Market Data Service (FIX/FAST 1.1)

Document version 1.1.0

01 November 2025

Revision history

Version 1.1.0 November 1, 2025

Updates to Tables 10 and 11 for Common Params.

Version 1.0.0 October 27, 2025

Created the document.

Table of Contents

List of Tables

1. Service for market data streaming (FIX/FAST 1.1)

The market data streaming service allows receiving market data in the process of exchange trading in FAST format.

The service uses data streaming in the FAST format incremental messaging.

Data streaming allows information to be transmitted from the source to the recipient without splitting it into separate messages for each event. One message can include a number of such events. It allows you to significantly reduce delays and increase data transfer rate.

The FAST protocol is a binary version of the FIX protocol (an international standard for information interaction and transactions in the foreign exchange, derivatives and stock markets), adapted for streaming market data. It helps to optimize the electronic exchange of financial information.

Using incremental messages can significantly reduce volumes of transmitted data. Ittakes only data that changed due to market events. They are refreshed by a minimum number of commands.

The FAST protocol is compatible with turnkey systems only. Otherwise, it is recommended to use a binary data transfer protocol.

1.1. Service architecture

Market data are streamed by the UDP protocol, and missing data are requested through the recovery gateway via TCP connection.

Data streams used in the service are as follows:

1. UDP streams for distributing incremental refreshes of market data
   - OrderBook Incremental Refreshes A / OrderBook Incremental Refreshes B
   - Trades Incremental Refreshes A / Trades Incremental Refreshes B
   - CurrentPriceOfMarket Incremental Refreshes A / CurrentPriceOfMarket Incremental Refreshes B
   - BestPrices Incremental Refreshes A / BestPrices Incremental Refreshes B
   - Commons Incremental Refreshes A / Commons Incremental Refreshes B
2. UDP streams for distributing snapshots (slices) of market data
   - OrderBook Snapshots A / OrderBook Snapshots B
   - BestPrices Snapshots A / BestPrices Snapshots B
   - Commons Snapshots A / Commons Snapshots B
3. TCP sessions to request missing data (recovery gateway)

   TCP connection provides the opportunity to request missing messages in the UDP-stream for distributing incremental refreshes. Restoring slices is not available.

UDP streams transmit the following market data:

- OrderBook streams is a combined queue of orders from one or more liquidity pools, aggregated by price levels. Number of price levels: 50.
- Trades streams is a list of transactions made in liquidity pools by clients of the trading system during the current trading day.
- CurrentPriceOfMarket streams is the current market price that ranges upon transacting or placing a better bid.
- BestPrices streams is the best purchase and sale prices in the combined queue of orders.
- Commons streams is statistical market parameters of liquidity pools.

1.2. UDP streams

Data in UDP streams is sent in the form of messages that are generated in accordance with the FIX protocol and encoded using the FAST protocol (the system uses the FIX protocol version 5 Service Pack 2 and the FAST protocol version 1.1. The protocol version is available on the FIXProtocol.org portal: http://www.fixprotocol.org/fast).

One UDP packet can include one message.

The OrderBook, BestPrices and Commons streams transmit slices and refreshes, while the Trades and CurrentPriceOfMarket streams - only refreshes.

A slice (snapshot) is a set of current data, starting from the opening of the trading day to the present moment. It is transmitted by `MarketDataSnapshotFullRefresh[W]` messages at predetermined intervals. One slice can be transmitted not in one, but in several successive messages. The value of `SecurityID[48]`, field indicates the last message of the slice `MDEntry` group.

A refresh is a set of data that changed as a result of a market event. Refreshes are transmitted via incremental `MarketDataIncrementalRefresh[X]` messages. A single market event generates a single refresh message which may include market data related to several instruments. The market data itself is located in repeating tags of `MDEntry` group.

In the absence of refreshes, the system sends a `Heartbeat` message at a predetermined intervals. It is used to confirm the communication in the channel. The absence of messages for a long period of time means either delays or lack of communication in the channel.

More detailed information about messages transmitted in UDP streams can be found in section 2 Message Specification in UDP Streams.

1.2.1. Data duplication in streams

Data in all UDP streams is duplicated, i.e. distributed in two identical streams *A* and *B*. Each stream is broadcast on a separate multicast address. Parameters for stream connection are specified in the *Network connectivity*.

Messages in these streams are sent simultaneously and have the same numbers. Such duplication makes transmission more reliable by reducing significantly the likelihood of packet loss, since UDP guarantees neither delivery of overall messages nor delivery of messages in the order in which they were sent. Thus, if after message *n*-1 in *A* stream you received message *n*+1, then message *n* could have been received in *B* stream.

1.2.2. The order of receiving and handling data from streams

When you work with streams that transmit both slices and refreshes (OrderBook, BestPrices and Commons), it is better to connect to the slice and refresh streams simultaneously at the beginning of the trading day, then accumulate refreshes until all slices are received. After all messages of one slice have been received, i.e. `LastFragment[893]=Y` field, it is better to refresh it.

