		Value from RetransmitRequest Timestamp field. Used to match responses to requests.
NextSeqNo	u64	Y		Sequence number of the next message to be retransmitted
Count	u32	Y		Count of messages to be retransmitted in a batch

4.6.2.1 Retransmission Diagram



4.6.2.2 Interleaving and Pacing Retransmissions

This protocol does *not* require real-time messages to be held by the sender until retransmission of a range of messages is complete. Rather, ranges of retransmitted and real-time messages may be interleaved. Each time some messages are retransmitted, they must be preceded by a Retransmission message with a count of messages. Each time real-time flow resumes, a Sequence message (or Context message on a multiplexed flow) must be sent.

The provider of a recoverable flow need not retransmit all requested messages in a single batch. Rather, retransmission should be executed as an iterative process. It is the requester's responsibility to determine whether the current batch fills the original gap. If not, it should send another RetransmitRequest for the remainder. Requests and responses should proceed iteratively until all desired messages have been retransmitted. This interaction automatically paces the retransmission flow while allowing real-time messages to flow through uninhibited.

Pacing is the responsibility of the retransmitter. It is managed by controlling the size of batches of retransmitted messages. To maximize interleaving with real-time messages without queuing, it is recommended that messages be retransmitted in small batches. Optimally, a batch should not exceed to the size of a datagram, even on a TCP stream.

However, when retransmission is provided through a separate recovery session without interleaving real-time messages, then the retransmitter may choose to fulfill requests in a single batch.

4.6.2.3 Retransmit Rejection

If the provider of a recoverable flow is unable to retransmit requested messages, it responds with a RetransmitReject message.

RetransmitRejectCode = OutOfRange | InvalidSession | RequestLimitExceeded

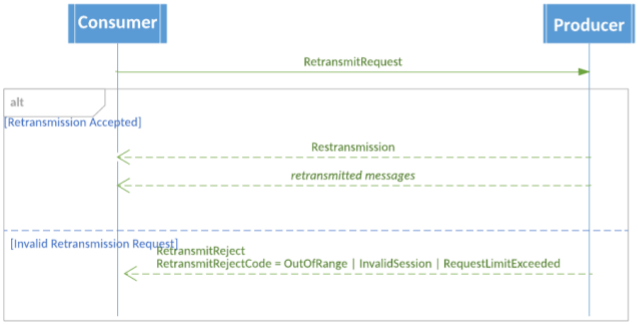
Rejection reasons:

- OutOfRange: NextSeqNo + Count is beyond the range of sequence numbers
- InvalidSession: The specified SessionId is unknown or is not authorized for the requester to access.
- RequestLimiitExceeded: The message Count exceeds a local rule for maximum retransmission size.

RestransmitReject

Field name	Type	Required	Value	Description
MessageType	Enum	Y	RestransmitReject	
SessionId	UUID	Y		Session identifier
RequestTimestamp	nanotime	Y		Value from RetransmitRequest Timestamp field. Used to match responses to requests.
Code	RetransmitRejectCode Enum	Y		
Reason	string	N		Reject reason details

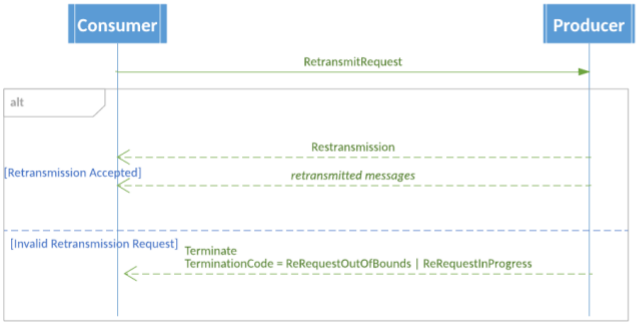
4.6.3 RetransmitReject Diagram



4.6.4 Retransmission Violations

For a RetransmitRequest that the requester should have known was invalid with certainty, the sender should terminate the session. Terminate message with **ReRequestInProgress** code should be sent if it sees a premature retransmit request.

4.6.5 Retransmit Violation Diagram



4.6.6 FIX Application Layer Recovery

As an alternative to a FIXP recoverable flow, application layer sequencing and recovery may be used. To avoid duplication of effort in two layers of the protocol stack, application layer sequencing should be used with a FIXP Unsequenced flow.

See FIX application specifications for a description of the ApplicationSequenceControl group and these message types:

- ApplicationMessageReport
- ApplicationMessageRequest
- ApplicationMessageRequestAck

4.7 Finalizing a Session

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

4.7.1 Finish Sending

A FinishedSending message should be 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. If 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	Enum	Y	FinishedSending	
SessionId	UUID	Y		SessionId is redundant and included only for robustness
LastSeqNo	u64	N		Must be populated for an idempotent or recoverable flow

The peer should evaluate LastSeqNo to determine whether it has processed the flow to the end. If received on a recoverable flow, the peer 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.

4.7.2 Finish Receiving

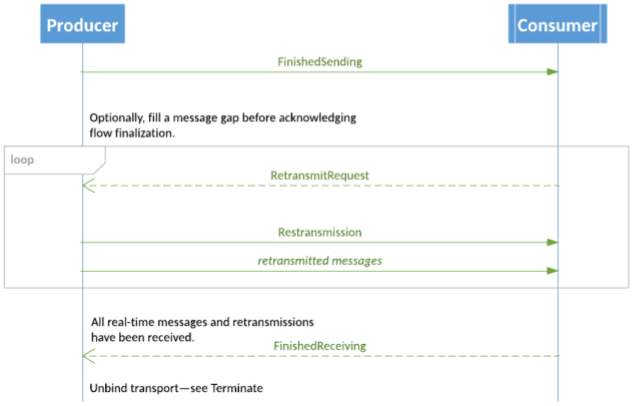
Upon processing the last application message indicated by the FinishedSending message (possibly received on a retransmission), a FinishedReceving message must be sent in response.

