THE SMALL EXCHANGE
MARKET DATA FEED SPECIFICATION

VERSION
0.12

19 November 2019

1 Table of Contents

2 Revision History

Version	Date	Author	Description
0.1	Oct-20-2018	Sergey Samushin Vladimir Danilov	Initial draft
0.2	Oct-24-2018	Sergey Samushin Vladimir Danilov	Added support for order book messages
0.3	Dec-12-2018	Sergey Samushin Vladimir Danilov	Updated order book messages fields and ordering
0.4	Jan-22-2019	Sergey Samushin	Added lines description
0.5	Jan-31-2019	Sergey Samushin	Added hours of operation Merged instruments and order book snapshots into a single line
0.6	Feb-12-2019	Yury Kudryashov	Retransmission messages and protocol description Event ordering guarantees Updates in field sizes Links between order book updates and corresponding trades
0.7	Mar-4-2019	Yury Kudryashov	LastIncrementalSeqNo field in snapshot messages Field reordering CFICode field renamed to CfiCode TradeDate, FirstTradeDate, LastTradeDate renamed to TradingSessionDate, FirstTradingSessionDate, LastTradingSessionDate SnapshotInstrumentsCount field in all snapshot messages moved to be the field #6. SBE TemplateIds corrected
0.8	Jun-7-2019	Sergey Samushin	Added message format and description of Index Data Feed
0.9	Jul-17-2019	Yury Kudryashov	Added Open-High-Low-Close index values Added StrategyType to Multi-leg definitions

0.10	Aug-08-2019	Vladimir Danilov Sergey Samushin	SBE schema version is updated from 2 to 4. Added in v3: - LastTradePrice, LastTradeSize, LastTradeTime, Volume to Trade Correct Incremental and Trade Bust Incremental messages - StrategyType to multi leg definition messages - Order time to OrderBookSnapshot message Added in v4: - SessionDate, OpenPrice, HighPrice, LowPrice, ClosePrice for IndexValueSnapshot message
0.11	Oct-03-2019	Vladimir Danilov	SBE schema version is updated to 5. Enums SettlementPriceType/OpenPriceType are extended to contain NO_PRICE value PutOrCall enum is extended to contain NOT_OPTION value. LastTradeTime field is switched to TimestampOptional Fields of optional presence with type Price migrated to PriceOptional
0.12	Nov-15-2019	Vladimir Danilov	Wording of last trade fields in Trade Bust/Correct messages (copy-pasted from Trades Incremental), would make more sense with "Last trade after bust/correct" Added details on time priority value for implied orders. Correction of byte offsets for snapshot messages. Added details on snapshot line incarnation.

3 Confidentiality/Disclaimer

This specification is being forwarded to you strictly for informational purposes and solely for the purpose of developing or operating systems for your use that interact with systems of Small Exchange, Inc. ("Small Exchange" or "Exchange"). This specification is proprietary to Small Exchange. Small Exchange reserves the right to withdraw, modify, or replace this specification at any time, without prior notice. No obligation is made by the Small Exchange regarding the level, scope or timing of the Small Exchange's implementation of the functions or features discussed in this specification. The specification is provided "AS IS," "WITH ALL FAULTS". The Small Exchange makes no warranties to this specification or its accuracy, and disclaims all warranties, whether express, implied, or statutory related to the specification or its accuracy. This document is not intended to represent an offer of any terms by the Small Exchange. While reasonable care has been taken to ensure that the details contained herein are true and not misleading at the time of publication, no liability whatsoever is assumed by the Small Exchange for any incompleteness or inaccuracies. By using this specification, you agree that you will not, without prior written permission from the Small Exchange, copy or reproduce the information in this specification except for the purposes noted above. You further agree that you will not, without prior written permission from the Small Exchange, store the information contained in this specification in a retrieval system, or transmit it in any form or by any means, whether electronic, mechanical, or otherwise except for the purposes noted above. In addition, you agree that you will not, without prior written permission from the Small Exchange, permit access to the information contained herein except to those with a need-to-know for the purposes noted above.

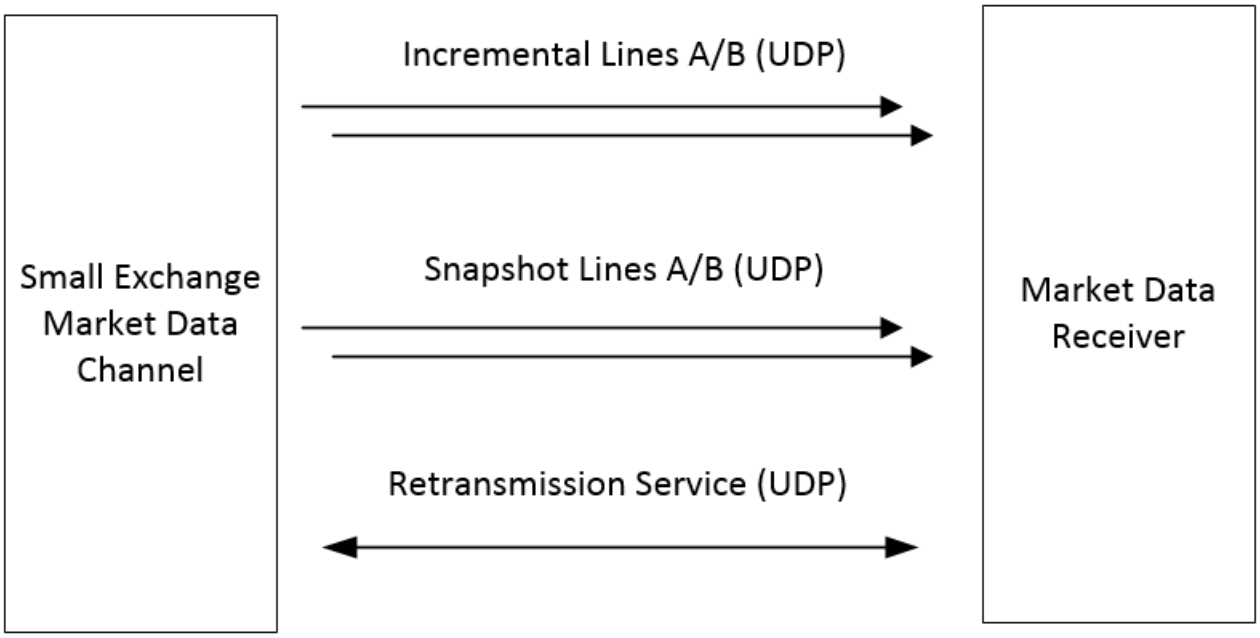
4 Overview

4.1 Order Book Feed

The Small Exchange Order Book Feed allows market participants to receive market information such as updated orders, trades, market state changes and instrument definitions for the Exchange market. The Feed consists of Market Data Channels; each Channel facilitates distributing market data information for a group of instruments provided by the services:

- **Incremental Lines** – UDP multicast group with sequenced messages with real-time market data updates – orders, trades, market state changes, instrument definitions.
- **Snapshot Lines** – UDP multicast group replaying market data snapshots in cycles to fetch initial market state when a session is established or re-established.
- **Retransmission Service** – stands for requesting retransmit for lost messages in the Incremental line

Note that in the Small Exchange production environment multiple Market Data Channels may be set up for publishing data for a different instrument products.