Refreshes include ongoing incremental changes as follows: (a) addition, (b) modification, or (c) deletion. Incremental refreshes should only be handled in the forward chronological sequence that corresponds to refresh numbers `RptSeq[83]` for the tool. The first should come a refresh with the number which is greater than `RptSeq[83]` number from the `MarketDataSnapshotFullRefresh[W]` message (a number of the last refresh sent before the slice was generated).

For more information about connecting to market data streams, see the *Network connectivity*.

Handling duplicate data in streams A and B

It is recommended to process both, *A* and *B*, streams, sequencing those messages by numbers. It is recoomended to use a sequence number of the message from the preamble or the `MsgSeqNum[34]` tag.

If a miss in sequence numbers is detected in both streams, you must either wait for the next slice to be received in the corresponding market data stream, or request the missing update messages through the recovery gateway (missed slice messages cannot be recovered).

1.2.3. FAST messages structure in UDP streams

Each FAST message is preceded by a sequence (preamble) of 8 bytes, which contains the value of the `MsgSeqNum[34]` field in unencoded form. This allows to find out the sequence number of the received message without resorting to decoding the FAST message itself.

A FAST message begins with a header - the `FastHeader` component.

Table 1. `FastHeader` component format

Tag	Field	✓	Type	Description
8	BeginString	R	String	Initial string The value is always `FIXT.1.1`

Tag	Field	✓	Type	Description
35	MessageType	R	String	Message Type The value is always `X`
1128	ApplVerID	R	String	Message version. The value is always `9`
49	SenderCompId	R	String	Receive ID. The value is always `BEX`
34	MsgSeqNum	R	int64	Message sequence number
52	SendingTime	R	int64	Message transmission time

FAST messages may not contain all values because encoding may use the following statements as field values:

- constants specified in the template,
- default value (if there is no specified value),
- previous value of this field,
- change with respect to the previous value (delta or increment).

These statements are only relevant for one UDP packet.

FAST message template

To decode messages, use the *fast.xml* template. In the template, the structure of the FAST message is presented as an XML structure in which the elements correspond to the fields of the FAST message and are arranged in the same sequence

The XML element name indicates the data type used (string, integer, etc.), the `name` attribute contains the name of the corresponding FAST message field, and the `id` attribute contains the FIX protocol tag number

The `presence="optional"` attribute is specified for fields that may not be present in the message. Since the template describes the structure of messages for all streams, the attribute will be set if the field may be missing in the message of at least one stream.

For each XML element, a decoding statement is specified (constant, default, copy, delta, increment).

1.3. Recovery gateway

The message recovery gateway allows you to request resending of market data updates if they were lost in transit in UDP streams. Restoring slices is not available.

In the Trades and CurrentPriceOfMarket streams, you can request the history of transactions recorded since the beginning of the trading day. In other streams, it is possible to restore messages for a limited period of time (see the document *Network connectivity*).

A TCP connection is used to request missed messages.

The gateway receives standard FIX messages. All messages distributed by the gateway (session, application, and recovery messages with market data) are encoded using the FAST protocol. This service should only be used if a small number of messages are missing in both streams (A and B).

More detailed information about messages transmitted in the recovery gateway can be found in section 3 Gateway Messaging Specification.

1.3.1. Procedure for data obtaining

To restore messages, the client must:

1. Connect to the gateway via a TCP connection.
2. Initiate a FIX session by sending a `Logon[A]`, FIX message and wait for a FAST message response `Logon[A]` (see section 3.1.1 Session Initialization).
3. Send an ApplicationMessageRequest[BW], specifying the range of message numbers to be restored (see section 3.1.4 Message Numbers), as well as the identifier of the UDP recovery stream from which information is being requested (see the *Network connectivity*).
4. If the request is completed successfully, the client will receive messages with market data, and upon completion of the transmission of the requested messages, an ApplicationMessageRequestAck[BX] report.

If the request is rejected, the client will receive a Reject[3] message and an ApplicationMessageRequestAck[BX] report.

5. Upon completion of data transfer, the system sends a FAST `Logout` message to the client and waits for a response FIX `Logout` message (see section 3.1.7 Ending a session).
6. After the FIX session is completed, the TCP connection is closed.

Within one session, you can send several requests for data recovery (to limit the frequency of sending requests, see sections 3.1.3 and 3.2.2). It is recommended to establish a session at the beginning of work and not close it, maintaining it with `Heartbeat` messages (see section 3.1.2 Heartbeat messages).

1.3.2. FAST messages structure in the recovery gateway

Each FAST message transmitting by the recovery gateway is preceded by a 4-byte sequence (preamble) that contains the length of each message. This is done to be able to separate messages from each other in a continuous TCP stream.

The format of the recovered FAST messages is completely identical to the format of the messages that were sent earlier in the corresponding UDP market data stream (see section 1.2.3 FAST messages structure in UDP streams).

1.3.3. FIX messages structure in the recovery gateway

Each FIX message begins with a header (the `FixHeader`component) and ends with a trailer.

