FASTMATCH

Accelerating Forex Trading to the Next Level
- Superior Quality Order Flow
- Fast Technology
- Innovative Functionality
- Transparency

ITCH PROTOCOL SPECIFICATION DOCUMENT

MARKET DATA

REVISION HISTORY

Version	Last Updated	Updates
1.0	June 23, 2015	Initial Version
1.1	July 14, 2015	Changes in condition in trade message as following: A = Added vs AutoEx; B = Added vs Stream; R = Removed vs AutoEx ; S = Removed vs Stream
1.1.1	August 14, 2015	Added description of when midpoint value is zero
1.2	September 16, 2015	Multiple editorial changes.
1.2.1	October 26, 2015	Clarification on UDP speed advantages.

1. INTRODUCTION

1.1 PURPOSE

This document is provided as a guide for FASTMATCH clients, as to how the FASTMATCH's ITCH Protocol (FMITCH) may be used to establish connectivity with FASTMATCH for the purposes of receiving Fastmatch market data.

ITCH is a simplified protocol that allows FASTMATCH participants to subscribe and receive market data either over TCP/IP or over UDP/IP protocols.

All counterparties will need to certify their trading system with FASTMATCH in the User Acceptance Testing ("UAT") environment before being called production ready.

1.2 CONTENT

Included in this document are the following:

- General definitions and specifications for clients using ITCH to subscribe and receive market data
- ITCH message formats to be used and details of their expected parameters.

2. OVERVIEW

2.1 SCOPE

The ITCH Interface allows clients to subscribe to market data streams and receive quote, trade, and midpoint updates, as well as MarketOnClose price updates.

3. CONNECTIVITY

3.1 FASTMATCH MATCHING ENGINE LOCATIONS

FASTMATCH matching engine is located in Equinix NY4, LD4 and TY3 Data Centers:

- NY4, 755 Secaucus Road, Secaucus, NJ 07094
- LD4, 2 Buckingham Avenue, Slough, Berkshire, SL1 4NB
- TY3, 1-9-20 Edagawa Koto-Ku Tokyo 135-0051

3.2 CONNECTIVITY OPTIONS

- Clients have a choice of establishing cross-connect and internet connectivity to FASTMATCH NY4, LD4 and TY3 locations.
- Local cross-connect to FASTMATCH ECN cages in NY4, LD4 and TY3 data centers could be used for both Production and UAT access.
- Metro connections from other data centers are accepted.
- Market Data traffic will be either TCP based with unique target (IP:port) for each ITCH session, or UDP based for clients requiring ultimate latency.
- No multicast traffic will travel via client connectivity.
- Market Data dissemination over UDP can only be used by cross-connected clients.

- To start UAT certification process, client can establish the Internet connectivity to FASTMATCH.

3.3 CROSS-CONNECTIVITY

- All client cross-connect connectivity is 1Gpbs Multimode or Single-mode fiber.
- FASTMATCH will issue a client a LOA to connect to FASTMATCH ECN with up to two fiber cross-connects.
- To avoid delays, we ask a client to confirm the correct firm or third-party agent name to be used in LOA.
- If two cross-connects are ordered, then they will be connected the different access switches for redundancy.
- BGP is preferred choice even on a single cross-connect connection for support purposes. Static routing is accepted if client hardware cannot support BGP.
- FASTMATCH will advertise registered IP address space from a registered BGP ASN.
- Cross-connect will be addressed using RFC 1918 address space (preferred). Registered IP space could be used to avoid the IP address conflict.
- FASTMATCH will accept client’s registered IP Address and BGP ASN. If required, FASTMATCH will assign client’s server farm IP addresses and BGP ASN.

3.4 INTERNET CONNECTIVITY

Internet connectivity is available at NY4, LD4 and TY3 locations. There are two internet providers at every location. They may be utilized as main and failover connections. The main connection can be selected based on roundtrip statistics.

3.5 SYSTEM/SERVICE AVAILABILITY

3.5.1 TRADING SESSION AND STP SESSION:

3.5.1.1 SESSION AVAILABILITY:

Market hours for trading are the same as for streaming: the market opens on Sunday at 5:30 PM NY time and closes on Friday at 5:00 PM NY time. The ITCH market data service is off-line from 17:00:00 EST/EDT until 17:30:00 EST/EDT, Daily Monday through Thursday. During this time FASTMATCH resets the inbound and outbound sequence numbers on all ITCH trade sessions.