When a FinishedReceiving has been received by the party that initiated the finalization handshake, a Terminate message should be 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	Enum	Y	FinishedReceiving	
SessionId	UUID	Y		SessionId is redundant and included only for robustness

4.7.3 Terminating a Recoverable Session Sequence Diagram



4.8 Idempotent Flow

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

To guarantee idempotence, a unique sequential identifier must be allocated to each operation to be carried out. The response flow must identify which operations have been carried out, and is sequenced. The lack of acknowledgment of an operation should trigger the operation to be reattempted (at least once semantics). The lack of acknowledgment should 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 start of a idempotent flow must be initiated with a Sequence message (or Context message on a multiplexed transport) that explicitly provides the sequence number of the next application message in its field NextSeqNo. The first application message after a Sequence (or Context) message has the implicit sequence number NextSeqNo. For subsequent application messages, the sequence number is incremented implicitly. That is, the sequence number is not sent on the wire in every application

message, but rather, sender and receiver each should keep track of the next expected sequence number.

As explained in section 3, a Sequence or Context message must be sent after any context switch or once per packet on a Datagram oriented transport. Additionally, as explained in Session Heartbeat, they must be sent as hearbeats during idle periods. After every explicit NextSeqNo, the sequence number of subsequent application messages should be tracked implicitly.

The recoverable server return flow should report 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 if a more specific application message is not provided.

4.8.1 Applied

This is an optional application response message to support an idempotent flow. 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	Enum	Y	Applied	
FromSeqNo	u64	Y		The first applied sequence number
Count	u32	Y		How many messages have been applied

4.8.2 NotApplied

When a receiver on an idempotent flow recognizes a sequence number gap, it should send the NotApplied message immediately but continue to accept messages with a higher sequence number after the gap.

The sender on an idempotent flow uses the NotApplied message to discover which its requests have not been acted upon. It 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.

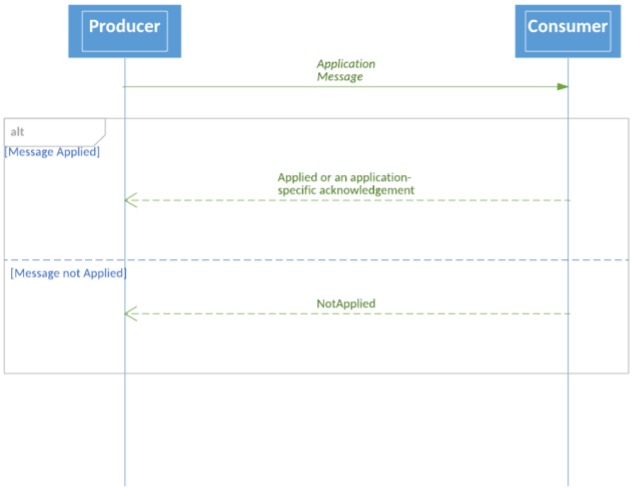
It is recommended that the return flow of an idempotent request flow be recoverable to allow Applied and NotApplied message to be resynchronized if necessary. Thus, the sender can determine with certainty (perhaps after some delay) which requests have been accepted.

Sending NotApplied for a Recoverable, Unsequenced or None flow is a protocol violation. On a recoverable flow, RetransmitRequest must be used instead.

NotApplied

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

4.8.3 Idempotent Flow Sequence Diagram



4.9 WebSocket Usage

WebSocket runs over TCP, so FIXP usage with WebSocket is largely the same as regular point-to-point session usage, with a few exceptions listed below.

4.9.1 Message Framing

WebSocket is a message-oriented protocol. That is, it performs message framing, so an additional framing protocol such as SOFH is unnecessary.

WebSocket has two defined subprotocols, text and binary. The appropriate subprotocol should be used depending on whether message encoding is character-oriented or binary.

4.9.2 Session Initiation

A WebSocket session is initiated by a client with an HTTP request and optionally, a TLS handshake. See the FIX-over-TLS (FIXS) standard, referenced in section 1, for recommendations about authentication and cipher suite selection.

4.9.3 Heartbeats

WebSocket protocol defines Ping and Pong frames to be used as keep-alives. However, their intervals and message contents are not precisely defined by the protocol, and implementations may vary widely in their behavior. Therefore, WebSocket Ping/Pong is not considered a suitable substitute for FIXP heartbeats (Sequence or Context messages) especially since they do not convey sequence numbers needed to rapidly detect gaps. Therefore, FIXP heartbeats should be used as specified above.

4.9.4 Termination

The FIXP Terminate message and WebSocket Close frame have practically the same behavior. In both cases, either side can initiate closing of a transport session and the other side responds with the same message type. No more messages may be sent after Terminate or Close. Therefore, only the WebSocket Close frame is needed to unbind the transport from a logical session. Normally, the status code of the Close frame is set to 1000 indicating a normal closure. Other error codes may be set as defined by the protocol.

5 Multicast Session Protocol

A multicast session conveys messages one way from a publisher to any number of listeners. It is conducted over a connectionless transport such UDP multicast. Multicast session protocol is typically used for publishing market data or common reference information to many consumers. Multiple independent flows may be multiplexed over a shared multicast transport.

5.1 Multicast Session Lifecycle

Since a multicast transport is connectionless, there is no negotiation or binding or unbinding of the transport as in the point-to-point protocol. Thus, Negotiation and Establishment messages and their respective responses are not used.

Multicast addresses and publishing schedules must be provided out-of-band to listeners. To capture all messages, listeners must be ready to receive at scheduled times. Publishing continues until the end of a logical flow.