Only three fields have a fixed position in the header: the first field is `BeginString[8]=FIXT.1.1`, followed by the `BodyLength[9]` field and then `MsgType[35]`. The value of `BodyLength[9]` is the length of the message in bytes, which is calculated starting from the tag following `BodyLength[9]` ,and ending with the separator before `CheckSum[10]`.

Table 2. `FixHeader` component format

Tag	Field	✓	Type	Description
8	BeginString	R	String	First field of the message Possible values: • `FIXT.1.1`
9	BodyLength	R	Length	Message body length in bytes
35	MsgType	R	String	Message Type
49	SenderCompId	R	String	Receive ID
56	TargetCompId	R	String	Recipient ID
34	MsgSeqNum	R	SeqNum	Message sequence number
52	SendingTime	R	UTCTimestamp	Message transmission time

The message trailer consists of a `CheckSum[10]`, tag in which a three-byte simple checksum is written.

Table 3. Message Trailer Format

Tag	Field	✓	Type	Features
10	CheckSum	R	String	Message checksum (3 bytes)

1.4. Volume dimension

Regardless of the routing direction, the volume of any order is indicated in the internal lots of the trading system.

The internal lot size in the trading system may differ from the lot size in the external system for the same trading instrument.

The lot size of a trading instrument depends on the period and is indicated in the corresponding on-balance-sheet instrument.

2. Message Specification in UDP Streams

This section provides a description of FAST messages transmitting in UDP streams. For more information about UDP streams, see section 1.2, for information about the ability to recover missed messages, see section 1.3 Recovery Gateway.

Message formats are presented in the table form. The message type is indicated in parentheses after the message name and is defined in the `MsgType[35]` field. For example, `MarketDataIncrementalRefresh[X]`.

The mandatory presence of the field in the message is indicated in the column with √. Possible values:

- R [required] — required;
- N [nonrequired] — nonrequired;
- C [conditionally required] — necessary under a certain condition.

Messages may contain repeating groups of tags, which are indicated in the document by the ">" symbol.

The data type is determined by the FAST protocol.

2.1. OrderBook stream

The OrderBook stream displays the volumes of price levels: the slice contains no more than 50 levels; the updates also apply to the 50 visible price levels.

The value of the `MDEntryType[269]` field is `0` for buy quotes and `1` for sell quotes. In this case, the price is indicated in `MDEntryPx[270]`, and the total volume of orders at the price level is indicated in `MDEntrySize[271]`.

Table 4. `MarketDataIncrementalRefresh[X]` message format for OrderBook stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
268	NoMDEntries	Length	R	Number of entries in the component
83	> RptSeq	Int64	R	Instrument update number. Corresponds to `RptSeq[83]` in the `MarketDataFullSnapshotRefresh[W]` message
48	> SecurityID	Int32	R	Trading instrument ID
22	> SecurityIDSource	Int32	R	Liquidity pool ID
279	> MDUpdateAction	Int32	R	Update type. Values are: • `0` (add); • `1` (replacement); • `2` (removal)
269	> MDEntryType	String	R	Record type. Values are: • `0` (buy price); • `1` (sales price)
270	> MDEntryPx	Decimal	R	Price
271	> MDEntrySize	Int64	R	Total volume of orders at price level, in lots
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Update time

Table 5. MarketDataSnapshotFullRefresh[W] message format for OrderBook stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
48	SecurityID	Int32	R	Trading instrument ID
22	SecurityIDSource	Int32	R	Liquidity pool ID
83	RptSeq	Int64	R	Instrument update number. Corresponds to RptSeq[83] in the MarketDataIncrementalRefresh[X] message
893	LastFragment	String	R	Sign of the message, the last in the series, which forms a slice for this instrument. Values are: • Y (last message, slice formed); • N (not the last message, the slice has not yet been formed);
268	NoMDEntries	Length	R	Number of entries in the component
269	> MDEntryType	String	R	Record type. Values are: • 0 (buy price); • 1 (sales price)
270	> MDEntryPx	Decimal	R	Price
271	> MDEntrySize	Int64	R	Total volume of orders at price level, in lots
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Update time

2.2. Trades stream

The Trades stream transmits data on transactions concluded during the trading day: when receiving information from the liquidity pool about the conclusion of one or more transactions , a MarketDataIncrementalRefresh[X], message is generated, in which each transaction is an entry in a repeating component.

The value of the MDEntryType[269] field is always 2. ПThe price is specified in the MDEntryPx[270], field, the volume in MDEntrySize[271], and the MDEntryID[278] field contains the trade identifier assigned by the liquidity pool.

