MARKET DATA B3: BINARY UMDF

Messaging Specification Guidelines – Version 1.9.0.4

Last modified: November 13th, 2024

CONTACTS

- Services Development Department: handles all enquiries for connectivity setup and general exchange supported services.
  - contratacao@b3.com.br
  - +55 11 2565-5081
- Certification and Testing Center: performs certification of all software solutions applying for Order Entry and Market Data connectivity.
  - tradingcertification@b3.com.br
  - +55 11 2565-5029
- Trading Support Department (TSG): provides real time connectivity monitoring and troubleshooting.
  - tradingsupport@b3.com.br
  - +55 11 2565-5021

1. CHANGE LOG 7

2. PREFACE 10

2.1 ABBREVIATIONS 11
2.2 GLOSSARY 11

3. TRADING HOURS 13

3.1 TRADING SESSION HOURS 13
3.2 EXCHANGE HOLIDAYS 13

4. FEATURES IN BINARY UMDF 13

4.1 SIMPLE BINARY ENCODING 14
4.2 TRADING PLATFORM EVENTS 14
4.2.1 ORDER ACTION-DRIVEN EVENTS 14
4.2.2 MARKET DATA MESSAGES THAT DO NOT BELONG TO AN EVENT 14
4.2.3 EVENTS THAT DO NOT GENERATE MARKET DATA 15
4.3 TRACKING OF BOOK UPDATES 15

5. IMPLEMENTING SBE 16

5.1 SBE DESIGN PRINCIPLES 16
5.2 SBE SPECIFICATION 16
5.3 SBE FIELD ORDER AND SPEED OF ACCESS 16
5.3.1 ALIGNMENT AND PADDING 17
5.3.2 SIMILAR MESSAGES WITH COMMON FIELDS 17
5.4 NESTED FIX MESSAGES VS FLATTENED SBE MESSAGES 17
5.5 SBE, OPTIONAL FIELDS AND DEFAULT VALUES, EMPTY FIELDS 18
5.5.1 DECIMAL 'NULL' VALUE 20
5.6 SBE TEMPLATES 20
5.7 FIX MESSAGE AND SBE TEMPLATES 21
5.8 LIMITING THE SBE MESSAGE SIZE 21
5.9 SCHEMA EXTENSION MECHANISM 21
5.9.1 OBJECTIVE 21

5.9.2 MESSAGE SCHEMA FEATURES FOR EXTENSION 22
5.9.3 WIRE FORMAT FEATURES FOR EXTENSION 23
5.10 COMPATIBILITY STRATEGY TO HANDLE SCHEMA EVOLUTION 24
5.10.1 INCREMENT SCHEMA VERSION AND CHANGES ARE MADE FOR SOME TEMPLATES 24
5.10.2 SCHEMA VERSION IS INCREMENTED, NEW TEMPLATE IS CREATED FOR EXISTING MESSAGE 25

6. SYSTEM ARCHITECTURE 27

6.1 MARKET DATA CHANNEL 27
6.2 INCREMENTAL STREAM 28
6.3 SNAPSHOT RECOVERY STREAM 29
6.4 INSTRUMENT DEFINITION STREAM 30
6.5 ENGAGEMENT RULES 30
6.5.1 LATEST SBE TEMPLATES 30
6.5.2 NETWORK CONFIGURATION 30
6.5.3 MARKET DATA NETWORK CONTINGENCY FEED 31
6.5.4 MESSAGE FRAMING 32
6.5.5 PACKET STRUCTURE 32
6.5.6 EXAMPLES OF SBE MESSAGE ENCODING 35
6.5.7 MESSAGES, PACKETS AND EVENTS 38
6.5.8 INSTRUMENT DEFINITION STREAM PROCESSING 38
6.5.9 INITIAL MARKET DATA SYNCHRONIZATION PROCEDURE 40
6.5.10 START OF DAY “HEARTBEATS” – SEQUENCE MESSAGES 42
6.5.11 STREAM RESET MESSAGE 43
6.5.12 CHANNEL RESET 44
6.5.13 EMPTYBOOK (BOOK RESET) 47

7. RECOVERY 48

7.1 SNAPSHOT MESSAGES 48
7.2 SNAPSHOTREFRESH_HEADER 50
7.3 SNAPSHOTREFRESH_ORDERS 51
7.4 SNAPSHOT RECOVERY 53
7.4.1 USING *RPTSEQ* AND *LASTRPTSEQ* FOR SYNCHRONIZATION 57