5.1.1 Multicast Session Establishment

Like a point-to-point session, a multicast session is identified by a UUID. Each time a session is initiated, a new UUID must be generated, and sequence numbers of subsequent application messages must begin with 1.

5.1.1.1 Topic Message

To associate a transient UUID to a permanent or semi-permanent classification of messages, a Topic message must be used to initiate a flow. Multiple topics may be published on a transport.

FlowType = Recoverable | Idempotent

Valid flow types on a multicast session are:

- **Recoverable**: Messages are sequenced and recoverable. Since the transport is one-way, RetransmitRequests must be delivered through a separate session, however.
- **Idempotent**: Messages are sequenced to allow detection of loss but any missed messages are not recoverable.

Each Topic carries a Classification for the flow to associate it to a permanent or semi-permanent application layer entity. Typical classifications are product types, market symbols or the like.

To support late joiners, Topic messages should be repeated at regular intervals on a session. This specification does not dictate a specific interval, but the shorter the interval, the less time it takes for a late joiner to identify flows. It is recommended that Topic message be sent with Session heartbeats when the session is otherwise idle. See session heartbeat section below.

Topic

Field name	Type	Required	Value	Description
MessageType	Enum	Y	Topic	
SessionId	UUID	Y		Session Identifier

Field name	Type	Required	Value	Description
Flow	FlowType Enum	Y		Type of flow from publisher
Classification	Object	Y		Category of application messages that follow

5.1.2 Finalizing a Multicast Session

Finalization ends a logical flow when there are no more application messages to send. A multicast flow should be finalized by transmitting a FinishedSending message. No further messages should be sent, and the session ID is no longer valid after that.

5.2 Idempotent Flow over Multicast

The goal of an idempotent flow over multicast is to provide at-most-once delivery guarantee to each consumer. Unlike a point-to-point session, however, there is no opportunity to return a NotApplied message to the producer over a one-way transport if a sequence number gap is detected. Therefore, on a multicast, an idempotent flow provides a means to detect data loss, but no direct way to notify the producer or initiate recovery.

An idempotent flow is implemented by the producer in the same way over a multicast transport as for point-to-point over UDP unicast. Each datagram must begin with either a Sequence message if non-multiplexed or a Context message if the flow is sent over a multiplexed transport.

5.3 Session Heartbeat

During the lifetime of a multicast session, its publisher should send Sequence or Context messages as a heartbeat at regular intervals when the session is otherwise inactive. This allows a receiver to tell whether a session is live and has not reached the end of its logical flow. If only a single Topic is published, then Sequence message may be used for heartbeats since there is no context switch. If multiple topics are published on a shared multicast transport, then Context must be used. See the Common Features section above for a description of sequence numbering and the Sequence and Context messages.

In addition to the Sequence or Context message, it is recommended that a Topic message should be published on the heartbeat interval. This provides an opportunity for late joiners to gather session characteristics during every idle period. Summary of Session Messages

6 Summary of Session Messages

6.1 FIXP Session Messages

Stage	Message Name	Purpose	Recoverable	Idempotent	Unsequenced / None	Multicast
Initialization	Negotiate	Initiates session	•	•	•	
	NegotiationResponse	Accepts session	•	•	•	
	NegotiationReject	Rejects session	•	•	•	
	Topic	Announces a flow				•
Binding	Establish	Binds session to transport	•	•	•	
	EstablishmentAck	Accepts binding	•	•	•	
	EstablishmentReject	Rejects binding	•	•	•	
Transferring	Sequence	Initiates a sequenced flow, keep-alive	•	•		•
	Context	Multiplexing	•	•	•	•
	UnsequencedHeartbeat	Keep-alive			•	
	RetransmitRequest	Requests resynchronization	•			
	Retransmission	Resynchronization	•			
Unbinding	Terminate[1]	Unbinds a transport	•	•	•	
Finalizing	FinishedSending	Ends a logical flow	•	•	•	•
	FinishedReceiving	Ends a logical flow	•	•	•	•

6.2 Related Application Messages

These optional application messages respond to application messages on an idempotent flow.

Stage	Message Name	Purpose
Transferring	Applied	Acknowledge idempotent operations
	NotApplied	Negative acknowledgement of idempotent operations

[1] On WebSocket transport, Close frame is used instead of the Terminate message.

6.3 Summary of Protocol Violations

If any of these violations by a peer is detected, the session should be immediately terminated. Any application messages that cause a violation, such as a message sent after FinishedSending, should be ignored.

- Sending a session message that is inappropriate to the flow type, such as a Sequence message on an unsequenced flow. See table above.
- Sending an application message on a point-to-point session that is not in established state. That is, prior to EstablishmentAck.
- Sending Establish without having successfully negotiated a session. That is, a NegotiationResponse must have been received for the session.
- Sending an application message after logical flow has been finalized with FinishedSending. The responder on a point-to-point session may not send an application message after sending FinishedReceiving.
- Sending FinishedReceiving without having received FinishedSending from the peer.
- Sending any application or session message after sending Terminate.
- Reusing the session ID of a session that was finalized. (The server may have a practical limit of session history to enforce this rule.)
- Sending a RetransmitRequest while a retransmission is in progress.
- Sending a RetransmitRequest with request range out of bounds. That is, it is a violation to request a retransmission of a message with a sequence number that has not been sent yet.