Table 6. MarketDataIncrementalRefresh[X] message format for Trades stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
268	NoMDEntries	Length	R	Number of entries in the component
83	> RptSeq	Int64	R	Instrument update number
48	> SecurityID	Int32	R	Trading instrument ID
22	> SecurityIDSource	Int32	R	Liquidity pool ID
279	> MDUpdateAction	Int32	R	Update type. Value: 0 (add)
269	> MDEntryType	String	R	Record type. The value is always 2 (transaction)
270	> MDEntryPx	Decimal	R	Price

Tag	Field	Type	✓	Description
271	> MDEntrySize	Int64	R	Volume in lots
272	> MDEntryDate	Int32	R	Transaction date
273	> MDEntryTime	Int32	R	Transaction time
278	> MDEntryID	Int64	R	Transaction ID
828	> TrdType	Int32	R	Transaction type. Value: `1` (regular transaction)

2.3. CurrentPriceOfMarket stream

The CurrentPriceOfMarket stream transmits data about the current market price: when receiving information from the liquidity pool about changes in the current market price, a `MarketDataIncrementalRefresh[X]`, message is generated, in which each message represents an entry in a repeating component.

The current market price is calculated continuously based on transaction prices and firm quotes according to the following rules:

1. when a transaction is formed, the price becomes equal to the transaction price;
2. when an anonymous purchase order appears in the queue with a price higher than the current market price, the current market price is equal to the price of the purchase order;
3. when an anonymous sell order appears in the queue with a price lower than the current market price, the current market price is equal to the price of the sell order.

For a transaction, the value of the `MDEntryType[269]` field is always `2`. In this case, the price is indicated in the `MDEntryPx[270]`, field, the volume - in `MDEntrySize[271]`, and the `MDEntryID[278]` field contains the transaction identifier assigned by the liquidity pool.

For a quote, the value of the `MDEntryType[269]` field is `0` for buy quotes and `1` for sell quotes. In this case, the price is indicated in the `MDEntryPx[270]`, field, and the `MDEntrySize[271]`, `MDEntryID[278]` and `TrdType[828]` fields will be absent.

Table 7. `MarketDataIncrementalRefresh[X]` message format for CurrentPriceOfMarket stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
268	NoMDEntries	Length	R	Number of entries in the component
83	> RptSeq	Int64	R	Instrument update number
48	> SecurityID	Int32	R	Trading instrument ID
22	> SecurityIDSource	Int32	R	Liquidity pool ID
279	> MDUpdateAction	Int32	R	Update type. Value: `0` (add)
269	> MDEntryType	String	R	Record type. Values are: • `0` (best buy price); • `1` (best sell price); • `2` (last trade price)
270	> MDEntryPx	Decimal	R	Current market price
271	> MDEntrySize	Int64	R	For a transaction - volume in lots
272	> MDEntryDate	Int32	R	Date to conclude a transaction or place an order

Tag	Field	Type	✓	Description
273	> MDEntryTime	Int32	R	Time to conclude a transaction or place an order
278	> MDEntryID	Int64	R	For a transaction - transaction ID
828	> TrdType	Int32	R	For a transaction - the transaction type. Value: `1` (regular transaction)

2.4. BestPrices stream

The BestPrices stream transmits messages containing the price level with the best buy price (`MDEntryType[269]=0`), the price level with the best sell price (`MDEntryType[269]=1`) and the last transaction (`MDEntryType[269]=2`) for current session. The `MDEntryPx[270]` field indicates the price level or transaction price, and `MDEntrySize[271]` indicates the total volume of orders at the price level or the transaction volume in lots. The `MDEntryTime[273]` field contains the time of the last update of the price level or the time of the transaction.

Table 8. `MarketDataIncrementalRefresh[X]` message format for BestPrices stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
268	NoMDEntries	Length	R	Number of entries in the component
83	> RptSeq	Int64	R	Instrument update number. Corresponds to `RptSeq[83]` in the `MarketDataFullSnapshotRefresh[W]` message
48	> SecurityID	Int32	R	Trading instrument ID
22	> SecurityIDSource	Int32	R	Trading platform ID
279	> MDUpdateAction	Int32	R	Update type. Values are: • `0` (add); • `1` (replacement); • `2` (removal)
269	> MDEntryType	String	R	Record type. Values are: • `0` (best buy price); • `1` (best sell price); • `2` (last trade price)
270	> MDEntryPx	Decimal	R	Price
271	> MDEntrySize	Int64	R	Total volume of orders at the price level or transaction volume in lots
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Time of the last update of the price level or time of the transaction

Table 9. `MarketDataSnapshotFullRefresh[W]` message format for BestPrices stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
48	SecurityID	Int32	R	Trading instrument ID

Tag	Field	Type	✓	Description
22	SecurityIDSource	Int32	R	Liquidity pool ID
83	RptSeq	Int64	R	Instrument update number. Corresponds to `RptSeq[83]` in the `MarketDataIncrementalRefresh[X]` message
893	LastFragment	String	R	Sign of the message, the last in the series, which forms a slice for this instrument. Values are: • `Y` (last message, slice formed); • `N` (not the last message, the slice has not yet been formed);
268	NoMDEntries	Length	R	Number of entries in the component
269	> MDEntryType	String	R	Record type. Values are: • `0` (best buy price); • `1` (best sell price); • `2` (last trade price)
270	> MDEntryPx	Decimal	R	Price
271	> MDEntrySize	Int64	R	Total volume of orders at the price level or transaction volume in lots
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Time of the last update of the price level or time of the transaction

2.5. Commons stream

The Commons stream transmits messages containing statistical market parameters of liquidity pools. The parameter type is specified in `MDEntryType[269]`, the set of fields to be filled in depends on the specified type. For updates, a message is generated when one or more statistical parameters change. The slice is broadcast periodically.