7.5 SEQUENCE MESSAGE 61

8. MARKET DATA ENTRY TYPES 62

9. TRADING EVENT PROCESSING – *MATCHEVENTINDICATOR* 65

10. ORDER BOOK 67

10.1 BOOK: POSITION NUMBER 67

11. INCREMENTAL ORDER BOOK MANAGEMENT 68

11.1 INCREMENTAL BOOK MANAGEMENT–- MBO 68

11.1.1 ORDER DEPTH BOOK 69

11.1.2 DELETE FROM 71

11.1.3 DELETE THRU 72

12. TRADE AND REAL-TIME STATISTICAL DATA 73

12.1 EXECUTIONSUMMARY 74

12.1.1 MATCHING DISCLOSED QUANTITY ORDERS (ICEBERG ORDERS) 76

12.1.2 TRIGGERING SELF-TRADE PREVENTION DURING A MATCHING 77

12.1.3 TRIGGERING SEVERAL STOP ORDERS/TRADES IN A SINGLE MATCHING EVENT 79

12.2 EXECUTIONSTATISTICS 81

12.3 TRADE 82

12.3.1 TRADECONDITION 83

12.3.2 TRDSUBTYPE 83

12.3.3 RELATIONSHIP BETWEEN TRADECONDITION AND TRDSUBTYPE 83

12.4 TRADE BUST 84

12.5 FORWARD TRADE 85

12.6 LAST TRADE PRICE 85

12.7 TRADING SESSION HIGH/LOW PRICE 86

12.8 CALCULATION OF TRADING SESSION VWAP PRICE 87

12.9 OPENING PRICE / THEORETICAL OPENING PRICE / CLOSING PRICE 87

12.9.1 OPENINGPRICE MESSAGE 87

12.9.2 THEORETICALOPENINGPRICE MESSAGE 88

12.9.3 CLOSINGPRICE MESSAGE 88
12.10 AUCTION IMBALANCE 89
12.10.1 *AUCTIONIMBALANCE* MESSAGE 89
12.11 PRICE AND QUANTITY BANDS INFORMATION 89
12.11.1 PRICEBAND MESSAGE 91
12.11.2 QUANTITYBAND MESSAGE 92
12.11.3 SETTLEMENTPRICE MESSAGE 92
12.11.4 OPENINTEREST MESSAGE 94
12.11.5 SPECIFIC USAGE FOR EACH TYPE OF BAND AND TUNNEL 95

13. GROUP PHASE/INSTRUMENT STATE INFORMATION 96

13.1 INSTRUMENT STATES 98
13.2 TRADING PHASES 99
13.3 TRADING STATISTICS RESET 100
13.4 GROUP PHASE AND INSTRUMENT STATE IN THE SNAPSHOT STREAM 100