7 Appendix A - Usage Examples (TCP)

These use cases contain sample values for illustrative purposes only

7.1 Initialization

7.1.1 Session negotiation (both Recoverable)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Recoverable	--	**123**
	NegotiationResponse	ABC	--	T1	--	Recoverable	**--**

7.1.2 Session negotiation (both Unsequenced)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Unsequenced	--	123
	NegotiationResponse	ABC	--	T1	--	Unsequenced	--

7.1.3 Session negotiation (Client Idempotent and Server Recoverable)

This use case is highly recommended.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Idempotent	--	123
	NegotiationResponse	ABC	--	T1	--	Recoverable	--

7.1.4 Session negotiation (Client None and Server Recoverable)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	None	--	123
	NegotiationResponse	ABC	--	T1	--	Unsequenced	--

7.1.5 Session negotiation (Client Unsequenced and Server Recoverable)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Unsequenced	--	123
	NegotiationResponse	ABC	--	T1	--	Recoverable	--

7.1.6 Session negotiation (Client None and Server Unsequenced)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	None	--	123
	NegotiationResponse	ABC	--	T1	--	Unsequenced	--

7.1.7 Session negotiation (rejects)

7.1.7.1 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	Session ID	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		ABC	T1	--	Idempotent		--	456
	NegotiationReject	ABC	--	T1	--	Bad Credentials	Invalid Trader ID	--

7.1.7.2 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	Session ID	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		ABC	T1	--	Recoverable	--	--	123

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
	NegotiationReject	ABC	--	T1	--	FlowTypeNotSupported	Client Recoverable Flow Prohibited	--

7.1.7.3 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	Session ID	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		000	0	--	Idempotent	--	--	123
	NegotiationReject	000	--	0	--	Unspecified	Invalid SessionID Format	--

7.1.7.4 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	Session ID	Timestamp	Request Timestamp	Client Flow	Code	Reason	Credentials
Negotiate		ABC	86400	--	Idempotent	--	--	123
	NegotiationReject	ABC	--	86400	--	Unspecified	Invalid Timestamp Format	--

7.1.7.5 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 followed by a new Negotiate message

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Recoverable	--	123
	NegotiationResponse	DEF	--	T2	--	Recoverable	--
<Ignore NegotiationResponse message since it contains incorrect Session ID and/or RequestTimestamp and Generate Internal Alert and Possibly Retry>							
Negotiate		XYZ	T3	--	Recoverable	--	123
<New Negotiate message should contain new Session ID>							

7.1.7.6 NegotiationResponse or Reject Not Received

For example – the Negotiate message is neither accepted nor rejected and one KeepAliveInterval* has lapsed then an internal alert may be generated followed by a new Negotiate message.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials
Negotiate		ABC	T1	--	Recoverable	--	123
<One KeepAliveInterval has lapsed without any response>							
Negotiate		XYZ	T3	--	Recoverable	--	123
<New Negotiate message should contain new Session ID>							

*Even though the KeepAliveInterval is part of the Establish message, generally speaking there will be a recommended value or range agreed to by the counterparties which can serve as a catch-all for these types of scenarios.

7.1.8 Establishment (Recoverable)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		ABC	T1	--	Recoverable	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	Recoverable
Establish		ABC	T2	--	--	10	1	--
	EstablishmentAck	ABC	--	T2	--	10	1	--

7.1.9 Establishment (Unsequenced)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		ABC	T1	--	Unsequenced	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	Unsequenced
Establish		ABC	T2	--	--	10	--	--
	EstablishmentAck	ABC	--	T2	--	10	--	--

7.1.10 Establishment (idempotent)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		ABC	T1	--	Idempotent	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	Recoverable
Establish		ABC	T2	--	--	10	1	--
	EstablishmentAck	ABC	--	T2	--	10	1	--

7.1.11 Establishment (none)

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		ABC	T1	--	None	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	None
Establish		ABC	T2	--	--	10	--	--
	EstablishmentAck	ABC	--	T2	--	10	--	--

7.1.12 Establishment rejects

7.1.12.1 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Establish		ABC	T2	--	--	--	10
	Establishment Reject	ABC	--	T2	Unnegotiated	Establishment Not Allowed Without Negotiation	--

7.1.12.2 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		ABC	T1	--	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--
Establish		ABC	T2	--	--	--	10

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
	EstablishmentAck	ABC	--	T2	--	--	10
Establish		ABC	T3	--	--	--	10
	EstablishmentReject	ABC	--	T3	Already Established	Session is Already Established	--

7.1.12.3 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		ABC	T1	--	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--
Establish		ABC	T2	--	--	--	10
	EstablishmentReject	ABC	--	T2	Session Blocked	Session Has Been Blocked, Please Contact Market Operations	10

7.1.12.4 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		ABC	T1	--	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Establish		ABC	T2	--	--	--	1
	EstablishmentReject	ABC	--	T2	KeepAlive Interval	Invalid KeepAlive Interval	1

7.1.12.5 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		ABC	T1	--	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--
Establish		000	T2	--	--	--	10
	EstablishmentReject	000	--	T2	Unspecified	Invalid Session ID Format	10

7.1.12.6 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
Negotiate		ABC	T1	--	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--
Establish		ABC	86400	--	--	--	10

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Code	Reason	Keep Alive Interval
	EstablishmentReject	ABC	--	86400	Unspecified	Invalid Timestamp Format	10

7.1.12.7 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	Session ID	Timestamp	Request Timestamp	Code	Reason	Credentials
Negotiate		ABC	T1	--	--	--	123
	NegotiationResponse	ABC	--	T1	--	--	--
Establish		ABC	T2	--	--	--	456
	EstablishmentReject	ABC	--	T2	Bad Credentials	Invalid Trader ID	--

7.1.12.8 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	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
Negotiate		ABC	T1	--	Idempotent	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	Recoverable
Establish		ABC	T2	--	--	10	--	--
	EstablishmentAck	DEF	--	T3	--	10	1	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Next SeqNo	Server Flow
<Ignore EstablishmentAck message since it contains incorrect Session ID and/or RequestTimestamp and Generate Internal Alert and Possibly Retry>								
Establish		ABC	T4	--	--	10	--	--
<New Establish message should contain same Session ID>								