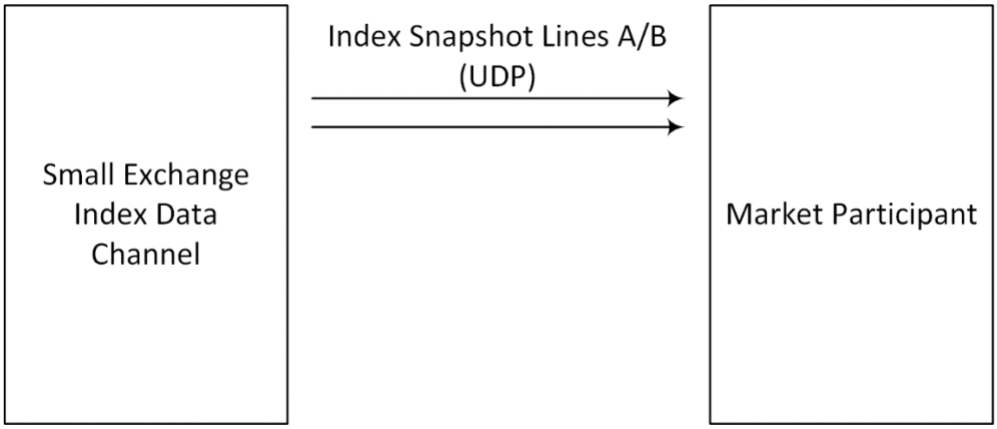
Key notes:

- Each line has a sequence incrementing with each message sent in a line
- UDP packets have size of up to 1300 bytes length.
- Little-endian byte order is used for messages, headers and packets framing

4.2 Index Feed

The Small Exchange Index Feed allows market participants to receive current values of the Exchange indices.

The Feed consists of Index Data Channels; each Channel facilitates distributing index data information for a group of instruments provided over **Index Snapshot Lines**. Snapshot lines are UDP multicast groups replaying current indices values in cycles.



5 Message Structure

Messages in the lines and retransmission service are encoded with Simple Binary Encoding v1.0 format (See https://www.fixtrading.org/standards/sbe/). Little-endian byte order is used for messages, headers and packets framing. UDP packets have size of up to 1300 bytes.

5.1 Packet Structure

Each UDP packet may contain multiple messages with total payload size up to 1300 bytes. The packet has the following structure:

Packet Header	Message 1 Header	Message 1 Data	...	Message N Header	Message N Data
10 bytes	10 bytes	<Variable length>		10 bytes	<Variable length>

5.2 Message Sequencing and Incarnations

Each message in a packet has a transport-level sequence that can be calculated as *MessageSequence* (from the Packet Header) plus the message sequence within the packet.

Within each market data channel there is a notion of 'incarnation' which is the number increasing whenever message sequence is reset to 1. The Exchange can reset the message sequence at any time. Under normal conditions, sequence reset is marked with a packet containing the *Incarnation End* flag – this can either be a regular packet or a special Incarnation End Packet. Further messages will be published with a new incarnation number (which is the previous number increased by 1). First message sequence in any incarnation is always 1. Snapshot line is always reset together with incremental messages. Current snapshot cycle (if any) is aborted on reset.

In the rare case of a serious failure on the Exchange side, incarnation number may increase without proper *Incarnation End* flag sent in any previous message. If such a situation is detected, the clients are expected to drop the existing market state and restart their market data session from scratch (see Session Establishment and Maintenance Algorithm). Note that the incarnation number in this case can be increased arbitrarily – for example, it can go from 2 to 200 when the sequence is reset.

The clients might follow a simple algorithm to deal with sequence resets:

1. Start a market data session as described in Session Establishment and Maintenance Algorithm
2. Process each incoming packet in the order of message sequence as described in Packet Ordering and Delivery Guarantees
3. If the *Incarnation End* flag is set in an incoming packet, reset the next expected message sequence to 1 and increase the expected incarnation number by 1.
    a. Note that Incarnation End Packets announcing sequence reset may be published multiple times in a row
4. When a new packet with sequence of 1 is received in the new incarnation, it is safe to apply this update to the market state.

5. If a new packet has increased incarnation number and there were no packets with the *Incarnation End* flag set, this is an indication that the market state should be reset and re-initialized from scratch. Some examples are:
    a. *Incarnation 1 without Incarnation End* followed by *Incarnation 2* (there were no packets that marked the end of Incarnation 1)
    b. *Incarnation 1 + Incarnation End*, followed by *Incarnation 3* (there were no packets to mark the end of Incarnation 2)

5.3 Packet Header

Payload in UDP packets sent in Incremental or Snapshot lines (including the packets sent by the retransmission service) starts with the Packet Header. All the header values are encoded using the Little-endian byte order.

#	Name	Type	Offset	Length	Description
1	ChannelId	uint8	0	1	Identifier of Market Data Channel distributing data for a group of instruments.
2	Incarnation	uint16	1	2	Incarnation of the Market Data Channel. Incarnation is increased when message sequence is reset – this is accompanied with a message with Incarnation End flag set
3	Source	uint8	3	1	Packet source line type: • 73 ('I') = Incremental Line • 83 ('S') = Snapshot Line • 88 ('X') = Index Snapshot Line
4	Flags	uint8	4	1	Packet flags bit set: • Bit 0 – Incarnation End flag • Bit 1 – Retransmission flag (set when the packet is resent by the retransmission service) • Bit 2 – Administrative message (set when the packet contains an administrative message such as an indication of the retransmission error) Bits 3-7 are reserved.
5	MessageSequence	uint32	5	4	Sequence of the first message in this packet (or in the next packet if this message doesn't contain any messages). This sequence is incremented with any message. First sequence in any incarnation is 1.
6	MessageCount	uint8	9	1	Number of messages in the packet. Can be 0 for heartbeats.

5.4 Heartbeat Packet

When any UDP line has not sent any packets in 5 seconds period, the Exchange will transmit a Heartbeat Packet. The Heartbeat Packet contains the Packet Header with no messages (MessageCount=0).

The packet is useful to identify the line is working and get the next expected message sequence. Note that heartbeats do not increase line message sequence as they contain no messages.

5.5 Incarnation End Packet

When the UDP line message sequence is reset, it is accompanied with the *Incarnation End* flag set in an outgoing packet (see Message Sequencing and Incarnations). In absence of actual market data updates the Exchange might transmit a Heartbeat Packet with the *Incarnation End* flag set. This is an indication that message sequence

is reset. This packet can be published multiple times in a row in the same incarnation. Incarnation End packets do not increase line message sequence.

5.6 Packet Ordering and Delivery Guarantees

Due to the nature of the UDP protocol packets can be reordered or lost in the network. Packet gaps can be detected with the help of *MessageSequence* field in the packet header – the client should track the expected sequence and deal with packet reorderings or losses when sequence mismatch occurs. Possible ways of restoring missed data are via the retransmission service (for the incremental line) or from the next snapshot cycle (for the snapshot line).

In addition, the client should also be ready to deal with duplicate messages. In case of transient failures on the Exchange side messages can be re-published, also their grouping into packets may change. The client should track the expected sequence of the next message (distributed in the *MessageSequence* field in the packet header) and ignore any messages with the sequence that has already been processed. Below is a simple example:

1. A packet with *MessageSequence* = 7 and *MessageCount* = 3 is received.
2. All 3 messages are processed, expected sequence for the next message is 10.
3. A packet with *MessageSequence* = 6 and *MessageCount* = 5 is received.
4. Four messages from this packet are ignored as duplicates (their sequence is 6…9 and the expected sequence is 10).
5. One message from the second packet is processed, expected sequence for the next message is now 11.

5.7 Message Header

Each message in an UDP packet starts with the Message Header containing total message length and standard SBE header fields. Message Header values are encoded using the Little-endian byte order. SBE header fields describe schema and template ID to parse the following message body. See SBE Technical Specification for more details on SBE header fields.

#	Name	SBE Type	Len	Description
1	FrameLength	uint16	2	Total message size in bytes including this header length
	Start of standard SBE header			
2	BlockLength	uint16	2	Total length of the root level of the following message not including any repeating groups or variable-length fields. Block length only represents message body fields; it does not include the length of the message header itself, which is a fixed size
3	TemplateId	uint16	2	Message template identifier. Each distinct message type has its own template identifier.
4	SchemaId	uint16	2	Identifier of message schema containing the template. The following schemas are used: • 1 = Market data schema (incremental, snapshot and index messages) • 2 = Administrative response schema (see Administrative Retransmission Response)
5	Version	uint16	2	Version of message schema. Always 0. This version can be increased in the future if message structure is modified.

6 Schema Extension and Compatibility

The Small Exchange reserves the right to extend Market Data Feed schema with new messages, fields and repeating groups. Market data consumers will be notified in advance about upcoming schema changes. Changes will follow SBE schema extension mechanism for forward compatibility and thus consumers with implementation of an older schema version should be compatible with an updated schema.