14. DERIVATIVES/FX SPECIFIC MARKET DATA FUNCTIONALITY 100

14.1 OPTION STRIKE PRICE 100

15. TRADE VOLUME, VWAP AND NUMBER OF TRADES 101

16. IMPLIEDS 101

16.1 IMPLIED IN ORDERS 102
16.2 IMPLIED OUT ORDERS 103
16.3 MATCH THAT INVOLVES AN IMPLIED ORDER 103

17. MISCELLANEOUS REMARKS 105

1. Change Log

Date	Version	Description	Author
Mar. 31st, 2020	1.0	- Initial version.	AYSF, RDRC
Apr. 29th, 2020	1.1	- Including the version 1.0 of SBE Market Data specification.	AYSF, EEW, RDRC
May 12th, 2020	1.2	- Clarifying snapshot, book reset, trade summary, including diagrams, including the SBE message names.	AYSF, EEW, RDRC
Jun. 25th, 2020	1.3	- A diagram for the handling of Order Book reset was reincluded. - Included a message for 269=c and other for Last Trade Price. - Clarifying the handling of null values by SBE. - Order Priority is also available for MBO diffusion. - Template version: 1.1	EEW
Feb. 16th, 2022	1.3	- MBO only; removed several messages. - Template version: 1.3	EEW, RNKH
Mar. 10th, 2022	1.3.1	- Revised text. Template version: 1.3.1.	EEW, RNKH, ANJ
Apr. 6th, 2022	1.3.2	- Minor text revisions due to feedback. - Fixed message examples.	EEW, RNKH
Apr. 29th, 2022	1.3.3	- Minor text revisions due to customer and developer feedback.	EEW, RKNH
May 30th, 2022	1.4.0	- Inclusion of *ExecutionSummary*, *ExecutionStatistics* messages, due to customer and developer feedback.	EEW, RKNH
Jun. 21st, 2022	1.4.1	- More clarifications and minor text revisions due to developer feedback. - Fixed message examples.	EEW, RNKH
Jul. 7th, 2022	1.5.0	- New messages (TradeBust, ChannelReset). - New fields: RptSeq, LastRptSeq, NumberOfTrades. - Fields renamed: OrderID → SecondaryOrderID. - New diagrams.	EEW, RNKH
Sep. 21st, 2022	1.5.1	- *StreamID* field removed in all messages. - *TradeCondition* field removed from *TradeBust* message. - *TradeDate* field added to *ExecutionStatistics* message. - Scenarios to describe the behavior of *ExecutionSummary* message added. - Scenarios that trigger publication of *StreamReset*, *ChannelReset*, *EmptyBook* messages clarified. - Scenario where snapshot is empty in the loop included. - Fixed message examples. - Description of 12-bit in TradeCondition field changed to: "Block Book Trade = PT".	RNKH, EEW
Nov. 17th, 2022	1.5.2	- Clarification on how to handle price bands information on page 74.	RNKH, TV

Jan. 6th, 2023	1.5.3	- Removed matchEventIndicator (tag 37035) field in the ExecutionSummary message. - mDEntrySize (tag 271) field in the DeleteOrder_MBO message is now optional. - mDEntrySize (tag 271) field in the AuctionImbalance message is now optional. - mDEntrySize (tag 271) and mDEntryPx (tag 270) fields in the TheoreticalOpeningPrice message are now optional. - priceBandType (tag 6939), priceLimitType (tag 1306) and priceBandMidpointPriceType (tag 37008) fields in the PriceBand message are now optional.	RNKH
Apr. 14th, 2023	1.5.4	- Adding minCrossQty (tag 35561) field in the SecurityDefinition message. - Adjusting composition of streams in Feed B (text and figures). - More clarification on alignment and padding in SBE. - Including a chapter for message versioning. - Highlight the need to read block length from SBE Header for seamless support for future additional optional fields at the end of a message (message versioning). - Correcting the declaration of exponent of Price type to -4 from -3. - More details of behavior of iceberg orders in the market data perspective. - More clarification on the definition of a trading platform event.	RNKH
May. 3rd, 2023	1.5.5	- Guidelines is related to the **version 1.5.5** of Binary UMDF specification. - Clarification for changing supported decimal to 8 for *mDEntryPx* field in the *ClosingPrice* message and *netChgPrevDay* field. - Statements in the “Implementing SBE” section adjusted for more clarification.	RNKH
May 11th, 2023	1.5.6	- Clarification for the precision of the *SendingTime* field in the Packet Header. - Clarification for passive orders canceled by STP in a matching event. - Clarification of value of *SequenceNumber* field in *Sequence* message.	RNKH
May 26th, 2023	1.5.6.1	- Fixed order of some messages in matching event in some examples for better clarification. - Disclaimer included for tracking book updates. - Disclaimer included for the nature of UDP transmission (packet loss and out-of-order).	RNKH
June, 9th, 2023	1.6.0	- Highlighting features in the UMDF FIX/FAST not supported in the Binary UMDF. - imbalanceCondition field (ImbalanceCondition type) used instead of tradeCondition field in the *AuctionImbalance* message. - *tradeCondition* type streamlined, non-regular trade types moved to *TrdSubType* type. - *trdSubType* field added to *LastTradePrice*, *Trade* and *ForwardTrade* messages. - Describing cases where *LastRptSeq* is not present in the snapshot of a given instrument.	RNKH
June, 20th, 2023	1.6.0.1	- Completely revised section for schema extension mechanism to support new templates and new versions of the schema.	RNKH
July, 20th, 2023	1.6.0.2	- Including description of the scenarios in which value of *RptSeq* field is reset to one (sections 6.5.13 and 7.4.1). - Including a table to summing up possible values for each trade type that can have others trade condition and trade sub type (section 12.3.3). - Added scenarios when *ExecutionSummary* messages are/are not published before Trade messages (section 12.1). - Added which quantities can compose the hiddenQty field in the *ExecutionSummary* message (section 12.1). - Added details of conditions for MULTI_ASSET_TRADE and LEG_TRADE values in the *TrdSubType* field in the *Trade* message (section 12.3.2).	RNKH