7.1.12.9 EstablishmentAck or Reject Not Received

For example – the Establish message is neither accepted nor rejected and one KeepAliveInterval has lapsed then an internal alert may be generated followed by a new Establish message.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials	KeepAliveInterval
Negotiate		ABC	T1	--	Idempotent	--	123	
	Negotiation-Response	ABC	--	T1	--	Recoverable	--	
Establish		ABC	T2	--	--	--	--	10
<One KeepAliveInterval has lapsed without any response>								
Establish		ABC	T3	--	--	--	--	10

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Server Flow	Credentials	KeepAliveInterval
<New Establish message should contain same Session ID>								

7.2 Unbinding

7.2.1 Ungraceful termination (time out)

When the KeepAliveInterval has expired and no keep alive message is received then the session is terminated ungracefully and will need to be re-established. The transport level connection is still open (TCP socket) therefore Negotiation is not required. Termination due to error does not require the sender to wait for corresponding Terminate response from counterparty.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Code	Reason
Negotiate		ABC	T1	--	Idempotent	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	--	--
Establish		ABC	T2	--	--	10	--	--
	EstablishmentAck	ABC	--	T2	--	10	--	--
<Time Interval Greater Than Keep Alive Interval Has Lapsed Without Any Message Being Received>								
	Terminate	ABC	--	--	--	--	Timed Out	Keep Alive Interval Has Lapsed
Establish		ABC	T3	--	--	10	--	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Code	Reason
	EstablishmentAck	ABC	--	T3	--	10	--	--
<New Establish message should be sent with same Session ID>								

7.2.2 Ungraceful termination (sequence message received with lower sequence number)

The session could also be deliberately terminated due to Sequence message received with lower than expected sequence number and then it will need to be re-established. The transport level connection is still open (TCP socket) therefore Negotiation is not required. Termination due to error does not require the sender to wait for corresponding Terminate response from counterparty.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--		Recoverable	--	--
Establish		ABC	T2	--	200	--	--	--	--	--
	EstablishmentAck	ABC	--	T2	1000	--	--	--	--	--
Sequence		--	--	--	100	--	--	--	--	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Invalid Next-SeqNo
Establish		ABC	T4	--	200	--	Idempotent	--	--	--
	EstablishmentAck	ABC	--	T4	1001	--	--	Recoverable	--	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
<New Establish message should be sent with same Session ID>										

7.2.3 Ungraceful termination (establishment ack received with lower sequence number)

The session could also be deliberately terminated due to EstablishmentAck message received with lower than expected sequence number and then it will need to be re-established. The transport level connection is still open (TCP socket) therefore Negotiation is not required. Termination due to error does not require the sender to wait for corresponding Terminate response from counterparty.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--		Recoverable	--	--
Establish		ABC	T2	--	200	--	--	--	--	--
	EstablishmentAck	ABC	--	T2	1000	--	--	--	--	--
Sequence		--	--	--	100	--	--	--	--	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Invalid Next-SeqNo
Establish		ABC	T4	--	200	--	Idempotent	--	--	--
	EstablishmentAck	ABC	--	T4	1001	--	--	Recoverable	--	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
<New Establish message could be sent with same Session ID>										

7.2.4 Graceful Termination

Graceful termination is possible when there are no more messages to be sent for the time being and the TCP socket connection could be disconnected for now. This allows the sender to re-establish connectivity with the same session ID as before since the session was terminated without finalization (FinishedSending was not used to indicate logical end of flow). Graceful termination (not due to error) does require the sender to wait for corresponding Terminate response from counterparty before disconnecting TCP socket connection. The receiver should not attempt to initiate TCP socket disconnection since the sender will do that upon receiving the response.

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--		Recoverable	--	--
Establish		ABC	T2	--	200	--	--	--	--	--
	EstablishmentAck	ABC	--	T2	1000	--	--	--	--	--
Sequence		--	--	--	201	--	--	--	--	--
Terminate		ABC	--	--	--	--	--	--	Finished	--
	Terminate	ABC	--	--	--	--	--	--	Finished	--

Message Received	Message Sent	Session ID	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
<TCP socket connection is disconnected by sender>										
Establish		ABC	T4	--	200	--	Idempotent	--	--	--
	EstablishmentAck	ABC	--	T4	1001	--	--	Recoverable	--	--
<New Establish message could be sent with same Session ID>										

7.2.5 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	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Server Flow	Reason
Negotiate		ABC	T1	--	Idempotent	--	--	--
	NegotiationResponse	ABC	--	T1	--	--	Recoverable	--
Establish		ABC	T2	--	--	10	--	--
	EstablishmentAck	ABC	--	T2	--	10	--	--
<TCP socket connection is disconnected>								
Negotiate		DEF	T3	--	Idempotent	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Client Flow	Keep Alive Interval	Server Flow	Reason
	NegotiationResponse	DEF	--	T3	--	--	Recoverable	--
Establish		DEF	T4	--	--	10	--	--
	EstablishmentAck	DEF	--	T4	--	10	--	--
<New Negotiate message requires new Session ID>								

7.3 Transferring

7.3.1 Sequence

Over TCP – a Client could 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	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Client Flow	Server Flow	Implicit SeqNo
Negotiate		ABC	T1	--	--	Idempotent		--
	Negotiation Response	ABC	--	T1	--		Recoverable	--
Establish		ABC	T2	--	100			--
	EstablishmentAck	ABC	--	T2	1000			--
Sequence		--	--	--	100			--
NewOrderSingle		ABC	T3	--	--	--	--	100
	ExecutionReport	ABC	T4	--	--	--	--	1000

Sequence message is applicable for idempotent and recoverable flows and if received for unsequenced and none flows then issue terminate message to sender since it is a protocol violation.