Clients can reconnect after 17:30:00 EST/EDT. Typically, when initially logging into a server the client will set the Requested Sequence Number field to 1 and leave the Requested Session field blank in the Login Request Packet

3.6 CONNECTING INTRADAY

Clients connecting intra-day cannot request any retransmissions of market data messages. In the future Fastmatch may add retransmission of trade messages.

3.7 PROTOCOL VERSIONS

The only supported version currently is 1.

3.8 TCP PROTOCOL

Fastmatch ITCH specification uses TCP or UDP protocol for market data distribution. Similarly to OUCH, TCP distribution utilizes SoupBinTCP protocol on top of TCP for session control.

SoupBinTCP is a lightweight point-to-point protocol, built on top of TCP/IP sockets that allow delivery of a set of sequenced messages from a server to a client in real-time. SoupBinTCP guarantees that the client receives each message generated by the server in sequence, even across underlying TCP/IP socket connection failures.

SoupBinTCP is ideal for systems where a server needs to deliver a logical stream of sequenced messages to a client in real-time but does not require the same level of guarantees for client generated messages either because the data stream is unidirectional or because the server application generates higher-level sequenced acknowledgments for any important client-generated messages.

SoupBinTCP is designed to be used in conjunction with higher lever protocols that specify the contents of the messages that SoupBinTCP messages deliver. The SoupBinTCP protocol layer is opaque to the higher-level messages. Note that unlike the ASCII version, messages may include any possible byte.

SoupBinTCP also includes a simple scheme that allows the server to authenticate the client on login.

Fastmatch employs SoupBinTCP in accordance with SoupBinTCP, Version 3.0 specification with **the two exceptions**:

By default, all integers are sent and received in little-endian format for performance reason.

There is an extra field in Login Request Packet message indicating version number. It is equal 1 for the current version.

This protocol deviation helps to circumvent redundant conversion from little-endian to network order (big-endian) and back, assuming prevalence of Linux/Windows x64-processor based systems. Yet, upon a client's request SoupBinTCP/FMITCH may be configured to exchange data in network byte order (big-endian) endianness.

3.9 UDP PROTOCOL

Clients can take advantage of market data dissemination over UDP, over what we called SoupBinUDP. Fastmatch may send out multiple book updates in each datagram. Trade/Midpoint/MarketOnClose updates will always be sent one per datagram. Size of data in each datagram will not exceed MTU (full datagram not exceeding 1500 bytes). SoupBinUDP message cannot span across multiple UDP datagrams. **Note that only cross-connected clients can take advantage of UDP market data dissemination**. Internet clients can only utilize TCP. Each datagram will start with SoupBinUDP datagram header, followed by ITCH header.

SoupBinUDP protocol also utilizes same session-level Logon/Logout/Heartbeat/EndOfSession messages. Upon startup client can send one or multiple login requests. Login requests can be sent from multiple permissioned destinations. If a specified IP is not permissioned, login will be rejected. In LoginRequest is accepted, Fastmatch will sent LoginAccept message, containing SessionID. Client can subsequently subscribe for market data using provided SessionID. Fastmatch will send market data to the destination/IP from which login was received and accepted. Support for multiple logins allows clients to partition the traffic and forward it to multiple market data processing instances. Clients can partition traffic any way they choose. For example a client may split currency pairs into several logical groups and process them in multiple instances, or a client may wish to process trades, midpoint, or MarketOnClose updates by separate instances, then those processing quotes. This can also be used to forward drop copies to a separate processing destination. The maximum number of allowed logins is configurable per client on

Fastmatch side (default is 1). Clients are required to send heartbeats on each active session. Each LoginRequest also contains version.

There are multiple advantages of using UDP vs TCP for market data. TCP is susceptible to retransmissions. The retransmissions may introduce significant spiked delays. Additionally, TCP may detect network congestion and slow down message transmissions. Since UDP has no error-checking it allows clients to implement their own data flow control with performance in mind.

There are also infrastructural benefits of using UDP as datagrams can be converted on the client's switch (e.g. Arista switch) into multicast and the switch will forward that market data into multiple listening instances. With TCP this is not possible, since only a single process/instance can open a connection and receive full market data.