Consider the chapter “5 Schema Extension Mechanism” of FIX Simple Binary Encoding Technical Specification

5 Schema Extension Mechanism

5.1 Objective

It is not always practical to update all message publishers and consumers simultaneously. Within certain constraints, message schemas and wire formats can be extended in a controlled way. Consumers using an older version of a schema should be compatible if interpretation of added fields or messages is not required for business processing.

This specification only details compatibility at the presentation layer. It does not relieve application developers of any responsibility for carefully planning a migration strategy and for handling exceptions at the application layer.

5.1.1 Constraints

Compatibility is only ensured under these conditions:

- Fields may be added to either the root of a message or to a repeating group, but in each case, they must be appended to end of a block.
- Existing fields cannot change data type or move within a message.
- A repeating group may be added, but only after existing groups and if there are no subsequent variable data elements at the end of the message.
- A variable data element may be added, but only after existing groups and data.
- Message header encoding cannot change.
- In general, metadata changes such as name or description corrections do not break compatibility so long as wire format does not change.

  Changes that break those constraints require consumers to update to the current schema used by publishers. An message template that has changed in an incompatible way must be assigned a new template "id" attribute.

5.2 Message schema features for extension

5.2.1 Schema version

The `<messageSchema>` root element contains a version number attribute. By default, version is zero, the initial version of a message schema. Each time a message schema is changed, the version number is incremented.

Version applies to the schema as a whole, not to individual elements. Version is sent in the message header so the consumer can determine which version of the message schema was used to encode the message.

See section 4.3.1 above for schema attributes.

5.2.2 Since version

When a new field, enumeration value, group or message is added to a message schema, the extension may be documented by adding a sinceVersion attribute to the element. The sinceVersion attribute tells in which schema version the element was added. This attribute remains the same for that element for the lifetime of the schema. This attribute is for documentation purposes only, it is not sent on the wire.

Over time, multiple extensions may be added to a message schema. New fields must be appended following earlier extensions. By documenting when each element was added, it possible to verify that extensions were appended in proper order.

5.2.3 Block length

The length of the root level of the message may optionally be documented on a `<message>` element in the schema using the blockLength attribute. See section 4.5.3 above for message attributes. If not set in the schema, block length of the message root is the sum of its field lengths. Whether it is set in the schema or not, the block length is sent on the wire to consumers.

Likewise, a repeating group has a blockLength attribute to tell how much space is reserved for group entries, and the value is sent on the wire. It is encoded in the schema as part of the NumInGroup field encoding. See section 3.4.8.2 above.

5.2.4 Deprecated elements

A message schema may document obsolete elements, such as messages, fields, and valid values of enumerations with deprecated attribute. Updated applications should not publish deprecated messages or values, but declarations may remain in the message schema during a staged migration to replacement message layouts.

5.3 Wire format features for extension

5.3.1 Message size

It is assumed that a either message boundaries are delimited by a transport or session protocol header conveys the size of the whole message. See section 3.1 above. This enables a consumer to properly frame messages even when the message has been lengthened in a later version of the schema.

5.3.2 Block size

The length of the root level of the message is sent on the wire in the SBE message header. See section 3.2.2 above. Therefore, if new fields were appended in a later version of the schema, the consumer would still know how many octets to consume to find the next message element, such as repeating group or variable-length Data field. Without the current schema version, the consumer cannot interpret the new fields, but it does not break parsing of earlier fields.

Likewise, block size of a repeating group is conveyed in the NumInGroup encoding.

5.4 Compatibility strategy

This suggested strategy is non-normative.

A message decoder compares the schema version in a received message header to the version that the decoder was built with.

If the *received version is equal to the decoder's version*, then all fields known to the decoder may be parsed, and no further analysis is required.

If the *received version is greater than the decoder's version* (that is, the producer's encoder is newer than the consumer's decoder), then all fields known to the decoder may be parsed but it will be unable to parse added fields.

Also, an old decoder may encounter unexpected enumeration values. The application layer determines whether an unexpected value is a fatal error. Probably so for a required field since the business meaning is unknown, but it may choose to allow an unknown value of an optional field to pass through. For example, if OrdType value J="Market If Touched" is added to a schema, and the consumer does not recognize it, then the application returns an order rejection with reason "order type not supported", even if it does not know what "J" represents. Note that this is not strictly a versioning problem, however. This exception handling is indistinguishable from the case where "J" was never added to the enum but was simply sent in error.

If the *received version is less than the decoder's version* (that is, the producer's encoder is older than the consumer's decoder), then only the fields of the older version may be parsed. This information is available through metadata as "sinceVersion" attribute of a field. If sinceVersion is greater than received schema version, then the field is not available. How a decoder signals an application that a field is unavailable is an implementation detail. One strategy is for an application to provide a default value for unavailable fields.

FIX Simple Binary Encoding Technical Specification by FIX Protocol Ltd. licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License

7 Incremental Line

Incremental Line is the UDP multicast group that receives sequenced incremental messages with order updates, trades and other market information as a result of business event processing by the Matching Engine.

Market data channel has two Incremental Lines (A and B) which carry same packets and messages. Framing and packaging messages logic into UDP packets in A and B lines of a Market Data Channel is the same, thus arbitration of the lines may be performed on a packet or message level. To get the best possible performance the client is expected to connect to both lines, receive the market data messages and use the message that arrived first.

A business event such as new order may produce multiple incremental update messages. To get a consistent market data view the client is expected to apply all these updates atomically at once.

Each message in the Incremental Line contains the following fields:

- **InstrumentMessageNo** - number of messages sent for an instrument. Starts with 1 on instrument creation and increases by 1 with each message sent for the instrument. The field can be used to detect individual instrument message gaps to continue real-time incremental messages processing for instruments w/o gaps detected.
- **IncrementalMessageInstructions** – a set of flags signifying common incremental message properties:
    - Bit 0 = TransactionBegin / Bit 1 = TransactionEnd – can be used to distinguish the first and the last messages resulting a single business event processing.
    - Bit 2 = InstrumentBegin / Bit 3 = InstrumentEnd – are set in the first and the last messages of an instrument within a single transaction. When a message with InstrumentEnd bit is received, there will be no more messages for this instrument in this transaction.
    - Bit 4 = BookBegin / Bit 5 = BookEnd – are set in the first and the last Order Book Incremental messages of an instrument. Can be used to apply all instrument book changes transactionally.
    - Bit 6 = BookReset – the flag marks an instrument order book as empty; the next order book message should be applied to the empty book.

A single business event such as market open may produce multiple incremental messages of different types for multiple instruments. All the messages are published in the following order for each of the instruments:

1. Instrument Definition Incremental Update
2. Trading Status Incremental Update
3. Trades, Trade Corrections and Busts, Order Book Updates
4. Market Summary Incremental Updates

There is no particular order between instruments in updates. Updates related to different instruments may be interleaved. Trades, trade corrections and busts and order book updates for a single instrument can be interleaved arbitrarily however it is guaranteed that any order book modification caused by a trade is published after the trade itself.

7.1 Single Instrument Definition Incremental Message

Sent on added/updated/deleted futures or option instrument on the market.

#	Name	Type	Offset	Length	Description
<Standard Message Header, TemplateId = 1, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved
7	InstrumentUpdateAction	char	25	1	Type of instrument update action • A = instrument is added • D = instrument is deleted • M = instrument is modified
8	Symbol	char	26	20	Instrument symbol
9	Product	char	46	8	Instrument product
10	Description	char	54	120	Instrument text description
11	InstrumentType	char	174	1	Instrument type: • F = futures • O = option
12	MaturityDate	uint16	175	2	Instrument maturity date as number of days since Unix epoch
13	FirstTradingSessionDate	uint16	177	2	Instrument first trading business day as number of days since Unix epoch
14	LastTradingSessionDate	uint16	179	2	Instrument last trading business day as number of days since Unix epoch
15	ExpirationDate	uint16	181	2	Day of instrument final expiration settlement price calculation as number of days since Unix epoch.