November, 17th, 2023	1.7.0.0	- Sequence Version - Rolling over at the max value of 65534 not at 65535 (null value of the field). See section 6.5.5.1.	RNKH
December, 3rd, 2023	1.7.0.1	- Including a scenario of triggered stop orders aggressing several orders in the section 12.1.3.	RNKH
April, 1st, 2024	1.8.0	- Following new messages are added: *SettlementPrice* (template id=28) in section 12.11.3 and *OpenInterest* (template id=29) in section 12.11.4. - 109 - SWEEP_TRADE added to *TrdSubType* type for Sweep and Cross support in sections 12.3.2 and 12.3.3. - More clarification in the description of *openCloseSettlFlag* field in the messages that uses this field in sections 12.9.1, 12.9.2, 12.9.3 and 12.11.3.	RNKH
May 15th, 2024	1.8.0.1	- Links to repository of schema files fixed at section 6.5.1.	RNKH
May 17th, 2024	1.8.0.2	- Compatibility strategy to handle schema evolution section moved from 6.3.1 to 5.10.	RNKH
August 9th, 2024	1.9.0	- Tag number for *openCloseSettlFlag* field fixed at section 12.11.3. - Table of commonly combination of values for *openCloseSettlFlag* and *settlPriceType* fields included at section 12.11.3. - Implied book mechanism described in section 16. - Minor adjustments in the order and trade messages to include implied flag in the description of *matchEventIndicator* field throughout the document.	RNKH
August 22th, 2024	1.9.0.1	- Least priority nature of the implied order described in a note in section 16. - Moment #4 to illustrate the least priority nature of the implied order behavior included at the end of section 16.1.	RNKH
September 19th, 2024	1.9.0.2	- Complement the description of *mDEntrySize* field in the *DeleteOrder_MBO* message: “Absent if the deletion is the result of a matching event.”. - Description of *matchEventIndicator* field equalized throughout the document. - Settlement prices and Open interest are removed from the section that describes existing features in the FIX/FAST UMDF not supported in the Binary UMDF. - DAILY replaces SESSION in the table of regular schedule of *openCloseSettlFlag* field in *SettlementPrice* message in section 12.11.3.	RNKH
October 15th, 2024	1.9.0.3	- Trade bust that involved an implied order also has the implied flag set in *matchEventIndicator* field. See section 16.3. - More clarification for definition of *matchEventIndicator* field in each of message types that have this field. - Clarification in the case of value of *securityTradingStatus* field is null in snapshots in section 13.	RNKH
November 13th, 2024	1.9.0.4	- Description of events in the *outrights* that impacts the synthetic implied order in section 16. - Description of new behavior when the partial match occurs that involves the synthetic implied order with examples to illustrate in section 16.3.	RNKH

2. Preface

This document outlines the B3 Unified Market Data Feed (UMDF) specification contemplating the use of Simple Binary Encoding (SBE) over UDP multicast transport for Equities, Derivatives and FX market segments on **PUMA Trading System** platform.

The implementation of SBE into UMDF was based on the version 1.0 of the FIX SBE. FIX SBE targets high performance trading systems. It is optimized for low latency of encoding and decoding while keeping bandwidth utilization small. For compatibility, it is intended to represent all FIX semantics.

The encoding standard is complementary to other FIX standards for session protocol and application-level behavior.

B3 provides this market data feed based on the Financial Information eXchange ("FIX") Protocol. FIX is a technical specification for electronic communication of trade-related messages. It is an open standard managed by members of FIX Protocol Limited (http://www.fixprotocol.org/). It is assumed that the reader of this document has basic knowledge of the FIX protocol.