7.3.1.1 Sequence (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	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
Sequence		--	--	--	100	--	--	--	--	--
NewOrderSingle		ABC	T3	--	--	100	--	--	--	--
	Execution Report	ABC	T4	--	--	1000	--	--	--	--
Sequence		--	--	--	200	--	--	--	--	--
NewOrderSingle		ABC	T5	--	--	200	--	--	--	--
	NotApplied	--	--	--	--	--	--	--	101	100
	Execution Report	ABC	T6	--	--	1001	--	--	--	--

7.3.1.2 Sequence (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	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	Code	Reason
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment-Ack	ABC	--	T2	1000	--	--	--	--	--
Sequence		--	--	--	100	--	--	--	--	--
NewOrder-Single		ABC	T3	--	--	100	--	--	--	--
	Execution-Report	ABC	T4	--	--	1000	--	--	--	--
Sequence		--	--	--	50	--	--	--	--	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Invalid Next-SeqNo

7.3.1.3 Sequence (heartbeat)

The Sequence message could also be used as a heartbeat for idempotent and recoverable flows.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Client Flow	Server Flow	Keep Alive Interval	Implicit SeqNo
Negotiate		ABC	T1	--	--	Idempotent	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Client Flow	Server Flow	Keep Alive Interval	Implicit SeqNo
	Negotiation Response	ABC	--	T1	--		Recoverable	--	--
Establish		ABC	T2	--	100	--	--	10	--
	EstablishmentAck	ABC	--	T2	1000	--	--	10	--
Sequence		--	-- (T2+10)	--	100	--	--	--	--
	Sequence	--	-- (T2+11)	--	1000	--	--	--	--
Sequence		--	-- (T2+20)	--	100	--	--	--	--
	Sequence	--	-- (T2+21)	--	1000	--	--	--	--

7.3.2 Context (Multiplexing Session ID's)

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 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 value of NextSeqNo in each 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	Session ID (UUID)	Timestamp	Request Timestamp	Next Seq No	Implicit SeqNo
Negotiate		ABC	T1	--	--	--
	NegotiationResponse	ABC	--	T1	--	--
Establish		ABC	T2	--	--	--
	EstablishmentAck	ABC	--	T2	1000	--
Negotiate		DEF	T3	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next Seq No	Implicit SeqNo
	NegotiationResponse	DEF	--	T3	--	--
Establish		DEF	T4	--	--	--
	EstablishmentAck	DEF	--	T4	2000	--
Context		ABC	--	--	100	--
NewOrderSingle		ABC	T5	--	--	100
	Context	ABC	--	--	1000	--
	ExecutionReport	ABC	T6	--	--	1000
Context		DEF	--	--	200	--
NewOrderSingle		DEF	T7	--	--	200
	Context	DEF	--	--	2000	--
	ExecutionReport	DEF	T8	--	--	2000

7.3.2.1 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	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo
Negotiate		ABC	T1	--	--	--
	NegotiationResponse	ABC	--	T1	--	--
Establish		ABC	T2	--	--	--
	EstablishmentAck	ABC	--	T2	1000	--
Sequence		--	--	--	100	--
NewOrderSingle		ABC	T3	--	--	100
	ExecutionReport	ABC	T4	--	--	1000
Negotiate		DEF	T5	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo
	NegotiationResponse	DEF	--	T5	--	--
Establish		DEF	T6	--	--	--
	EstablishmentAck	DEF	--	T6	2000	--
Sequence		--	--	--	200	--
NewOrderSingle		DEF	T7	--	--	200
	ExecutionReport	DEF	T8	--	--	2000

7.3.3 Unsequenced Heartbeat

This message is used to keep alive the session on unsequenced and none flows.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Client Flow	Server Flow	Keep Alive Interval	Implicit SeqNo
Negotiate		ABC	T1	--	--	Unsequenced	--	--	--
	Negotiation Response	ABC	--	T1	--		Recoverable	--	--
Establish		ABC	T2	--	100	--	--	10	--
	Establishment-Ack	ABC	--	T2	1000	--	--	10	--
UnsequencedHeartbeat		--	-- (T2+10)	--	--	--	--	--	--
UnsequencedHeartbeat		--	-- (T2+20)	--	--	--	--	--	--
UnsequencedHeartbeat		--	-- (T2+30)	--	--	--	--	--	--

7.3.4 Retransmission Request

For recoverable flows, a Retransmission Request could be issued to recover gap in sequence numbers

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
	Execution Report	ABC	T3	--	--	1100	--	--	--	--
Retransmission-Request		ABC	T4	--	--	--	--	--	1000	100
	Retransmission	ABC	--	T4	1000	--	--	--	--	100
<100 messages between 1000 to 1099 are replayed and message number 1100 is queued for processing after Retransmisison is satisfied>										

Retransmission message is not applicable for idempotent, unsequenced and none flows and if received for these flows then issue terminate message to sender since it is a protocol violation.

7.3.4.1 Retransmission (Concurrent)

More than one RetransmissionRequest is not allowed at a time and subsequent in-flight requests will lead to session termination.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recover-able	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
	Execution Report	ABC	T3	--	--	1100	--	--	--	--
Retransmission-Request		ABC	T4	--	--	--	--	--	1000	100
	Retransmission	ABC	--	T4	1000	--	--	--	--	100
<50 messages between 1000 and 1099 are replayed>										
Retransmission-Request		ABC	T5	--	--	--	--	--	1050	50
	Terminate	ABC	--	--	--	--	--	--	--	--
<Session terminated with TerminationCode = ReRequestInProgress>										