16	CfiCode	char	183	6	ISO 10962 instrument 6-character classification code
17	Currency	char	189	3	ISO 4217 instrument currency code. For instance “USD”
18	PriceIncrement	int64	192	8	Instrument price increment. Positive number with 7 implied decimal places. Price increment of 0.01 is represented as 100000 field value. All prices for the instrument order and trade messages are a multiple of the price increment.
19	PriceMultiplier	int64	200	8	Ratio between contract price and its value expressed in instrument's currency amount. Signed value with 7 implied decimal places.
20	PutOrCall	char	208	1	Option put/call flag • P = put • C = call • N = not option (for futures)
21	StrikePrice	int64	209	8	Option strike price. Positive number with 7 implied decimal places. N/A (0x8000000000000000) for futures
22	SharesPerContract	int64	217	8	Option number of shares per contract N/A (0x8000000000000000) for futures

7.2 Multi-Leg Definition Incremental Messages

Sent on added/updated/deleted multi-leg instrument on the market. Currently futures multi-leg instruments with up to 2 legs and ratio from 1 to 5 are supported.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 2, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved
7	InstrumentUpdateAction	char	25	1	Type of instrument update action • A = instrument is added • D = instrument is deleted • M = instrument is modified
8	Symbol	char	26	46	Multi-leg instrument symbol
10	Description	char	72	120	Instrument text description
11	InstrumentType	char	192	1	Always 'M' – multi-leg
12	MaturityDate	uint16	193	2	For multi-leg instruments is usually defined as the nearest maturity date from all legs.
13	FirstTradingSessionDate	uint16	195	2	Instrument first trading business day as number of days since Unix epoch
14	LastTradingSessionDate	uint16	197	2	Instrument last trading business day as number of days since Unix epoch
15	ExpirationDate	uint16	199	2	For multi-leg instruments is usually defined as the nearest expiration date from all legs

16		CfiCode	char	201	6	ISO 10962 instrument 6-character classification code
17		Currency	char	207	3	ISO 4217 instrument currency code. For instance “USD”
18		PriceIncrement	int64	210	8	Instrument price increment. Positive number with 7 implied decimal places. Price increment of 0.01 is represented as 100000 field value. All prices for the instrument order and trade messages are a multiple of the price increment.
19		PriceMultiplier	int64	218	8	Ratio between contract price and its value expressed in instrument's currency amount. Signed value with 7 implied decimal places.
20		StrategyType *since schema version 3*	uint8	226	1	Type of multi-leg strategy: 0 = Custom spread/Unrecognized 1 = Futures calendar, front minus back month 2 = Futures Inter-Commodity Spread, two-legged cross-product futures strategy with up to 5 ratio in legs Reserved values of option strategies for future use, not supported currently: 3 = Covered spread 4 = Option Butterfly 5 = Option Vertical 6 = Option Strangle 7 = Option Straddle 8 = Option Diagonal 9 = Option Calendar
21		NoLegs	uint16	227	2	Legs repeating group block byte length
			uint8	229	1	Number of legs in the repeating group
Begin of legs repeating group						
>	22	LegInstrumentId	int32	0	8	Unique numeric identifier of the leg instrument
>	23	LegSymbol	char	8	20	Leg instrument symbol
>	24	LegProduct	char	28	8	Leg instrument product
>	25	LegRatioQty	uint64	36	8	Leg quantity in a single quantity of the multi-leg instrument
>	26	LegSide	char	44	1	Leg side • B = buy • S = sell
>	27	LegCfiCode	char	45	6	ISO 10962 instrument 6-character classification code
>	28	LegMaturityDate	uint16	51	2	Instrument maturity date as number of days since Unix epoch
>	29	LegStrikePrice	int16	53	8	Option strike price. Positive number with 7 implied decimal places. N/A (0x8000000000000000) for futures
>	30	LegPutOrCall	char	61	1	Option put/call flag

						• P = put • C = call • N = not option (for futures)

7.3 Instrument Trading Status Incremental Message

Sent on updated instrument trading session status individually for every instrument

#	Namee	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 3, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved

7.4 Trades Incremental Message

Represents a set of trades for a single instrument resulting from a business event processing. Note that trades from an event may be sent in separate UDP packets and separate trade messages.

An order book update caused by a trade is guaranteed to be sent after the trade itself, however the system may interleave order book updates and trades arbitrarily.

The message can contain zero or several trades. A message with zero trades can be used to correct last trade price, last trade size, or total volume.

Note that total volume, last traded price, size and time reflect information after last trade in this message. Synthetic trades do not update the last trade information. For these trades LastTradePrice, LastTradeSize, LastTradeTime are not provided and have N/A values.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 4, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved
7	LastTradePrice	int64	25	8	Last traded price for the instrument as of the last trade in this message. Signed number with 7 implied decimal places.
8	LastTradeSize	int64	33	8	Last trade size as of the last trade in this message
9	LastTradeTime	int64	41	8	Timestamp of the last trade occurred for the day as of the last trade in this message
10	TotalVolume	int64	49	8	Updated total day traded volume for the instrument as of the last trade in this message. Single instruments volume is updated with synthetic spread leg fills as well.

11		NoTrades	uint16	57	2	Trades repeating group block byte length
			uint8	59	1	Number of trades in the repeating group
Begin of trades repeating group						
>	11	TradeId	int64	0	8	Unique numeric trade identifier
>	12	Price	int64	8	8	Trade price. Signed number with 7 implied decimal places
>	13	Size	int64	16	8	Trade quantity. Always positive
>	14	AggressorSide	char	24	1	Trade aggressor side • N = no aggressor • B = buy side is a trade aggressor • S = sell side is a trade aggressor
>	15	BuyOrderId	int64	25	8	Id of the 'buy' order. N/A (0x8000000000000000) for a trade that is not related to any orders. Order id here is the same as reported in FIX protocol.
>	16	SellOrderId	int64	32	8	Id of the 'sell' order. N/A (0x8000000000000000) for a trade that is not related to any order. Order id here is the same as reported in FIX protocol.
>	17	TradeConditions	uint16	40	2	Bit set of trade condition flags • Bit 0 = Synthetic – spread leg fill with calculated synthetic price • Bit 1 = Auction – trade is a result of opening auction Bits 2-15 are reserved

7.5 Trade Correct Incremental Message

Represents trade correct message for an instrument. In the repeating group there are always 2 trade update records – data for a busted/deleted trade and data for a new correcting trade.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 5, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved
7	LastTradePrice *since schema version 3*	int64	25	8	Price of the last trade after the correct. Signed number with 7 implied decimal places.
8	LastTradeSize *since schema version 3*	int64	33	8	Size of the last trade after the correct.
9	LastTradeTime *since schema version 3*	int64	41	8	Timestamp of the last trade after the correct
10	TotalVolume *since schema version 3*	int64	49	8	Total day traded volume after the correct
11	NoTrades	uint16	57	2	Trades updates repeating group block byte length
		uint8	59	1	Number of trade updates in the repeating group
Begin of trades update repeating group					
>	12 TradeUpdateAction	char	0	1	Trade update type • D = delete/bust original trade • N = new correcting trade

>	13	TradeId	int64	1	8	Identifier of the original busted trade
>	14	TradeTime	int64	9	8	Trade timestamp
>	15	Price	int64	17	8	Trade price. Signed number with 7 implied decimal places.
>	16	Size	int64	25	8	Trade quantity. Always positive
>	17	AggressorSide	char	33	1	Original trade aggressor side • N = no aggressor • B = buy side is a trade aggressor • S = sell side is a trade aggressor
>	18	BuyOrderId	int64	34	8	Id of the 'buy' order. N/A (0x8000000000000000) for a trade that is not related to any orders. Order id here is the same as reported in FIX protocol.
>	19	SellOrderId	int64	42	8	Id of the 'sell' order. N/A (0x8000000000000000) for a trade that is not related to any orders. Order id here is the same as reported in FIX protocol.
>	20	TradeConditions	uint16	50	2	Bit set of trade condition flags • Bit 0 = Synthetic – spread leg fill with calculated synthetic price • Bit 1 = Auction – trade is a result of opening auction Bits 2-15 are reserved

7.6 Trade Bust Incremental Message

Represents information about one or more busted trades for an instrument.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 6, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused

						• H = halted See Hours of Operation
6		IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd Bits 4-15 are reserved
7		LastTradePrice *since schema version 3*	int64	25	8	Price of the last trade after the bust. Signed number with 7 implied decimal places.
8		LastTradeSize *since schema version 3*	int64	33	8	Size of the last trade after the bust.
9		LastTradeTime *since schema version 3*	int64	41	8	Timestamp of the last trade for the day after the bust.
10		TotalVolume *since schema version 3*	int64	49	8	Total day traded volume after the bust.
11		NoTrades	uint16	57	2	Busted trades repeating group block byte length
			uint8	59	1	Number of busted trades in the repeating group
Begin of busted trades repeating group						
>	12	TradeId	int64	0	8	Identifier of the original busted trade
>	13	TradeTime	int64	8	8	Busted trade timestamp
>	14	Price	int64	16	8	Busted trade price. Signed number with 7 implied decimal places.
>	15	Size	int64	24	8	Busted trade quantity. Always positive
>	16	AggressorSide	char	32	1	Original trade aggressor side • N = no aggressor • B = buy side is a trade aggressor • S = sell side is a trade aggressor
>	17	BuyOrderId	int64	33	8	Id of the 'buy' order. N/A (0x8000000000000000) for a trade that is not related to any orders. Order id here is the same as reported in FIX protocol.
>	18	SellOrderId	int64	41	8	Id of the 'sell' order. N/A (0x8000000000000000) for a trade that is not related to any order. Order id here is the same as reported in FIX protocol.
>	19	TradeConditions	uint16	49	2	Bit set of trade condition flags • Bit 0 = Synthetic – spread leg fill with calculated synthetic price • Bit 1 = Auction – trade is a result of opening auction Bits 2-15 are reserved

7.7 Order Book Incremental Message

The message reflects updated orders in an instrument central limit order book. Note that order updates from a single transaction may be sent in separate UDP packets and separate Order Book Incremental messages.

Order updates are published as they occur with no particular ordering between instruments. Any trade that causes an order book update is published before the order update itself and the order update reflects this trade.

If an order changes its state multiple times during a transaction, all the updates will be published in order of their occurrence.

Note that information on Stop orders waiting for trigger is not published in the protocol. However triggered Stop orders and Market orders (with protection limit price) are published and seen as regular limit orders incremental messages.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 7, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of message for the instrument in a line
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument as of transaction processing end • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd • Bit 4 = BookBegin • Bit 5 = BookEnd • Bit 6 = BookReset Bits 7-15 are reserved.
7	NoOrders	uint16	25	2	Orders repeating group block byte length
		uint8	27	1	Number of orders in the repeating group
Begin of orders repeating group					
> 8	OrderUpdateAction	char	0	1	• N = new order in an order book • U = existing order in a book - price or working quantity is updated • D = existing order is removed from a book due to cancel or fill

>	9	OrderId	int64	1	8	Unique numeric order identifier. Order id here is the same as reported in FIX protocol.
>	10	TradeId	int64	9	8	Id of the trade if this order update is caused by a trade. N/A (0x8000000000000000) if this update is not related to a trade.
>	11	Side	char	17	1	Order side • B = buy • S = sell
>	12	Price	int64	18	8	Order price as a signed number with 7 implied decimal places. For instance, price of 271.82 is represented as 2718200000. For modified order represents a price after modification.
>	13	Size	int64	26	8	Number of order working quantity. Positive or zero for removed orders
>	14	OrderPriority	int64	34	8	Order priority assigned to all orders in a book. This field can be used to compare order priority in a queue across all orders on the same price level – an order with lower field value has higher priority. Implied orders have maximum value of order priority (0x7fffffffffffffff) so that they always have less priority than any direct order.
>	15	OrderAttributes	uint16	42	2	Bit set of order attribute flags • Bit 0 = Implied order Bits 1-15 are reserved

7.8 Market Summary Incremental Message

Reflects trading session updated summary information of an instrument. This message is published only if any of the summary metrics have changed:

- Open Price (and or its Type)
- Close Price
- High
- Low
- Open Interest
- Settlement Price (and or its Type)

The message contains actual values for all of the day summary metrics for an instrument.

Spread leg fill trades with calculated synthetic price do not update Open / Close / High / Low prices of the leg instruments.

Note, that Traded Volume and Last Trade Price for a day are updated and published with Trades Incremental Message

All price fields are signed numbers with 7 implied decimal places and are a multiple of an instrument price increment. Currently supported products on the Exchange have standard increment of 0.01 so all the prices are a multiple of 0.01. For instance, price of 271.82 is represented as 2718200000.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 8, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the instrument
2	InstrumentMessageNo	int64	4	8	Number of incremental message for the instrument. Starts with 1 on instrument creation and monotonically increases with each message sent for the instrument
3	TransactTime	int64	12	8	Message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Message trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	IncrementalMessageInstructions	uint16	23	2	Incremental message handling instructions flags. See details in incremental line description above. • Bit 0 = TransactionBegin • Bit 1 = TransactionEnd • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd

					Bits 4-15 are reserved
7	OpenPrice	int64	25	8	The session opening price. Signed value with 7 implied decimal places. Can be indicative N/A (0x8000000000000000) if there is no traded or indicative open price
8	OpenPriceType	char	33	1	Open price type: • I = indicative price calculated and published every second during the pre-opening auction • T = traded open price as a result of opening after the day pre-opening auction • N = no price
9	HighPrice	int64	34	8	Highest traded price for the session. Signed value with 7 implied decimal places. N/A (0x8000000000000000) if no trades occurred for the day
10	LowPrice	int64	42	8	Lowest trade price for the summary date. N/A (0x8000000000000000) if no trades occurred for the day
11	ClosePrice	int64	50	8	Closing price for the summary date. N/A (0x8000000000000000) if no trades occurred for the day
12	OpenInterest	int64	58	8	Contract open interest. Total number of long (and equally short) contract positions. Published for single instruments only as latest known value calculated by a clearing organization. N/A (0x8000000000000000) for multi-leg instruments
13	SettlementPrice	int64	64	8	Settlement price for the session day. Signed value with 7 implied decimal places. Published for single instruments only. N/A (0x8000000000000000) for multi-leg, or if not calculated yet
14	SettlementPriceType	char	72	1	Settlement price type: • F = final settlement price for a day, published within 15 mins after instrument market is closed for a day. See Hours of Operation. • P = preliminary. Reserved for future use, not published. • N = no price. For multi-leg, or if not calculated yet

8 Snapshot Line

The snapshot line is the UDP multicast group that periodically distributes instrument market data snapshots to be used for recovery purposes and synchronizing with the Incremental Line.

UDP packets with instrument definitions, order book and summary snapshots for all instruments in the Market Data Channel are replayed in a line periodically in cycles, each cycle consists of messages for each instrument in the following order:

1. One Instrument Definition message (single or multi-leg)
2. Order Book snapshot messages – one or more messages to reflect the full snapshot of the instrument order book as of instrument message number sent in the incremental line of the market data channel.
3. One previous trading day summary snapshot message
4. One current trading day summary snapshot message

Total number of instruments in a cycle is provided in *SnapshotInstrumentCount* field of each snapshot message. Within a single cycle the number of instruments may increase but can never decrease.

ChannelID and *Incarnation* in the line packet headers correspond to *ChannelID* and *Incarnation* of an incremental line.

Each message in the Snapshot Line contains the following common fields:

- **InstrumentMessageNo** - Number of an instrument incremental message the snapshot message reflects the instrument data for. The field is to be used to synchronize received instrument snapshot data with the Incremental Line instrument messages.
- **SnapshotMessageInstructions** – a set of instruction handling flags
    - Bit 7 = SnapshotBegin / Bit 8 = SnapshotEnd – mark the first and the last messages of a single snapshot replay in a line
    - Bit 2 = InstrumentBegin / Bit 3 = InstrumentEnd – mark the first and the last messages of an instrument within a snapshot replay
    - Bit 4 = BookBegin / Bit 5 = BookEnd – are set in the first and the last Order Book Snapshot messages of an instrument
- **LastIncrementalMessageSeq** – last incremental message sequence in this incarnation related to this instrument (see Message Sequencing and Incarnations). This field can be used to simplify initial session establishment. See Session Establishment and Maintenance Algorithm.