In a network error and congestion free environment, UDP is just slightly faster than TCP and under such conditions Fastmatch UDP Market Data would only several hundred nanoseconds faster than TCP.

3.10 SUBSCRIPTION REQUESTS

Client is required to send SubscriptionRequest messages in order to initiate market data stream. In response Fastmatch sends SubscriptionResponse message. In case subscription request is accepted, Fastmatch will specify InstrumentID in subscription response. The instrumentID will be used to identify book/trade/midpoint/MarketOnClose updates. Additionally, clients can send InstrumentListRequest to get list of all supported instruments with their respective instrument IDs and settlement dates. Client can send multiple subscription requests for the same symbol but different subscription entry types (book/trades/midpoint). SubscriptionResponse should be correlated with SubscriptionRequest using SubsRequestID, which must be at least unique per symbol.

Note that subscription request for all symbols is not supported. Each symbol has to be requested explicitly with a separate subscription request.

3.11 LOGICAL STREAMS

Both TCP and UDP support StreamID (located in ITCH header). StreamID is similar to tag 109 in Fastmatch FIX spec. It allows clients to receive different price streams over the same session. StreamID is a logical identifier of the stream. The allowed mappings must be configured and permissioned on Fastmatch side.

3.12 SUPPORTED MESSAGE SET

The supported message set is as follows:

3.12.1.1 SESSION LEVEL MESSAGES

Session-level messages are the same for TCP and UDP and are identical to session-level messages used in OUCH spec.

Direction	Message	Type	Description
In	**Login Request Packet**	L	The client must send a Login Request Packet immediately upon establishing a new TCP/IP socket connection to the server or to login into UDP session. Client and server must have mutually agreed upon the username and password fields. They provide simple authentication to prevent a client

			from inadvertently connecting to the wrong server/session. Both Username and Password are case-insensitive and should be padded on the right with spaces. The server can terminate an incoming TCP/IP socket if it does not receive a Login Request Packet within a reasonable period of time (typically 30 seconds).
Out	**Login Accept Packet**	A	Fastmatch server sends a Login Accepted Packet in response to receiving a valid Login Request from the client. This packet will always be the first sent by the server after a successful login request.
Out	**Login Reject Packet**	J	Fastmatch server sends this packet in response to an invalid Login Request Packet from the client. The server closes the socket connection after sending the Login Reject Packet. The Login Rejected Packet will be the only packet sent by the server in the case of an unsuccessful login attempt.
Out	**Sequenced Data Packet**	S	The Sequenced Data Packets act as an envelope to carry the actual sequenced data messages that are transferred from the server to the client. Each Sequenced Data Packet carries one message from the higher-lever protocol. The sequence number of each message is implied; the initial sequence number of the first Sequenced Data Packet for a given TCP/IP connection is specified in the Login Accepted Packet and the sequence number increments by 1 for each Sequenced Data Packet transmitted. Since logical packets are carried via TCP/IP sockets, the only way logical packets can be lost is in the event of a TCP/IP socket connection failure. In this case, the client can reconnect to the server and request the next expect sequence number and pick up where it left off.
Out	**Server Heartbeat**	H	The Fastmatch server should send a Server Heartbeat Packet anytime more than 1 second passes where no data has been sent to the client. The client can then assume that the link is lost if it does not receive anything for an extended period of time.
Out	**End of Session**	Z	The server will send an End of Session Packet to denote that the current session is finished. The connection will be closed shortly after this packet, and the user will no longer be able to reconnect to the current session.

In	**Unsequenced Data Packets**	U	The Unsequenced Data Packets act as an envelope to carry the actual data messages that are transferred from the client to the server. These messages are not sequenced and may be lost in the event of a socket failure. The higher-level protocol must be able to handle these lost messages in the case of a TCP/IP socket connection failure.
In	**Client Heartbeat**	R	The client should send a Client Heartbeat Packet anytime more than 1 second passes where no data has been sent to the server. The server can then assume that the link is lost if it does not receive anything for an extended period of time.
In	**Client Logout Request**	O	The client may send a Logout Request Packet to request the connection be terminated. Upon receiving a Logout Request Packet, the server will immediately terminate the connection and close the associated TCP/IP or UDP/IP socket.