7.3.4.2 Retransmission (Interleaving)

While responding back to a RetransmissionRequest it is possible that the sender could interleave real time original messages with duplicate retransmission responses. This interleaving will happen between both flows in ranges which could be the chunk of messages which can fit into a

single datagram or packet. Each batch of duplicate replayed messages will be preceded by a Retransmission message in the same packet and each batch of real time original messages will be preceded by a Sequence message in the same packet.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	From SeqNo	Count
RetransmissionRequest		ABC	T1	--	--	--	--	--	1000	100
	Retransmission	ABC	--	T1	1000	--	--	--	--	50
<50 duplicate messages between 1000 and 1049 are replayed in first packet which includes Retransmission message> <Real time messages between 2000 and 2049 are queued by the sender at this time>										
	Sequence				2000					
<50 original real time messages between 2000 and 2049 are sent in second packet which includes Sequence message> <Duplicate messages between 1050 and 1099 are queued by sender at this time>										
	Retransmission	ABC	--	T1	1050					50

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Client Flow	Server Flow	From SeqNo	Count
<Second batch of 50 duplicate messages between 1050 and 1099 are send in third packet which includes Retransmission message>										

7.3.5 Retransmission Reject

7.3.5.1 Invalid FromSeqNo

RetransmissionRequest could be rejected if the messages being requested (FromSeqNo) belong to an invalid sequence range i.e. greater than last sent sequence number from sender.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
Retransmission-Request		ABC	T3	--	--	--	--	--	2000	100
	Retransmit-Reject	ABC	--	T3	--	--	OutOfRange	Invalid FromSeqNo	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
<Here FromSeqNo is greater than last value of NextSeqNo from sender>										

7.3.5.2 Retransmission Reject (OutOfRange)

RetransmissionRequest could be rejected if the messages being requested (FromSeqNo + Count) belong to an invalid sequence range i.e. greater than last sent sequence number from sender.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
Retransmission-Request		ABC	T3	--	--	--	--	--	900	175
	Retransmit-Reject	ABC	--	T3	--	--	OutOfRange	Invalid Range	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
<Here FromSeqNo + Count is greater than last value of NextSeqNo from sender>										

7.3.5.3 Retransmission Reject (Invalid Session ID)

RetransmissionRequest could be rejected if the messages are being requested with a different session ID such that it is unknown or not authorized.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
Retransmission-Request		DEF	T3	--	--	--	--	--	850	50
	Retransmit-Reject	DEF	--	T3	--	--	Invalid Session	Unknown Session ID	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
<Here DEF is an unknown session ID since it has not negotiated and established a session>										

7.3.5.4 Retransmission Reject (Request Limit Exceeded)

RetransmissionRequest could be rejected if the messages being requested exceed the limit for allowable count in each request.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	100	--	--	--	--	--
	Establishment Ack	ABC	--	T2	1000	--	--	--	--	--
	Sequence	--	--	--	1000	--	--	--	--	--
Retransmission-Request		ABC	T3	--	--	--	--	--	1	999
	Retransmit-Reject	ABC	--	T3	--	--	RequestLimit-Exceeded	Count Exceeds 500	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	Code	Reason	From SeqNo	Count
<Here the Retransmisison-Request was rejected due to the count of messages requested bring greater than what is supported by the sender>										

7.3.5.5 Retransmission Reject (Retrasmission Out of Bounds)

RetransmissionRequest asking to replay messages which are no longer available (for example older than three days) could also be rejected.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	From SeqNo	Count	Code	Reason
Negotiate		ABC	T1	--	--	--	--	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	--	--	--
Establish		ABC	T2	--	200	--	--	--	--	--
	Establishment-Ack	ABC	--	T2	1000	--	--	--	--	--
Retransmit-Request		ABC	T3	--	--	--	1	175	--	--
	Retransmit-Reject	ABC	--	T3	--	--	--	--	ReRequestOutOf-Bounds	Messages No Longer Available

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Implicit SeqNo	From SeqNo	Count	Code	Reason
<Here the messages being requested (1-175) were older than 72 hours>										

7.4 Finalizing

7.4.1 Finished Sending & Finished Receiving

The FinishedSending message is used by the initiator to inform the acceptor that the logical flow of messages has reached its end i.e. the FIXP session is in the process of being wound down and gracefully terminated, for example at the end of the day or at the end of the week etc. Once the acceptor responds back with a FinishedReceiving confirmation message then the session could be half-closed i.e. no more messages will be sent from the initiator but the acceptor could continue to send messages until it does not send a FinishedSending message itself to the counterparty to indicate that it too has reached the end of its logical flow and it has no more messages to send.

The FinishedReceiving message is used to confirm that the FinishedSending message has been successfully received and acknowledged and that the FIXP session could be terminated once both counterparties have sent and received a FinishedReceiving message. The initiator is then expected to re-negotiate the session with a new SessionID.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
Negotiate		ABC	T1	--	--	--	Idempotent	--	--	--
	Negotiation Response	ABC	--	T1	--	--	--	Recoverable	--	--
Establish		ABC	T2	--	200	--	---	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
	Establishment-Ack	ABC	--	T2	1000	--	--	--	--	--
NewOrderSingle		ABC	T3	--	--	--	--	--	--	--
	Eecution-Report	ABC	--	T3	--	--	--	--	--	--
FinishedSending		ABC	--	--	--	201	--	--	--	--
	Finished-Receiving	ABC	--	--	--	--	--	--	--	--
	Finished-Sending	ABC	--	--	--	1001	--	--	--	--
FinishedReceiving		ABC	--	--	--	--	--	--	--	--
Terminate		ABC	--	--	--	--	--	--	Finished	--
	Terminate	ABC	--	--	--	--	--	--	Finished	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
Here the initiator has sent the Terminate message but either counterparty could have sent it since both have sent and received a Finished-Receiving message. The TCP socket connection is disconnected by the initiator upon receipt of the corresponding Terminate ack.										
Negotiate		DEF	T4	--	--	--	--	--	--	--
	Negotiation-Response	DEF	--	T4	--	--	--	--	--	--