2.1 Abbreviations

Abbreviation	Description
B3	B3 S.A. – Brasil, Bolsa, Balcão
TSG	B3 Trading Support Group.
CFI Code	Classification of Financial Instruments Code.
SBE	Simple Binary Encoding
FIX	Financial Information eXchange Protocol
IP	Internet Protocol
TCP	Transmission Control Protocol
UDP	User Datagram protocol
EQT	The Equities segment, previously available as BOVESPA signal.
DRV	Derivatives and FX segment, previously available as BMF segment.

2.2 Glossary

Term	Definition
Broker	A broker is an individual or firm who acts as an intermediary between a buyer and seller, usually charging a commission.
Brokerage	Used interchangeably with broker when referring to a firm rather than an individual. Also called brokerage house or brokerage firm.
Counterparty	Party to a trade.
DMA	Direct Market Access – functionality that allows end-customers, such as hedge funds or investment banks, to directly access the exchange electronically without the need to go over physical broker firm infrastructure.
FIX Gateway	Service that provides connectivity to third-party clients and brokerages using the FIX protocol.
Instrument	Financial capital in a readily tradable form.
Market Data	A collective term for quotes, last sales, volume statistics and other information used by the market to evaluate trading opportunities.
Matching	The process by which two counterparties that have engaged in a trade compare the settlement details of the offers provided by both. Matching is done to verify all aspects of a trade and ensure that all parties agree on the terms of the transaction.

Term	Definition
IP Multicast	Method of forwarding IP datagrams to a group of interested receivers.
Security	A stock, bond or contract that has been authorized for trading on, and by, a registered exchange. Each exchange has different criteria to determine a security's eligibility for listing.
Vendor	Institution that sells services to its clients. In the context of this document, a vendor is an institution that sells access to market data feeds and order management interfaces to an Exchange.
Snapshot	The snapshot for **one** instrument comprises the state of each order book (bids and asks), as well as some statistics (like High Price and Last Trade Price) and the security status; it is valid as of the sequence number in the Incremental Market Data feed (*LastMsgSeqNumProcessed*). It has only the most recent statistics; previous values (like past trades) cannot be recovered from the current value of the snapshot.
PUMA	B3´s PUMA Trading System, that concentrates the trading for all exchange products.

3. Trading Hours

3.1 Trading Session Hours

For a list of FX, derivatives and equities trading hours and sessions, please visit:

http://www.b3.com.br/en_us/solutions/platforms/puma-trading-system/for-members-and-traders/trading-hours/

3.2 Exchange Holidays

For a list of exchange holidays for the FX, derivatives, and equities segments, please visit:

http://www.b3.com.br/en_us/solutions/platforms/puma-trading-system/for-members-and-traders/trading-calendar/holidays/

4. Features in Binary UMDF

Binary UMDF is the evolution of the current UMDF Market Data platform.

The main new features are:

- **SBE (Simple Binary Encoding)**.
- **Execution Summary** message**.**

Updated features:

- **Market By Order (MBO)** – Order depth books are encoded in SBE.
- **Market By Price (MBP)** and **Top Of Book (TOB) market depths** are not available in SBE**.**

  Please continue using the current UMDF Market Data platform if FAST encoding, MBP or TOB market depths are features required.

Existing features in the FIX/FAST UMDF **not supported** in the Binary UMDF:

- Indexes and Security Lending information (from FIX/FAST Market Data channel 62 and 61 respectively).
- News from News agencies (from FIX/FAST Market Data channel 63).
- Notional volume (*tag 269=B*) statistics.
- Exercise and Blocking (*E&B*) Market Data information.

- TCP Recovery System (*UMDF TCP Replayer*).
- Market Data in FIX over TCP connection (*UMDF TCP*). There isn't *UMDF TCP* in binary format.

4.1 Simple Binary Encoding

Simple Binary Encoding (SBE) is a FIX standard for binary message encoding developed by the High-Performance Working Group of the FIX Trading Community, which aimed to optimize electronic exchange of financial data targeting high volume, low latency data dissemination.

Compared to FAST, it is simpler and faster to encode and decode, and not encumbered by any patents.

4.2 Trading Platform Events

Each action taken on an order or a quote, results in a trading platform event. *Binary UMDF* only supports order action-driven events.

4.2.1 Order Action-driven Events

Entering an order that generates a book update market data message is a simple example of an event.

Entering an order that matches against several resting orders is also an example of an event, and it can generate several market data messages, like trades, volume, book, and statistic updates.

If a market state (group phase or security status change) impacts the order book state, related messages such as order book deletion, resulting trades and statistics make up an event.