3.12.1.2 BUSINESS MESSAGES

Direction	**Message**	**Type**	**Purpose**
In	**SubscriptionRequest**	S	Request to subscribe to a symbol
Out	**SubscriptionResponse**	R	Request to unsubscribe from a symbol
In	**InstrumentListRequest**	N	Request to receive list of instrument
Out	**InstrumentInfo**	F	Instrument description supported by Fastmatch
Out	**BookUpdate**	B	Book Update, followed by PriceAdd and PriceCancel updates.
Out	**PriceAdd**	P	New Price Update
Out	**PriceCancel**	C	Cancelling existing price on a book
Out	**TradeUpdate**	T	New Trade Update
Out	**MidpointUpdate**	M	Midpoint Update
Out	**OnCloseUpdate**	O	MarketOnClose Price Update

Out	**Reject**	J	Reject Message that can be sent in response any message in case server considers message invalid. See Appendix for supported Reject Error codes.

4. IMPLEMENTATION

4.1 MESSAGE ENCRYPTION

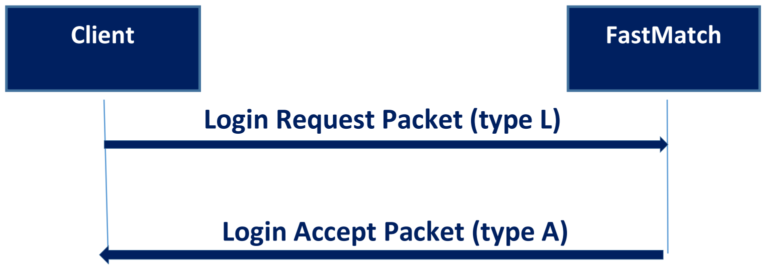
Encryption of binary messages themselves is not supported.

4.2 ESTABLISH CONNECTION / DISCONNECTION FOR TCP

4.2.1.1 CONNECT

Connection to the system is initiated by the client issuing a Login Request Packet.

If the connection can be accepted, then FASTMATCH will reply with Login Accept Packet, otherwise Login Reject Packet will be sent.



4.2.1.2 DISCONNECT

Closing of a connection is initiated by sending a Logout packet to the opposite party. At the end of trading day server sends End of Session message, followed by Logout. Additionally, client and server exchange heartbeats. If server misses at least 2 heartbeats from the client, the client is disconnected. Client later will have to re-subscribe to resume market data updates.

Client is expected to send unsubscribe messages if he no longer wishes to receive market data in specified symbol/type. If client sends Logout Request all subscription (that belong to specified SessionID) are cancelled.

4.3 ITCH FIELD TYPES

4.3.1.1 BYTE FIELD

8bit Signed Integer number (character interpreted as integer as per ASCII table)

4.3.1.2 SHORT FIELD

16bit Signed Integer number

4.3.1.3 INTEGER FIELD

32bit Signed Integer number

4.3.1.4 LONG FIELD

64bit Signed Integer Number

4.3.1.5 QUANTITY FIELD

Amounts are using 64bit Long numbers in little-endian format (unless client requests all integers to be in network order). Long value is scaled 2 decimal places to the right (multiplied by 100). For example, amount: 10000.12 should be sent as long number: 1000012. If quantity has no fractional part, it still has to be multiplied by 100. For example: 10000 should be sent as 1000000.

4.3.1.6 RATE FIELD

Rates are using 32bit Integer number. Integer value is scaled 5 decimal places to the right (multiplied by 100000 (100K)). For example 1.2345 should be sent as long number: 123450. 1200.01 should be sent as 120001000.

4.3.1.7 TIMESTAMP FIELD

In FMITCH header, timestamp is a 64bit Integer number, containing number of microseconds from Jan 1, 1970 GMT.

In FMITCH messages, timestamp field is a 64bit long containing microseconds from Jan 1, 1970 GMT. Example of such field is TransactTime.