Table 10. `MarketDataIncrementalRefresh[X]` message format for Commons stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
268	NoMDEntries	Length	R	Number of entries in the component
83	> RptSeq	Int64	R	Instrument update number. Corresponds to `RptSeq[83]` in the `MarketDataFullSnapshotRefresh[W]` message
48	> SecurityID	Int32	R	Trading instrument ID
22	> SecurityIDSource	Int32	R	Liquidity pool ID
279	> MDUpdateAction	Int32	R	Update type. Values are: • `0` (add); • `1` (replacement); • `2` (removal)
269	> MDEntryType	String	R	Parameter type. Values are: • `2` (latest trade price); • `4` (opening price) - comes only after the first trade;

Tag	Field	Type	✓	Description
				• `5` (closing price); • `6` (settlement price of latest intermediate clearing); • `7` (highest price) - comes only if a transaction is made with a price greater than the previous maximum; • `8` (lowest price) - comes only if a transaction is made with a price less than the previous minimum; • `9` (volume weighted average price); • `A` (volume disbalance at closing auction); • `b` (indicative quote); • `B` (turnover of anonymous trades, in instrument lots); • `c` (closing auction volume); • `d` (turnover of trades at closing auction price, in instrument units); • `f` (current price); • `g` (turnover of latest trade in price currency); • `h` (change in current price against official closing price of previous day); • `i` (lowest current price); • `j` (latest trade price involved in current price); • `k` (official closing price of previous day); • `L` (aggregated turnover in instrument units); • `M` (aggregated turnover in currency); • `n` (settlement price of latest main clearing); • `N` (highest bid price); • `o` (total turnover, in instrument lots); • `O` (lowest ask price); • `p` (total asset turnover); • `q` (total currency turnover); • `r` (turnover of anonymous trades, in instrument units); • `s` (currency turnover of anonymous trades); • `t` (total number of trades); • `u` (volume of buy orders); • `v` (volume of sell orders); • `w` (number of anonymous trades); • `W` (closing auction price); • `x` (number of buy orders); • `y` (number of sell orders); • `z` (official current price)
270	> MDEntryPx	Decimal	N	Price Filled when `MDEntryType[269] = 2, 4, 5, 7, 8, 9, f, n, 6, N, O, W, z, k, j, i, h, b`
271	> MDEntrySize	Int64	N	Volume in lots Filled when `MDEntryType[269] = 2, c, d, r, s, t, L, M, o, p, q, x, y, u, v, w, B, g, A`
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Update time

Tag	Field	Type	✓	Description
286	> OpenCloseSettleFlag	String	N	Indicator of the time period for which the parameter is calculated. Filled when `MDEntryType[269] = 9, k, 2`. Values are: • `1` (main session); • `4` (previous day) A combination of values is possible. The separator is a space.

Table 11. `MarketDataSnapshotFullRefresh[W]` message format for Commons stream

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
48	SecurityID	Int32	R	Trading instrument ID
22	SecurityIDSource	Int32	R	Liquidity pool ID
83	RptSeq	Int64	R	Instrument update number. Corresponds to `RptSeq[83]` in the `MarketDataIncrementalRefresh[X]` message
893	LastFragment	String	R	Sign of the message, the last in the series, which forms a slice for this instrument. Values are: • `Y` (last message, slice formed); • `N` (not the last message, the slice has not yet been formed);
268	NoMDEntries	Length	R	Number of entries in the component
269	> MDEntryType	String	R	Parameter type. Values are: • `2` (latest trade price); • `4` (opening price) - comes only after the first trade; • `5` (closing price); • `6` (settlement price of latest intermediate clearing); • `7` (highest price) - comes only if a transaction is made with a price greater than the previous maximum; • `8` (lowest price) - comes only if a transaction is made with a price less than the previous minimum; • `9` (volume weighted average price); • `b` (indicative quote); • `c` (closing auction volume); • `f` (current price); • `n` (settlement price of latest main clearing); • `x` (number of buy orders); • `y` (number of sell orders); • `u` (volume of buy orders); • `v` (volume of sell orders); • `w` (number of anonymous trades); • `B` (turnover of anonymous trades, in instrument lots); • `r` (turnover of anonymous trades, in instrument units); • `s` (currency turnover of anonymous trades); • `t` (total number of trades); • `o` (total turnover, in instrument lots);