8.1 Single Instrument Definition Snapshot Message

Single futures or option instrument definition.

#	Namee	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 9, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of the instrument incremental message the snapshot reflects the instrument data for. Any incremental update with *InstrumentMessageNo* less than or equal to this value are guaranteed to be incorporated in this snapshot.
3	TransactTime	int64	12	8	Last instrument incremental message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Last instrument incremental message business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N =pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	SnapshotMessageInstructions	uint16	23	2	Bit set of snapshot line message flags • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd • Bit 7 = SnapshotBegin • Bit 8 = SnapshotEnd
7	SnapshotInstrumentsCount	uint32	25	4	Total number of instruments in the Snapshot line publishing cycle. The number may increase within a single publishing cycle in case any instruments are created
8	LastIncrementalMessageSeq	int64	29	8	Last incremental message sequence in this incarnation related to this instrument (see Message Sequencing and Incarnations)
9	Symbol	char	37	20	Instrument symbol
10	Product	char	57	8	Instrument product
11	Description	char	65	120	Instrument text description
12	InstrumentType	char	185	1	Instrument type: • F = futures • O = option
13	MaturityDate	uint16	186	2	For multi leg instruments is usually defined as nearest maturity date from all legs.
14	FirstTradingSessionDate	uint16	188	2	Instrument first trading business day as number of days since Unix epoch
15	LastTradingSessionDate	uint16	190	2	Instrument last trading business day as number of days since Unix epoch

16	ExpirationDate	uint16	192	2	For multi leg instruments is usually defined as nearest expiration date from all legs.
17	CfiCode	char	194	6	ISO 10962 instrument 6-character classification code
18	Currency	char	200	3	ISO 4217 instrument currency code. For instance “USD”
19	PriceIncrement	int64	203	8	Instrument price increment. Positive number with 7 implied decimal places. Price increment of 0.01 is represented as 100000 field value.
20	PriceMultiplier	int64	211	8	Ratio between contract price and its value expressed in instrument's currency amount. Signed value with 7 implied decimal places.
21	PutOrCall	char	219	1	Option put/call flag • P = put • C = call • N = not an option (for futures)
22	StrikePrice	int64	220	8	Option strike price. Signed value with 7 implied decimal places. N/A (0x8000000000000000) for futures
23	SharesPerContract	int64	228	8	Option number of shares per contract N/A (0x8000000000000000) for futures

8.2 Multi-Leg Instrument Definition Snapshot Message

Multi-leg instrument definition.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 10, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of the instrument incremental message the snapshot reflects the instrument data for. Any incremental update with *InstrumentMessageNo* less than or equal to this value are guaranteed to be incorporated in this snapshot.
3	TransactTime	int64	12	8	Last instrument incremental message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Last instrument incremental message business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument • C = closed • P = pre-open • N =pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	SnapshotMessageInstructions	uint16	23	2	Bit set of snapshot line message flags • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd • Bit 7 = SnapshotBegin • Bit 8 = SnapshotEnd
7	SnapshotInstrumentsCount	uint32	25	4	Total number of instruments in the Snapshot line publishing cycle. The number may increase within a single publishing cycle in case any instruments are created
8	LastIncrementalMessageSeq	int64	29	8	Last incremental message sequence in this incarnation related to this instrument (see Message Sequencing and Incarnations)
9	Symbol	char	37	46	Instrument symbol
10	Description	char	83	120	Instrument text description
11	InstrumentType	char	203	1	Always M = multi-leg
12	MaturityDate	uint16	204	2	Instrument maturity date as number of days since Unix epoch
13	FirstTradingSessionDate	uint16	206	2	Instrument first trading business day as number of days since Unix epoch
14	LastTradingSessionDate	uint16	208	2	Instrument last trading business day as number of days since Unix epoch
15	ExpirationDate	uint16	210	2	Instrument expiration day as number of days since Unix epoch

16		CfiCode	char	212	6	ISO 10962 instrument 6-character classification code
17		Currency	char	218	3	ISO 4217 instrument currency code. For instance “USD”
18		PriceIncrement	int64	221	8	Instrument price increment. Positive number with 7 implied decimal places. Price increment of 0.01 is represented as 100000 field value. All prices for the instrument order and trade messages are a multiple of the price increment.
19		PriceMultiplier	int64	229	8	Ratio between contract price and its value expressed in instrument's currency amount. Signed value with 7 implied decimal places.
20		StrategyType *since schema version 3*	uint8	237	1	Type of multi-leg strategy: • 0 = Custom spread/Unrecognized • 1 = Futures calendar, front minus back month • 2 = Futures Inter-Commodity Spread, two-legged cross-product strategy with up to 5 ratio in legs Reserved values of option strategies for future use, not supported currently: • 3 = Covered spread • 4 = Option Butterfly • 5 = Option Vertical • 6 = Option Strangle • 7 = Option Straddle • 8 = Option Diagonal • 9 = Option Calendar
21		NoLegs	uint16	238	2	Legs repeating group block byte length
			uint8	240	1	Number of legs in the repeating group
Begin of legs repeating group						
>	22	LegInstrumentId	int32	0	8	Unique numeric identifier of the leg instrument
>	23	LegSymbol	char	8	20	Leg instrument symbol
>	24	LegProduct	char	28	8	Leg instrument product
>	25	LegRatioQty	uint64	36	8	Leg quantity in a single quantity of the multi-leg instrument
>	26	LegSide	char	44	1	Leg side • B = buy • S = sell
>	27	LegCfiCode	char	45	6	ISO 10962 instrument 6-character classification code
>	28	LegMaturityDate	uint16	51	2	Instrument maturity date as number of days since Unix epoch
>	29	LegStrikePrice	int16	53	8	Option strike price. Signed value with 7 implied decimal places. N/A (0x8000000000000000) for futures
>	30	LegPutOrCall	char	61	1	Option put/call flag • P = put

						• C = call • N = not option, used for futures

8.3 Order Book Snapshot Message

Reflects a snapshot of instrument order book. Empty order book snapshot is published having no orders (zero repeating group size) with BookBegin and BookEnd instruction flags. Instruments with the first trading day in the future are published with empty order book snapshot.

#	Name	Type	Offset	Length	Description
<Standard Message Headers*, TemplateId = 11, SchemaId = 1*>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of the instrument incremental message the snapshot reflects the instrument data for. Any incremental update with *InstrumentMessageNo* less than or equal to this value are guaranteed to be incorporated in this snapshot.
3	TransactTime	int64	12	8	Last instrument incremental message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Instrument trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument as of the incremental message number: • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	SnapshotMessageInstructions	uint16	23	2	Bit set of snapshot line message flags • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd • Bit 4 = BookBegin • Bit 5 = BookEnd • Bit 7 = SnapshotBegin • Bit 8 = SnapshotEnd
7	SnapshotInstrumentsCount	uint32	25	4	Total number of instruments in the Snapshot line publishing cycle. The number may increase within a single publishing cycle in case any instruments are created
8	LastIncrementalMessageSeq	int64	29	8	Last incremental message sequence in this incarnation related to this instrument (see Message Sequencing and Incarnations)

9		NoOrders	uint16	37	2	Orders repeating group block byte length
			uint8	39	1	Number of orders in the repeating group. May be zero if book is empty.
Begin of orders repeating group						
>	10	OrderId	int64	0	8	Unique numeric order identifier. Order id here is the same as reported in FIX protocol.
>	11	Side	char	8	1	Order side • B = buy • S = sell
>	12	Price	int64	9	8	Order price as signed number with 7 implied decimal places. For instance price of 271.82 is represented by 2718200000 field value.
>	13	Size	int64	17	8	Signed number of order quantity
>	14	OrderPriority	int64	25	8	Order priority assigned to all orders in a book. This field can be used to compare order priority in a queue across all orders on the same price level – an order with lower field value has higher priority. Implied orders have maximum value of order priority (0x7fffffffffffffff) so that they always have less priority than any direct order.
>	15	OrderAttributes	uint16	33	2	Bit set of order attribute flags • Bit 0 = Implied order Bits 1-15 are reserved
>	16	OrderTime *since schema version 3*	int64	35	8	Order timestamp as number of nanoseconds since Unix epoch