4.3.1.8 DATE FIELD

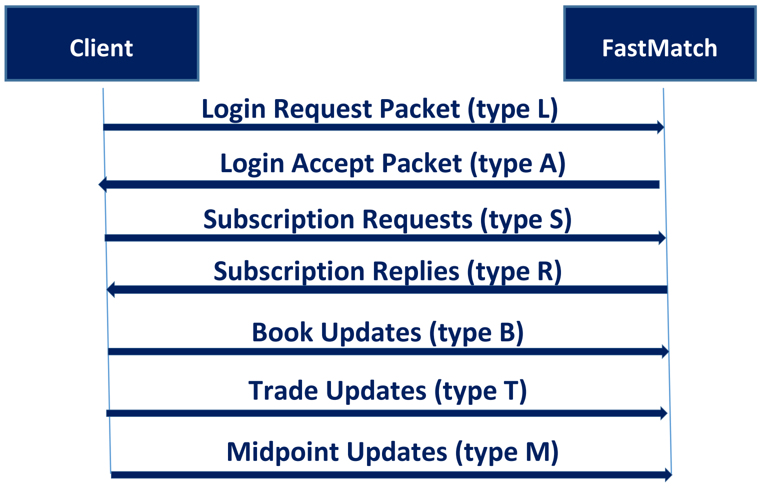
Date is sent as 32bit Integer value containing seconds from Jan 1, 1970 GMT.

4.3.1.9 ALPHANUMERIC FIELD

Contains alphanumeric characters.

4.4 MARKET DATA

The following diagram describes steps required to receive market data. Both TCP and UDP use the same steps.


Client
FastMatch
Login Request Packet (type L)
Login Accept Packet (type A)
Subscription Requests (type S)
Subscription Replies (type R)
Book Updates (type B)
Trade Updates (type T)
Midpoint Updates (type M)

4.5 SEQUENCE NUMBERS

SoupBinTCP packets use implicit sequence numbers. Both parties have to increment their relevant sequence numbers. TCP protocol handles packet loss/recovery and sequence number gap should never occur on the client side. In case of unrecoverable gap, session should be closed and reconnected.

UDP datagrams include sequence numbers in the datagram header. Additionally sequence number is included in each BookUpdate message. If client detects sequence gap, client can clear the corresponding book and start building it again, applying new price updates and ignoring price cancel messages for non-existent update IDs. Note that alternatively, client can clear all books as soon as gap is detected in datagram header sequence number. Client can choose the behavior that better fits their needs.

4.6 SUPPORTED MARKET DATA ENTRY UPDATE TYPES

The following entry update types are supported:

- '1' = Book
- '2' = Trade
- '3' = Midpoint
- '4' = MarketOnClose

4.7 SUPPORTED SUBSCRIPTION TYPES

The following subscription types are supported (only applicable to book updates):

- '1' = Aggregated
- '2' = Non Aggregated

4.8 SUPPORTED SUBSCRIPTION ACTION TYPES

The following subscription types are supported:

- '1' = Subscribe
- '2' = Unsubscribe

4.9 SUPPORTED BOOK DEPTH

The following subscription types are supported:

- 0 = Full Book (will be limited to the maximum number of levels configured for the client)
- 1 = Top of the book
- N = Number of levels (will be limited to the maximum number of levels configured for the client)

4.10 SUBSCRIPTION REQUEST ID

Each subscription must provide request ID unique at least per instrument symbol. For example client can send 2 subscription requests: one for Book updates in EUR/USD, the other for Trade updates in EUR/USD.

These 2 should have different subscription IDs, so that client can correlate subscription responses with original subscription requests using Request ID and Instrument Symbol. In order to correlate only by RequestID, client should make sure each subscription has a unique request ID.

4.11 INSTRUMENT SYMBOL AND INSTRUMENT ID

All updates (book/trade/midpoint/MarketOnClose) are sent using 2 bytes instrument ID instead of instrument symbol string. Client receives Instrument IDs in Subscription Responses. Alternatively, client can submit InstrumentInfoRequest and receive all Instrument descriptions, including InstrumentID, Settlement Date, and other relevant information. But this step is not required for market data subscription. Note that in case client utilizes UDP, server can send multiple InstrumentInfo updates in a single datagram, not to exceeding MTU.

Although InstrumentID is of type Short, spot currencies (and metals) are guaranteed to have InstrumentID under 256. Clients can take advantage of that and utilize direct access table to quickly look up their relevant books/instruments by InstrumentID. Non-Spot instruments may have higher InstrumentID numbers.

Also note that Instrument IDs are the same throughout the entire trading day. Instrument IDs are not guaranteed to be the same on the next trading day.

5. SESSION MESSAGE DEFINITIONS

Each message starts with a Session Level Header, either SoupBinTCP or SoupBinUDP. The rest of the message are similar for both transports.