Tag	Field	Type	✓	Description
				• `p` (total asset turnover); • `q` (turnover in currency); • `N` (highest bid price); • `o` (total turnover, in instrument lots); • `W` (closing auction price); • `z` (official current price) • `k` (official closing price of previous day); • `g` (turnover of latest trade in price currency); • `L` (aggregated turnover in instrument units); • `M` (aggregated turnover in currency); • `A` (volume disbalance at closing auction); • `j` (latest trade price involved in current price); • `i` (lowest current price); • `h` (change in current price against official closing price of previous day);
270	> MDEntryPx	Decimal	N	Price Filled when `MDEntryType[269] = 2, 4, 5, 7, 8, 9, f, n, 6, N, O, W, z, k, j, i, h, b`
271	> MDEntrySize	Int64	N	Volume in lots Filled when `MDEntryType[269] = 2, c, d, r, s, t, L, M, o, p, q, x, y, u, v, w, B, g, A`
272	> MDEntryDate	Int32	R	Date of Update
273	> MDEntryTime	Int32	R	Update time
286	> OpenCloseSettleFlag	String	N	Indicator of the time period for which the parameter is calculated. Filled when `MDEntryType[269] = 9, k, 2`. Values are: • `1` (main session); • `4` (previous day) A combination of values is possible. The separator is a space.

2.6. Heartbeat messages

The system sends the `Heartbeat[0]` message to the update stream if there are no other messages.

In some cases, the `Heartbeat[0]` message may arrive in a slice.

Table 12. `Heartbeat[0]` Message Format

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header

3. Message Specification in the Recovery Gateway

This section provides a description of messages transmitting in the recovery gateway. For more information about the recovery gateway, see Section 1.3 Recovery Gateway.

Message formats are presented in the table form. The message type is indicated in parentheses after the message name and is defined in the `MsgType[35]`. For example, `MarketDataIncrementalRefresh[X]`.

The mandatory presence of the field in the message is indicated in the column with √. Possible values:

- R [required] — required;
- N [nonrequired] — nonrequired;
- C [conditionally required] — necessary under a certain condition.

The type of message encoding (FIX or FAST protocol) is determined by the direction of the message: from the client or from the gateway.

3.1. Session layer

The session layer is compliant with FIX Session Protocol 1.1.

A FIX session is established over TCP-connection between a client gateway and the trading system gateway. Session participants are identified by fields `SenderCompID[49]` and `TargetCompID[56]`.

The ID of the trading system gateway is ECN_EQR and that of a client is the user name.

3.1.1. Session initialization

The `Logon[A]` initiates a FIX session or confirms its start. After establishing a TCP connection, the session initiator (client) sends this message and expects `Logon[A]` in response. The `ResetSeqNumFlag[141]`, `Password[554]` are filled only by the client, `NextExpectedMsgSeqNum[789]` is filled only by the trading system.

A receipt of a correct `Logon[A]` shall always result in sending response message `Logon[A]`, even if `MsgSeqNum[34]` is higher than expected. Any error in `Logon[A]` shall cause a disconnection, and the number of the next expected message will not be changed.

Table 13. Format of message `Logon[A]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
98	EncryptMethod	R	Int	Method of encryption. Possible values: • `0` (Encryption not supported)
108	HeartBtInt	R	Int	Timeout. Value in seconds. Recommended value: from 20 to 30
554	Password	N	String	Login password
1137	DefaultApplVerId	R	String	Protocol version. Possible values: • `9` (FIX50SP2)

3.1.2. Heartbeat messages

Client and trading system exchange messages `Heartbeat[0]` to monitor the connection status. A heartbeat is to be sent by a party if it passed no messages (of the session or application layer) within the heartbeat interval. A client specifies the timeout value `HeartBtInt[108]` in the message `Logon[A]`; the recommended value is from 20 to 30 seconds.

After the absence of messages during an interval exceeding `HeartBtInt[108]`, a party should send `TestRequest[1]` with the `TestReqID[112]` identifier. In answer the counterparty should send `Heartbeat[0]` containing the same identifier. If no response within the heartbeat interval, the system disconnects after sending message `Logout[5]` to a client. A client is expected to act the same.

If a client prefers not to send or receive heartbeats during this FIX session, `0` should be specified in `HeartBtInt[108]`.

Table 14. Format of message `HeartBeat[0]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
112	TestReqId	C	String	Request ID of TestRequest, to which this message is a response

Table 15. Format of message `TestRequest[1]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
112	TestReqId	R	String	Request ID Maximum length is 32 characters. Valid characters are letters and numbers

3.1.3. Send rate limit for session messages

The rate at which a client can send session level messages into the system is limited. When client's send rate exceeds the limit, the system terminates the user session.

3.1.4. Message numbers

All messages exchanged by the parties within a FIX session have a sequence number. The number is specified in the field `MsgSeqNum[34]` in the header of each message. The number of each subsequent message of a FIX session should be incremented, except for the cases of forced increase of the message number by request `SequenceReset[4]`.

As reference information for a client, the number of the last message processed by the trading system is indicated in the field `LastMsgSeqNumProcessed[369]`.

When receiving a message with the number higher than expected, a client should send `ResendRequest[2]`.

When the trading system receives a messages with the numbers lower than expected, a client will be sent `Logout[5]` with the value `SessionStatus[1409]=1` followed by TCP disconnection.