Another event to be highlighted is the one resulted in a trade of spreads (user defined or exchange defined) that results in trades and statistics changes of related underlying securities. All of them make up an event.

The last message in a sequence of market data messages that belong to the same event is marked with a 7th-bit in the tag 37035-*matchEventIndicator*.

4.2.2 Market Data messages that do not belong to an Event

Statistics unrelated to order book updates, such as closing prices, price/quantity bands and group phase and security status that do not trigger any changes in the book, do not make up an event and all those messages have a 7th-bit set in the tag 37035-*matchEventIndicator*. This procedure was made so that a client system can unequivocally determine the next event if it occurs after those statistics are disseminated.

4.2.3 Events that do not generate Market Data

Events like order rejections, entry of stop orders, cancellation of unelected stop orders, all of them do not generate Market Data.

4.3 Tracking of Book Updates

Order books are sorted by position number: *mDEntryPositionNo* field (tag 290). Resting orders are referred by their position numbers in decrement order from the top of the book.

Client systems shall manage their internal order-based book with the position number (*mDEntryPositionNo* field) in each of order related messages.

As there are several ***corner cases*** in the B3's order-book management system, client systems should only rely on position numbers in the order related messages (*Order_MBO*, *DeleteOrder_MBO* and *MassDeleteOrders_MBO*) to manage their order books. We do **not guarantee** order-book consistency if client system uses another criteria to manage their order book!

5. Implementing SBE

5.1 SBE Design Principles

As mentioned previously, SBE is optimized for low latency of encoding and decoding, while keeping bandwidth usage small. The main design principles are:

- Usage of native binary data types and simple types derived from native binaries (prices, timestamps).
- Preference for fixed positions and fixed length fields, to streamline direct access to data.
- For compatibility, all FIX semantics are supported.
- Metadata information, like tag numbers and field separators, is not sent; it is available as a message "template" (a collection of message schemas in an XML file).
- SBE encoding and decoding is much simpler and faster than FAST encoding and decoding (no presence maps, no "dictionary contexts" containing previous values of tags in messages or repeating groups, no variable-length integers, no special rules for missing values etc.).

5.2 SBE Specification

B3 uses the **version 1.0 of SBE**, that can be accessed at:
https://www.fixtrading.org/packages/simple-binary-encoding-technical-proposal-final/ .

SBE specification version 1.0 uses the following fixed header that is **very important** for all decoders to read and effectively use them to decode the following SBE body (see Schema Extension Mechanism section):

Name	Type	Size (bytes)	Description
blockLength	uint16	2	The total space reserved for the **root-level** of the message not counting any repeating groups or variable-length fields.
templateID	uint16	2	Identifier of the message template.
schemaID	uint16	2	Identifier of the message schema that contains the template.
schemaVersion	uint16	2	The version of the message schema in which the message is defined.

5.3 SBE Field Order and Speed of Access

SBE templates are not expected to lay out the fields in the same order as the original FIX Message; the order and alignment of the fields are important for fast access.

SBE messages that have no repeating groups behave as fixed-length, fixed-position messages, so they are very friendly to hardware processing (FPGA) and low-level languages (like C). In fact, it just behaves like a simple struct in C language.

5.3.1 Alignment and Padding

For speed of access, the most important fields are laid out in the limits of a 64-byte cache line and aligned according to their sizes (8-byte fields start in an offset that is a multiple of 8, 4-byte fields in an offset that's a multiple of 4, and so on).

Alignment is especially important for speeding up FPGA processing. It is guaranteed by judicious placement of the fields, and padding. (In SBE the alignment and padding are done by specifying the **field offsets** explicitly (see more information at https://www.fixtrading.org/standards/sbe-online/#message-body), and by carefully defining the **block length** of the message (block length must be greater than or equal to the sum of the sizes of all fields in the message or group: see more detail of padding at the end of a message at https://www.fixtrading.org/standards/sbe-online/#padding-at-end-of-a-message-or-group). Normally we tried to avoid 'dummy fields' for padding because it is harder to reclaim unused space later with updated specification. We highlighted field offsets and block lengths that are different than the sum of size of the fields in the Message Reference document for clarification.

5.3.2 Similar Messages with Common Fields

Some messages have common fields (like *securityID* or *matchEventIndicator*) for almost all Incremental Refresh messages). Such fields are placed in the same offset wherever possible.