5.1 LOGICAL SOUPBINTCP PACKET STRUCTURE

The SoupBinTCP client and server communicate by exchanging a series of logical packets. Each SoupBinTCP logical packet has:

Header:

A. a two byte little-endian length that indicates the length of rest of the packet (meaning the length of the payload plus the length of the packet type – which is 1)
B. a single byte header which indicates the packet type

Body:

C. a variable length payload

SoupBinTCP Logical Packet Structure

SoupBinTCP Header		Body
Two Byte Packet Length	Packet Type	Variable-length payload

For all cases when field length is one byte a corresponding ASCII numeric value of the character can be used. E.g. a printable value ‘L’ can be represented as one byte decimal 76

5.2 LOGICAL SOUPBINUDP DATAGRAM STRUCTURE

The SoupBinUDP client and server communicate by exchanging a series of logical datagrams. Each SoupBinUDP datagrams has:

Header:

A. Four Byte Sequence Number
B. a two byte little-endian length that indicates the length of rest of the datagram (meaning the length of the payload plus the length of the packet type – which is 1)
C. a single byte header which indicates the logical packet type

Body:

D. a variable length payload

UDP Logical Structure

SoupBinUDP Header			Body
Four Byte Sequence Number	Two Byte Packet Length	Packet Type	Variable-length payload

Deserialization of SoupBinUDP and SoupBinTCP messages is similar if a parser starts from the 5th bytes from the beginning of SoupBinUDP message.

6. SESSION MESSAGE DEFINITIONS

6.1 LOGIN REQUEST PACKET

PacketType = ‘L’

Session Level Header				
Name	**Offset**	**Length**	**Value**	**Description**
Version	0	2	Integer	= 1 for current version
Username	2	6	Alpha	Username
Password	8	10	Alpha	Password
SessionID	18	10	Alphanumeric	Specifies session that client wants to log into or all blanks to log into current session
NextSeqNum	28	20	Numeric	Next sequence number in ASCII that client wishes to receive or 0 to start receiving most recent message.

6.2 LOGIN ACCEPT PACKET

PacketType = 'A'

Fastmatch will send out Login Accepted Packet in response to Login Request from the client.

Session Level Header				
Name	**Offset**	**Length**	**Value**	**Description**
SessionID	0	10	Alphanumeric	SessionID assigned
SequenceNum	10	20	Numeric	Next sequence number to be sent in ASCII

6.3 LOGIN REJECT PACKET

PacketType = 'J'

Session Level Header				
Name	**Offset**	**Length**	**Value**	**Description**
Reject Reason	0	1	Byte	Reject codes: 'A' - Not authorized; 'S' - Session not available 'V' – Invalid Version

6.4 SEQUENCED DATA PACKET

PacketType = 'S

Session Level Header				
Name	**Offset**	**Length**	**Value**	**Description**
Message	0	Variable	Any	Outgoing Packet

6.5 SERVER HEARTBEAT

PacketType = 'H'

Session Level Header
No Packet Data

6.6 END OF SESSION

PacketType = 'Z

Session Level Header
No Packet Data

6.7 UNSEQUENCED DATA PACKETS

PacketType = 'U'

Session Level Header				
Name	**Offset**	**Length**	**Value**	**Description**
Message	0	Variable	Alphanumeric	

6.8 CLIENT HEARTBEAT

PacketType = 'R'

Session Level Header
No Packet Data

6.9 CLIENT LOGOUT

PacketType = 'O'

Session Level Header
No Packet Data

7. BUSINESS MESSAGE DEFINITIONS

Business message has the following logical structure:

Session Level Header	FM ITCH Header	Message Blocks

7.1 FM ITCH HEADER

Payload of each message will always start with ITCH header (10 bytes), containing timestamp, StreamID to which all subsequent updates belong, as well as MsgBlockCount with number of message blocks following this header. Each message block has a business message type.