3.1.5. Message resend request

To request the messages previously sent by the trading system, a client can use `ResendRequest[2]`, in particular for the purpose of restoring missing messages. When receiving a message with the number higher than expected, a client should also send `ResendRequest[2]`.

A client may request resending of all messages, sent during the current and previous trading days. If a client has intentionally reset message numbering (`ResetSeqNumFlag[141]=Y` in the message `Logon[A]`), a request for resending messages, sent prior to the reset, is not possible.

The fields `BeginSeqNo[7]` and `EndSeqNo[16]` set the range of requested messages. If a client uses `BeginSeqNo[7]=0` and `EndSeqNo[16]=0`, the system will resend all messages starting from the lowest number available. If a client specifies `0` only for `EndSeqNo[16]`, the system will resend all messages of current trading session starting from `BeginSeqNo[7]`. All possible cases are as follows:

1. `BeginSeqNo=n, EndSeqNo=m` — request for messages from *n* to *m*,
2. `BeginSeqNo=0, EndSeqNo=n` — request for messages from the lowest number available to *n*,
3. `BeginSeqNo=n, EndSeqNo=0` — request for messages from *n* to the highest number available,
4. `BeginSeqNo=0, EndSeqNo=0` — request for all available messages.

Number range for requested messages is not limitless (for more details please refer to *Network Connectivity*). When a client requires more messages, a client should send several consecutive requests. If the gateway has not finished sending messages on previous request, new requests for messages will be rejected.

Table 16. Format of message `ResendRequest[2]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
7	BeginSeqNo	R	SeqNum	Number of first requested message
16	EndSeqNo	R	SeqNum	Number of last requested message

In response to `ResendRequest[2]`, the system will return the requested messages or will modify `MsgSeqNum[34]` by the message `SequenceReset[4]`. The value `PossDupFlag[43]=Y` is a flag of resent messages.

After receiving `ResendRequest[2]`, the trading system will resend messages of the application layer only and will never resend session messages. Therefore, in response to message resend request a client should expect, among others, `SequenceReset[4]` with `GapFillFlag[123]=Y` and the number of the next expected message in `NewSeqNo[36]`.

If a client wants to increase the message number expected from the system, a client should sent `SequenceReset[4]` with `GapFillFlag[123]=N` and the new expected number in the field `NewSeqNo[36]`.

During resending, the trading system may also transmit new trading messages, so a client should also expect messages with a number exceeding the requested range. To ensure data completeness, a client is not recommended to ignore such messages with larger numbers.

Table 17. Format of message `SequenceReset[4]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
36	NewSeqNo	R	SeqNum	New sequence number
123	GapFillFlag	N	Boolean	Indicator of gap fill. Possible values: • `N` (mode Reset ignoring field MsgSeqNum; specified by the client); • `Y` (mode GapFill using field MsgSeqNum; specified by the server)

3.1.6. Message numbers reset by the client

The client can reset sequence numbers by the value `ResetSeqNumFlag[141]=Y` in the `Logon[A]` message. This functionality may be useful to avoid procedures for requesting and restoring missing messages. It is not recommended to use this feature during the trading session, because trading messages already sent will not be available for resend request.

In response to a client `Logon[A]` with `ResetSeqNumFlag[141]=Y` the trading system will send `Logon[A]` with `ResetSeqNumFlag[141]=Y`, `MsgSeqNum[34]=1`, and `NextExpectedMsgSeqNum[789]=2`. The next expected message number is `2`.

3.1.7. Session termination

`Logout[5]` initiates or confirms the session termination and shall be sent after a long-term absence of messages (please refer to section 3.1.2) or after receiving a message with number lower than expected.

The reason for rejection is specified in the tag `SessionStatus[1409]`. The field `Text[58]` may contain report on the session termination reasons.

Table 18. Format of message `Logout[5]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
1409	SessionStatus	N	Int	Numeric code of the reason. To be filled by the trading system only. Possible values: • `5` (invalid login or password); • `5000` (violation of message exchange protocol); • `5002` (client not active); • `5003` (server stopped); • `5200` (login is already in active session)
58	Text	N	String	Report on session termination reason

3.1.8. Message rejection

The message `Reject[3]` is sent in response to any invalid message (incorrectly generated or transmitted) from the other party. The reasons for rejection may be the absence of required fields, invalid message type or length, and invalid data type, etc. All session layer messages with invalid value of any field are also rejected by the message `Reject`.

The system specifies the rejected message number in the field `RefSeqNum[45]`. The value `RefSeqNum[45]=0` means that the field `MsgSeqNum[34]` is missing in the rejected message. If the system detects an invalid value, the tag will be indicated in `RefTagID[371]`. The field `SessionRejectReason[373]` may contain the rejection reason code and `Text[58]` may have a textual description of error.