8.4 Market Summary Snapshot Message

All price fields are signed numbers with 7 implied decimal places and are a multiple of an instrument price increment.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 12, SchemaId = 1>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the event instrument
2	InstrumentMessageNo	int64	4	8	Number of the instrument incremental message the snapshot reflects the instrument data for. Any incremental update with *InstrumentMessageNo* less than or equal to this value are guaranteed to be incorporated in this snapshot.
3	TransactTime	int64	12	8	Last instrument incremental message event timestamp as number of nanoseconds since Unix epoch
4	TradingSessionDate	uint16	20	2	Instrument trading session business date as number of days since Unix epoch
5	InstrumentTradingStatus	char	22	1	Trading session status of the instrument: • C = closed • P = pre-open • N = pre-open no cancel • O = open • U = paused • H = halted See Hours of Operation
6	SnapshotMessageInstructions	uint16	23	2	Bit set of snapshot line message flags • Bit 2 = InstrumentBegin • Bit 3 = InstrumentEnd • Bit 7 = SnapshotBegin • Bit 8 = SnapshotEnd
7	SnapshotInstrumentsCount	uint32	25	4	Total number of instruments in the Snapshot line publishing cycle. The number may increase within a single publishing cycle in case any instruments are created
8	LastIncrementalMessageSeq	int64	29	8	Last incremental message sequence in this incarnation related to this instrument (see Message Sequencing and Incarnations)
9	LastTradePrice	int64	37	8	Price of the session last traded price. Signed value with 7 implied decimal places. N/A (0x8000000000000000) if no trades occurred for the day
10	LastTradeSize	int64	45	8	Size of the session last trade. N/A (0x8000000000000000) if no trades occurred for the day
11	LastTradeTime	int64	53	8	Timestamp of the session last trade. N/A (0x8000000000000000) if no trades occurred for the day

12	TotalVolume	int64	61	8	Total traded volume for the instrument session. N/A (0x8000000000000000) if no trades occurred for the day
13	OpenPrice	int64	69	8	The session opening price. Signed value with 7 implied decimal places. N/A (0x8000000000000000) if the market doesn't have indicative or traded open price
14	OpenPriceType	char	77	1	Open price type: • I = indicative price calculated and published every second during the pre-opening auction • T = traded open price as a result of opening after the day pre-opening auction • N = no price (if the market doesn't have indicative of traded open price)
15	HighPrice	int64	78	8	Highest traded price for the session. Signed value with 7 implied decimal places N/A (0x8000000000000000) if no trades occurred for the day
16	LowPrice	int64	86	8	Lowest trade price for the summary date. Signed value with 7 implied decimal places N/A (0x8000000000000000) if no trades occurred for the day
17	ClosePrice	int64	94	8	Closing price for the summary date. Signed value with 7 implied decimal places N/A (0x8000000000000000) if the market is not closed
18	OpenInterest	int64	102	8	Contract open interest. Total number of long (and equally short) contract positions. Published for single instruments only. N/A (0x8000000000000000) for multi-leg instruments
19	SettlementPrice	int64	110	8	Settlement price for the session day. Signed value with 7 implied decimal places. Published for single instruments only. N/A (0x8000000000000000) for multi-leg instrument or if not calculated yet
20	SettlementPriceType	char	118	1	Settlement price type: • F = final settlement price for the day, published within 15 mins after instrument market is closed for the day. See Hours of Operation • P = preliminary. Reserved for future use, not published.

					• N = no price, for multi-leg or if not calculated yet.

9 Index Line

Index Line is the UDP multicast group that periodically distributes current index values. Each cycle contains current data for all the indices calculated by the Exchange. Order of index value messages might change between subsequent cycles.

Each message in the Index Line contains the following common fields:

- **SnapshotMessageInstructions** – a set of instruction handling flags
  - Bit 7 = SnapshotBegin / Bit 8 = SnapshotEnd – mark the first and the last messages of a single snapshot replay in a line
- **IndexCount** – total number of indices in the snapshot cycle

Indices have their own schedule and symbology:

Index	Symbol	Schedule
The Small Dollar Index	FXSME	6:00 AM – 5:00 PM America/Chicago
The Small Stocks 75 Index	75SME	8:30 AM – 3:00 PM America/Chicago
The Small Global Oil Index	GOSME	6:00 AM – 5:00 PM America/Chicago
The Small 10 Year U.S. Treasury Yield Index	10YSME	6:00 AM – 5:00 PM America/Chicago
The Small Precious Metals Index	PRESME	6:00 AM – 5:00 PM America/Chicago

9.1 Index Snapshot Message

This message contains the last calculated index value and the most recent Open-High-Low-Close values. If the index is not being calculated for any reason, the message will contain the last previously determined value.

Currently supported indices have standard increment of 0.01 so all the prices are a multiple of 0.01. For instance, price of 271.82 is represented as 2718200000.

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = XXX, SchemaId = 3, sinceVersion=2>					
1	InstrumentId	int32	0	4	Unique numeric identifier of the index as assigned by the Exchange
2	TransactTime	int64	4	8	Index value calculation time as number of nanoseconds since Unix epoch
3	SnapshotMessageInstructions	uint16	12	2	Bit set of snapshot line message flags • Bit 7 = SnapshotBegin • Bit 8 = SnapshotEnd
4	IndexCount	uint32	14	4	Total number of indices in the line publishing cycle
5	Symbol	char	18	20	Index symbol
6	Value	int64	38	8	Last index value. Signed with 7 implied decimal places.
7	SessionDate *since schema version 4*	uint16	46	2	Index session date as number of days since Unix epoch
8	OpenPrice *since schema version 4*	int64	48	8	First index value calculated for a day. Signed value with 7 implied decimal places.

					N/A (0x8000000000000000) if index calculation is not started for the current day
9	HighPrice *since schema version 4*	int64	56	8	Highest index value for the day. Signed value with 7 implied decimal places. N/A (0x8000000000000000) if index calculation is not started for the current day
10	LowPrice *since schema version 4*	int64	64	8	Lowest index value for the day. Signed value with 7 implied decimal places N/A (0x8000000000000000) if index calculation is not started for the current day
11	ClosePrice *since schema version 4*	int64	72	8	Last index value for the day. Signed value with 7 implied decimal places. N/A (0x8000000000000000) if the day is not closed.

10 Retransmission Service

In case message sequence gap is detected in the Incremental Line the Retransmission Service allows to re-request lost messages for a range of message sequences.

Please note the service limitation: number of messages that can be returned in a response to a single request is limited by UDP packet size, 1300 bytes for payload. The service will return as many messages as can fit into a single packet starting from the requested sequence.

If the service cannot fulfill the request, an administrative message is sent in response.

It is recommended to use the Snapshot Line for recovery purposes in a huge message sequence gap is detected. Recovering a large gap via the retransmission service might be slower, in addition the service has rate limit quotas. As a rule of thumb, it is advised to use the Snapshot Line when the sequence gap is over 2500, however this may vary depending on the market activity.

10.1 Retransmission Request

The request is a unicast UDP packet containing a range of incremental messages to be resent.

#	Name	Type	Offset	Length	Description
1	ChannelId	uint8	0	1	Identifier of Market Data Channel distributing data for a group of instruments. This is the channel to receive missing messages for.
2	Incarnation	uint16	1	2	Incarnation of the Market Data Channel. This is the requested incarnation.
3	Source	uint8	3	1	Packet source line type. Only 73 ('I') = Incremental Line value is supported.
4	RequestedMessage Sequence	uint32	4	4	Sequence of the first requested message (within the incarnation).
5	RequestedMessage Count	uint8	8	1	Number of requested messages.

10.2 Retransmission Response

Retransmission response is a unicast UDP message sent to the requestor. The response packet shares the same packet header and packet structure as the outgoing incremental and snapshot packets (see Packet Header and Packet Structure).