Field	Offset	Length	Type	Description
Timestamp	0	8	Long	Time in microseconds from Epoch (Jan 1, 1970)

StreamID	8	1	Byte	The ID of logical stream to which messages in a block belong (for multi-stream trading only)
MsgBlockCount	9	1	Byte	Number of message blocks following this header

7.2 BOOK UPDATE

MessageType: ‘B’ (8 bytes)

All add price updates and cancel price updates will be preceded with BookUpdate. BookUpdate contains instrument ID, number of price updates in specified instrument, as well as sequence number of this book update. BookUpdate should be followed by ‘UpdateMsgCount ‘of PriceAdd and PriceCancel messages.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type ‘B’
InstrumentID	1	2	Short	Instrument ID of this book update
UpdateMsgCount	3	1	Byte	Number of updates in specified instrumentID, following this update header
SequenceNumber	4	4	Integer	Sequence number of update for this book update

7.3 PRICE ADD

MessageType: ‘P’ (28 bytes)

PriceAdd will always be sent as part of BookUpdate.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type ‘P’
PriceUpdateID	1	4	Integer	Price update unique per session per instrument
Quantity	5	8	Long	Quantity x100
MinQuantity	13	8	Long	Minimum Quantity x100
Rate	21	4	Integer	Rate x100000
Side	25	1	Byte	Side ‘1’ – Bid, ‘2’ - Offer
MaxDelay	26	2	Short	Max Delay for Quotes in milliseconds 0 - for orders

* Note that MaxDelay of 0 means price add is for a firm order

7.4 PRICE CANCEL

Message Type 'C' (5 bytes)

PriceCancel will always be sent as part of BookUpdate.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type 'C'
PriceUpdateID	1	4	Integer	Price update ID of a previously quoted and active price

7.5 TRADE UPDATE

MessageType: 'T' (24 bytes)

Note, each trade update is always sent in a separate packet/datagram. It will never be mixed with any other updates and is not included BookUpdate.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type 'T'
InstrumentID	1	2	Short	ID of the instrument assigned during subscription
Rate	3	4	Integer	Rate x100000
Quantity	7	8	Long	Quantity x100
Condition	15	1	Byte	'A' = Added vs AutoEx ; 'B' = Added vs Stream 'R' = Removed vs AutoEx ; 'S' = Removed vs Stream
TransactTime	16	8	Long	MatchTime in milliseconds since Jan 1, 1970

7.6 MIDPOINT UPDATE

MessageType: 'M' (7 bytes)

Note, each midpoint update is always sent in a separate packet/datagram. It will never be mixed with any other updates and is not included in BookUpdate.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**

Type	0	1	Byte	Type ‘M’
InstrumentID	1	2	Short	ID of the instrument assigned during subscription
Rate	3	4	Integer	New midpoint Rate x100000; if value of the Rate equals to zero, the midpoint should be considered invalid

7.7 MARKET ON CLOSE UPDATE

MessageType: ‘O’ (7 bytes)

Note, MarketOnClose update is always sent in a separate packet/datagram. It will never be mixed with any other updates and is not included in BookUpdate.

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type ‘O’
InstrumentID	1	2	Short	ID of the instrument assigned during subscription
Rate	3	4	Integer	Current Rate x100000

7.8 SUBSCRIPTION REQUEST

MessageType: ‘S’ (31 bytes)

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type ‘S’
InstrumentSymbol	1	12	Alphanumeric	String symbol for instrument (e.g. EUR/USD). Should be right padded with zeros.
UpdateType	13	1	Byte	‘1’ = Book; ‘2’ = Trade; ‘3’ = Midpoint ‘4’ = MarketOnClose
RequestID	14	4	Integer	Subscription Request ID
SessionID	18	10	Alphanumeric	Session ID assigned at Login
Action Type	28	1	Byte	’1’ – Subscribe, ‘2’ – Unsubscribe
SubscriptionType	29	1	Byte	‘1’ – Aggregated, ‘2’ – Non-Aggregated *
SubscriptionDepth	30	1	Byte	Subscription Depth (‘0’ – full book, ‘1’ – top level, ‘N’ – level count) *

*During unsubscribe request, subscription type and depth are ignored.