Table 19. Format of message `Reject[3]`

Tag	Field	✓	Type	Description
	Header	R		Standard header
45	RefSeqNum	R	SeqNum	Number of rejected message
371	RefTagId	N	Int	Tag which caused the error
372	RefMsgType	N	String	Type of rejected message
373	SessionRejectReason	N	Int	Reason for rejection. Possible values: • `0` (invalid tag number); • `1` (required tag missing); • `2` (invalid tag in the message); • `4` (tag with no value); • `5` (invalid value); • `6` (invalid datatype); • `11` (incorrect message type); • `13` (tag repeated in message); • `14` (tag CheckSum[10] misplaced); • `15` (tag from the group misplaced); • `16` (invalid number of group entries)
58	Text	N	String	Error report

3.1.9. Disconnection

The TCP connection will be dropped if the server receives a message with an error in one of the first three fields (`BeginString[8]`, `BodyLength[9]`, and `MsgType[35]`) or the `Logon[A]` message of invalid format or containing invalid values.

3.2. Application layer

3.2.1. Data request

For market data recovery, the client is required to send a message to the gateway an `ApplicationMessageRequest[BW]`, message, specifying the FAST stream ID.

The range of requested data must be specified in the `ApplBegSeqNum[1182]` and `ApplEndSeqNum[1183]` fields in one of the following ways:

1. `ApplBegSeqNum[1182]=n, ApplEndSeqNum[1183]=m` — request messages from *n* to *m*,
2. `ApplBegSeqNum[1182]=n, ApplEndSeqNum[1183]=n` — message request *n*,
3. `ApplBegSeqNum[1182]=0, ApplEndSeqNum[1183]=n` — request messages starting from the smallest available number to *n*,
4. `ApplBegSeqNum[1182]=n, ApplEndSeqNum[1183]=0` — request messages starting from *n* to the last available number, but not more than the maximum available amount,
5. `ApplBegSeqNum[1182]=0, ApplEndSeqNum[1183]=0` — request all available messages, but not more than the maximum available amount.

FAST stream identifiers and the maximum amount of message numbers in one request cannot exceed the specified maximum value (see restrictions in the document *Network connectivity* clause 1.3).

In the current version, it is only possible to request one range per stream.

If the request is completed successfully, the client will receive messages with market data, and upon completion of the data transmission, an `ApplicationMessageRequestAck[BX]` report. If the request is rejected, the client will only receive an `ApplicationMessageRequestAck[BX]` report.

Table 20. `ApplicationMessageRequest[BW]` FIX message format

Tag	Field	Type	✓	Description
	FixHeader		R	FIX message header
1346	ApplReqID	String	R	Request ID
1347	ApplReqType	Int	R	Request type. Possible values: `0` (message re-query)
1351	NoApplIDs	NumInGroup	R	The number of entries in the `ApplIDRequestGroup` group. Possible values: `1`
1355	RefApplID	String	R	FAST stream ID
1182	ApplBegSeqNum	SeqNum	R	Number of the first message of the requested range
1183	ApplEndSeqNum	SeqNum	R	Number of the last message of the requested range

3.2.2. Send rate limit for client requests

The rate at which a client can send requests into the system is limited. There are two limits:

1. When the first threshold of send rate is reached, the system starts declining the application level requests and transmits the report on rejecting request with reason "Number of messages exceeded limit".
2. When the second threshold of send rate is reached, the system terminates the user session.

3.2.3. Request execution report

The client will receive an `ApplicationMessageRequestAck[BX]` report in the following cases:

1. Complete data transfer (ApplResponseType[1348]=0). The ApplTotalMessageCount[1349] field will indicate the number of messages transmitted.
2. Data request rejection (ApplResponseType[1348]=1 or 2). In the ApplResponseError[1354] field will indicate the reason for the rejection.

Due to technology delays in transferring data from the market data broadcast service to the recovery gateway, some messages may not be available when re-queried. If the value of the ApplEndSeqNum[1183] field in the report is less than that specified in the request, then you should request the unreceived messages again.

Table 21. ApplicationMessageRequestAck[BX] FAST message format

Tag	Field	Type	✓	Description
	FastHeader		R	FAST message header
1353	ApplResponseID	String	R	Request response ID
1346	ApplReqID	String	R	Request ID
1347	ApplReqType	Int32	R	Request type. Possible values: 0 (message re-query)
1348	ApplResponseType	Int32	R	Response type. If rejected, the reason is indicated in the ApplResponseError[1154] field. Possible values: • 0 (request is fully satisfied); • 1 (request rejected - stream not found); • 2 (request rejected - requested messages not found)
1349	ApplTotalMessageCount	Int32	R	Number of messages sent as a result of the request
1351	NoApplIDs	Length	R	The number of entries in the ApplIDRequestGroup group. Possible values: 1
1355	RefApplID	String	R	FAST stream ID
1182	ApplBegSeqNum	Int64	R	Number of the first message of the requested range
1183	ApplEndSeqNum	Int64	R	Number of the last message sent upon request
1354	ApplResponseError	Int32	N	Reason for request rejection. Possible values: • 0 (non-existent stream); • 1 (unavailable message numbers); • 2 (insufficient access rights)