#	Name	Type	Offset	Length	Description
1	ChannelId	uint8	0	1	Identifier of Market Data Channel distributing data for a group of instruments. Copied from the request.
2	Incarnation	uint16	1	2	Incarnation of the Market Data Channel. Copied from the request.
3	Source	uint8	3	1	Packet source line type. Only 73 ('I') = Incremental Line value is supported.
4	Flags	uint8	4	1	Packet flags bits: • Bit 0 – Incarnation End flag (see Incarnation End Packet) • Bit 1 – Retransmission flag (always set) • Bit 2 – Administrative message (not set)

					Bits 3-7 are reserved.
5	MessageSequence	uint32	5	4	Sequence of the first requested message (within the incarnation).
6	MessageCount	uint8	9	1	Number of messages in this response. This number can be smaller than the requested count if the requested count cannot fit a single UDP datagram or if one of the requested messages ends the incarnation
<Resent Incremental Line Messages>					

10.3 Administrative Retransmission Response

Administrative response is sent by the retransmission service if the request cannot be fulfilled. This response packet shares the same packet header and packet structure as the outgoing incremental and snapshot packets (see Packet Header and Packet Structure).

The service can fail to resend messages for various reasons, such as:

- Incorrect request (incorrectly defined channel or source etc.)
- Unknown message sequence or incarnation
- The client makes too many requests to the service and exceeds its request quota
- The client attempts to request messages that are too far in the past – the service is intended to be a tool to quickly repair the gaps and not to be a full-fledged historical market data service

The client can use the 'Administrative message' flag to distinguish an administrative response from the regular one. In addition, the administrative response always has zero (0) in the *MessageSequence* field.

#	Name	Type	Offset	Length	Description
1	ChannelId	uint8	0	1	Identifier of Market Data Channel distributing data for a group of instruments. Copied from the request.
2	Incarnation	uint16	1	2	Incarnation of the Market Data Channel. Copied from the request.
3	Source	uint8	3	1	Packet source line type. Copied from the request.
4	Flags	uint8	4	1	Packet flags bits: • Bit 0 – Incarnation End flag (not set) • Bit 1 – Retransmission flag (always set) • Bit 2 – Administrative message (always set) Bits 3-7 are reserved.
5	MessageSequence	uint32	5	4	Always 0. This indicates that this response contains no resent messages from the incremental line.
6	MessageCount	uint8	9	1	Number of administrative messages in this response.
<Administrative Messages>					

Administrative response message is an SBE message (see Message Header) with the following data:

#	Name	Type	Offset	Length	Description
<Standard Message Headers, TemplateId = 1, SchemaId = 2>					
1	RequestedMessageSequence	uint32	0	4	Starting message sequence requested (copied from the Retransmission Request)
2	RequestedMessageCount	uint32	4	4	Number of requested messages (copied from the Retransmission Request)

3	ResponseCode	uint8	8	1	Response code, one of: • 1 = UNKNOWN_SOURCE • 2 = REQUESTED_SEQ_TOO_LOW • 3 = REQUESTED _SEQ_TOO_HIGH • 4 = RATE_LIMIT_VIOLATION • 5 = REQUEST_DROP (see details below)
4	RateLimitTimeout	int64	9	8	If ResponseCode is RATE_LIMIT_VIOLATION, this field contains the number of nanoseconds to wait before sending a new request
<Variable-length Description Composite>					
5	DescriptionLength	uint16	17	2	Length of the description field
6	Description	char	19	variable	Human-readable description of the response

Response code can have one of the following values:

Code	Details
1 = UNKNOWN_SOURCE	Retransmission service is not aware of the requested Channel / Source / Incarnation. The client should either use another service instance or correct the request.
2 = REQUESTED_SEQ_TOO_LOW	Requested message sequence is outside the range that can be returned by the service – the sequence is too low. The packets cannot be retrieved from the retransmission service and the client is expected to re-start the market data session (see Session Establishment and Maintenance Algorithm).
3 = REQUESTED_SEQ_TOO_HIGH	Requested message sequence is outside the range that can be returned by the service – the sequence is too high. The service is not aware of the requested messages yet. The client can retry the request later.
4 = RATE_LIMIT_VIOLATION	The client requests packet retransmission too frequently. Request will not be processed. The client can retry the request later.
5 = REQUEST_DROP	The service discarded one or more pending requests from the client due to overload. The client can retry the request later.

11 Session Establishment and Maintenance Algorithm

The following simple algorithm can be used by market data clients to connect to the Exchange initially, build the initial market state from scratch and start receiving consistent updates for any instrument:

1. Connect to the incremental line, start buffering the incoming packets
    a. Receive the first incremental packet (could be a heartbeat), memorize the first message sequence
2. If any gaps are detected, re-request the missing data from the retransmission service and fill the gaps
3. Connect to the snapshot line and wait until a full snapshot cycle with the same incarnation as messages from incremental line completes (thus building the initial state of instrument records, order books and summary information)
    a. Disconnect from the snapshot line when you receive all the books with LastIncrementalSeqNo greater than or equal to the first message sequence seen in the incremental line
4. For each instrument of interest discard all the buffered incremental updates that have *InstrumentMessageNo* less than or equal to the *InstrumentMessageNo* from the snapshot line
5. Apply buffered incremental updates to the market state and discard the buffer
6. Apply new incoming incremental updates to the market state as they arrive

During regular operation it is recommended to use the Snapshot Line for recovery purposes in a huge message sequence gap is detected. Recovering a large gap via the retransmission service might be slower, in addition the service has rate limit quotas. As a rule of thumb, it is advised to use the Snapshot Line when the sequence gap is over 2500, however this may vary depending on the market activity.

12 Hours of Operation

The Small Exchange Order Book Feed starts distributing data at 5:00am and shuts down at 6:00pm America/Chicago.

Below is a typical trading day schedule for all supported futures instruments on the Small Exchange. All times specified below are in America/Chicago time zone.

Please note the special schedule for each third Friday of a month: all expiring contracts markets are closed at 3pm (instead of 4pm) for the day. Final expiration settlement price is published for the contracts within the 3:00pm – 3:15pm period. Schedule of other contracts expiring in the future is not affected and their markets are closed at 4pm.

<table>
<tr><th>Hours</th><th>Session Status</th><th>Incremental Lines</th><th>Snapshot Lines</th></tr>
<tr><td>Feed shutdown</td><td></td><td colspan="2">Lines are inactive</td></tr>
<tr><td>5:00am – 6:30am</td><td>Closed</td><td>Single Instrument Incremental and Multi-leg Instrument incremental may be sent to add new or delete expired instruments</td><td rowspan="6">Current state of all channel’s instruments order books and summary are replayed periodically in cycles:
- Single Instrument Snapshot
- Multi-leg Instrument Snapshot
- Order Book Snapshot
- Summary Snapshot</td></tr>
<tr><td>6:30am – 6:59am</td><td>Pre-open</td><td rowspan="2">Opening auction, buy/sell order books may cross.

All types of incremental business messages are sent except Trades Incremental.</td></tr>
<tr><td>6:59am – 7:00am</td><td>Pre-open no cancel</td></tr>
<tr><td>7:00am – 4:00pm</td><td>Open</td><td>Regular trading session for all product instruments.

All types of incremental business messages are sent</td></tr>
<tr><td>* 3rd Friday Expiration

3:00pm – 3:15pm

Closed</td><td>* Closed for expiring futures</td><td>On the third Friday of the month market is closed for expiring futures and final expiration settlement prices are published for the contracts. All other contracts continue to trade till 4pm.</td></tr>
<tr><td>4:00pm – 4:15pm

Closed</td><td>Closed</td><td>Market is closed.

Order Book Incremental are sent for expired DAY and GTC orders.</td></tr>
</table>

Hours	Session Status	Incremental Lines	Snapshot Lines
		Trade Bust and Correct messages may be sent for a day trades. Final settlement prices are published for a day.	
4:15pm – 6:00pm Closed	Closed	Single Instrument Incremental and Multi-leg Instrument Incremental may be sent for adding new or deleting expired instruments. After all business messages Incarnation End packets may be sent to reset message and instrument sequences	
Feed shutdown		Lines are inactive	