7.9 SUBSCRIPTION RESPONSE

MessageType: 'R' (31 bytes)

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type 'R'
InstrumentSymbol	1	12	Alpha	String symbol for instrument (e.g. EUR/USD). Should be right padded with zeros.
InstrumentID	13	2	Short	Instrument ID (immutable for entire trading day)
RequestID	15	4	Integer	Subscription Request ID
SessionID	19	10	Alphanumeric	Session ID assigned at Login
Status	29	1	Byte	'1' – Accepted, '2' – Rejected
ErrorCode	30	1	Byte	See list of reject codes in Appendix

7.10 INSTRUMENT LIST REQUEST

MessageType: 'N' (23 bytes)

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type 'N'
InstrumentSymbol	1	12	Alpha	String symbol for instrument (e.g. EUR/USD). Should be right padded with zeros. Set to "ALL" to receive all instruments.
SessionID	13	10	Alphanumeric	Session ID assigned at Login

7.11 INSTRUMENT INFO MESSAGE

MessageType: 'F' (20 bytes)

Session Level Header
FM ITCH Header

Field	Offset	Length	Type	Description
Type	0	1	Byte	Type ‘F’
InstrumentSymbol	1	12	Alpha	String symbol for instrument (e.g. EUR/USD). Should be right padded with zeros. Set to “ALL” to receive all instruments.
InstrumentID	13	2	Short	Instrument ID
InstrumentType	15	1	Byte	Type of Instrument
SettlementDate	16	4	Integer	Seconds in GMT from Jan 1, 1970

7.12 REJECT

MessageType: ‘J’ (24 bytes)

Session Level Header				
FM ITCH Header				
Field	**Offset**	**Length**	**Type**	**Description**
Type	0	1	Byte	Type ‘J’
MessageType	1	1	Byte	Type of rejected message
RejectCode	2	2	Short	Reject Code
RejectMessage	4	20	Alpha	Reject String

8. ERROR CODES

8.1 BUSINESS ERROR CODES

Error	Code	Description
NoError	‘0’	No Error
InvalidMessageType	‘1’	Invalid message type
InvalidMessageLength	‘2’	Invalid message length
InvalidInstrument	‘3’	Invalid Instrument
UnauthorizedDestination	‘4’	Destination not authorized (logged in)
OtherError	‘Z’	Any other error

8.2 SUBSCRIPTION REJECT REASON CODES

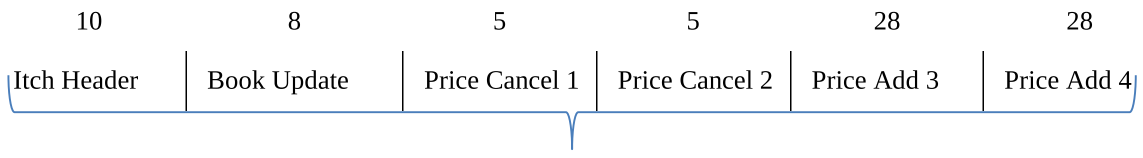
Error	Code	Description

NotAuthorized	A	Not authorized to subscribe
UnsupportedInstrument	S	Instrument not supported
UnsupportedVersion	V	Unsupported version number
DuplicateRequestID	D	Duplicate request ID
SubscriptionAlreadyActive	P	Subscription is already in progress
ExchangeNotOpen	E	Exchange not accepting subscriptions temporarily
InvalidActionType	F	Invalid subscription action
InvalidSubscriptionType	G	Invalid subscription type
InvalidDepth	H	Unsupported subscription depth
UnauthorizedSession	J	Subscription is not authorized

9. MESSAGE EXAMPLES

8.1 BOOK UPDATE MESSAGE EXAMPLE

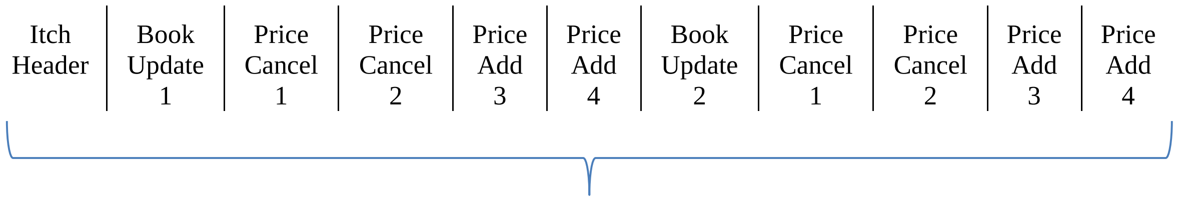
The following is an example of a sample book update with two quotes cancelled and two new quotes added:



Total: 84 bytes

The above update will take 84 bytes (excluding SoupBinTCP or SoupBinUDP header).

Example of similar book updates for two different currencies:



Total: 148 bytes

The above update will take 148 bytes (excluding SoupBinTCP or SoupBinUDP header).