7.4.2 Finished Sending & No Response Received

If the initiator has sent a FinishedSending message and does not receive a corresponding FinishedReceiving response from the counterparty within one KeepAliveInterval then it is supposed to keep sending the FinishedSending message until it hears back at the rate of one per KeepAliveInterval i.e. use it as a proxy for the Heartbeat message.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	Code	Reason
FinishedSending		ABC	--	--	--	201	--	--
One <KeepAliveInterval> lapses without any corresponding FinishedReceived response from the counterparty								
FinishedSending		ABC	--	--	--	201	--	--
One <KeepAliveInterval> lapses without any corresponding FinishedReceived response from the counterparty								
FinishedSending		ABC	--	--	--	201	--	--
	Finished-Receiving	ABC	--	--	--	--		
Even though multiple <FinishedSending> messages have been sent, a single <FinishedReceiving> response is sufficient to assume that the session could be terminated i.e. there is no need to wait for separate <FinishedReceving> responses for each <FinishedSending> request sent and the initiator could now terminate the session								

7.4.3 Finished Sending & Recoverable Flow

Upon receiving the FinishedSending message, if the counterparty detects a gap in the sequence by scrutinizing the <LastSeqNo> field (which is literal and not implied) then it will attempt to recover this gap in a recoverable flow before responding back with a corresponding FinishedReceiving confirmation message.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	FromSeqNo	Count	Code
FinishedSending		ABC	--	--	--	201	--	--	--
Last implicit sequence number or value of <NextSeqNo> from ABC is 198 therefore acceptor issues a <Retransmission-Request> to recover sequence gap of four messages (198-201) assuming that ABC was using recoverable flow									
	Retransmission-Request	ABC	T1	--	--	--	198	4	--
Retransmit		ABC	--	T1	198	--	--	4	--
Initiator replays messages in requested sequence range between 198-201 and acceptor processes these messages and responds back with corresponding acknowledgements. The initiator should be ready to process these acknowledgements from acceptor in response to retransmission even after sending a <FinishedSending> message									
	Finished-Receiving	ABC	--	--	--	--	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	FromSeqNo	Count	Code
Since the acceptor's retransmission request has been satisfied, it then proceeds to reply back with a <Finished-Receiving> message so that the initiator's logical flow of messages could cease.									

7.4.4 Finished Sending & Termination

The party which wishes to cease logical flow of messages from its connection at the end of the day, end of the week or upon market close should wait until the other counterparty is also ready to do the same before attempting to terminate the connection otherwise this will be regarded as a protocol violation and will result in an ungraceful termination of the connection by the other party which has not yet had the opportunity to cease logical flow of its own messages.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	FromSeqNo	Count	Code	Reason
Finished-Sending		ABC	--	--	--	201	--	--	--	--
	Finished-Receiving	ABC	--	--	--	--	--	--	--	--
Terminate		ABC	--	--	--	--	--	--	Finished	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Logical Flow Interrupted

7.4.5 Finished Sending & Further Message Flow

The party which wishes to cease logical flow of messages from its connection at the end of the day, end of the week or upon market close should not send any other message after the first FinishedSending message has been sent. The only exception to this rule is the Retransmission

message and replayed messages (in response to RetransmissionRequest from counterparty if it detects a gap based on the value of LastSeqNo). If it sends any other message either (FIXP or application) then it will lead to ungraceful termination by the other counterparty since this is a protocol violation. This should be avoided at all costs since if the other counterparty is trying to recover a gap in sequence then that will be aborted.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	FromSeqNo	Count	Code	Reason
FinishedSending		ABC	--	--	--	201	--	--	--	--
	Finished-Receiving	ABC	--	--	--	--	--	--	--	--
Sequence		ABC	--	--	202	--	--	--	--	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Logical Flow Cannot Resume After Finalization
Here a Sequence message was sent after the counterparty responded back with a Finished Receiving message and it led to an ungraceful termination										
FinishedSending		ABC	--	--	--	201	--	--	--	--
Sequence		ABC	--	--	202	--	--	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	LastSeqNo	FromSeqNo	Count	Code	Reason
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	Logical Flow Cannot Resume After Finalization
Here a Sequence message was sent before the counterparty could respond back with a Finished Receiving message and it too led to an ungraceful termination										

7.4.6 Finished Sending & Half-Close

Once one of the two parties has ceased logical flow of messages from its connection at the end of the day, end of the week or upon market close then it should still be ready and able to accept messages from the other counterparty till the time that the counterparty itself does not cease logical flow of messages from its own connection. However this should not lead to any corresponding output back from the connection which has been half-closed (with the exception of Retransmission) since that would be a protocol violation and lead to ungraceful termination.

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
FinishedSending		ABC	--	--	--	201	--	--	--	--

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
	Finished-Receiving	ABC	--	--	--	--	--	--	--	--
	Eecution-Report	ABC	--	T5	--	--	--	--	--	--
	Eecution-Report	ABC	--	T6	--	--	--	--	--	--
Retransmission-Request		ABC	T7	--	--	--	--	--	--	--
	Terminate	ABC	--	--	--	--	--	--	Unspecified Error	-- Logical Flow Cannot Resume After Finalization

Message Received	Message Sent	Session ID (UUID)	Timestamp	Request Timestamp	Next SeqNo	Last SeqNo	ClientFlow	ServerFlow	Code	Reason
Here the initiator has sent a Retransmission-Request message after ceasing logical flow of messages from its own connection while continuing to accept message flow from acceptor and this will result in an ungraceful termination since the initiator can only respond back to a Retransmisison-Request but cannot initiate one of its own after half-closing its connection.										

8 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		
Framing		
Session Layer		
Supported flow types		
Security protocols		
Authentication		
Transport Layer		
Transports supported		
Other network protocols		