5.4 Nested FIX Messages vs Flattened SBE Messages

Textual FIX Messages can be very lengthy, very nested, and carry lots of unrelated information in a single message; FAST-encoded messages represent faithfully such FIX messages, so they carry the same cruft as well.

For instance, a single textual FIX *MarketDataIncrementalRefresh* message (tag 35=X) (or its equivalent FAST representation), that must be processed as soon as possible, because it represents trades and order book updates, also carries statistical information that could be easily inferred by the client, like session low and high price. The client must decode the entire message and process all the *MDEntries* – they cannot be filtered before decoding.

This updated specification breaks the larger message (tag 35=X) into its components, or *MDEntries* – each component represented by one SBE message. They occupy more space, but the client can choose to decode and process some SBE messages according to its template ID.

SBE can represent FIX Repeating Groups, but its use was kept at a minimum, mainly in instrument definition messages or some low-traffic messages.

For instance, let's represent this simplified incremental (tag 35=X) message in SBE messages, that represents the first trade of an instrument whose security id = 100988, in the day, followed by the opening price, high price, low price, VWAP price and the matching (removal) of the resting order of number 727042222275:

```
35=X|268=6|
   279=0|269=2|288=85|289=88|1003=10|271=1|270=65265|48=100988|
   279=0|269=4|270=65265|48=100988|
   279=0|269=7|270=65265|48=100988|
   279=0|269=8|270=65265|48=100988|
   279=0|269=9|270=65265|48=100988|
   279=2|269=0|48=100988|198=727042222275|290=1|
```

The corresponding SBE messages are (some fields are not shown):

Message Name	FIX equi-valent	Template ID	Msg Size	Contents
ExecutionSummary	269=s	55	64	SecurityID=100988, LastPx=65265, FillQty=1, *CxlQty*=0, *TradedHiddenQty*=0, *AggressorSide*=2
Trade	269=2, 279=0	53	56	*SecurityID*=100988, *MDEntryBuyer*=85, *MDEntrySeller*=88, *TradeID*=10, *MDEntrySize*=1, *MDEntryPx*=65265
OpeningPrice	269=4, 279=0	15	40	*SecurityID*=100988, *MDEntryPx*=65265
HighPrice	269=7, 279=0	24	34	*SecurityID*=100988, *MDEntryPx*=65265
LowPrice	269=8, 279=0	25	34	*SecurityID*=100988, *MDEntryPx*=65265
DeleteOrder	269=0, 279=2	51	48	*SecurityID*=100988, *MDEntryPositionNo*=1, *SecondaryOrderID*=727042222275, *MDEntryType*=Bid
ExecutionStatistics	269=9	56	52	SecurityID=100988, *TradeVolume*=1, *VwapPx*=65265, *MatchEventIndicator*={**EndOfEvent**}

For example, an application can opt to skip "high price" and "low price" messages just by checking their template IDs, that are 24 and 25. It is easier and faster than decoding the entire FAST message and locating the MD entry with tag 269=7 and tag 269=8.

5.5 SBE, Optional Fields and Default Values, Empty Fields

Optional fields in SBE can hold 'null values', that an application can interpret as 'absence of contents (field is not set)'.

There is no 'default values' in SBE. Applications can replace 'null values' with 'default values' if required but it is not in the protocol specification.

Optional fields in SBE do not save space at all. If a field is defined as an optional *int32* (4 bytes), it always occupies 4 bytes, even though the contents of the field are 'null'.

For optional fields, there is a value that represents the 'null' value (the field is not set). It can be specified in the declaration of their types.

If not explicitly defined in the type, the 'null' value is assumed as default for the primitive type: for unsigned fields, it is the largest possible value (something like 0xFFFF… in hexadecimal); for signed fields, it is the most negative value (something like 0x8000… in hexadecimal); for char fields, it is always the binary value 0 ('\0').

For enumeration fields, it depends on the 'encodingType' attribute (for instance, for the *LotType* enum, whose encoding type is '*uint8*', the 'null' value is 255; for the *SecurityUpdateAction* enum, whose encoding type is 'char', the 'null' value is NUL ('\0').

For some types, the encoding for the 'null value' is not the default, but it is specified in the 'nullValue' attribute for the type. For instance, the type of the 'enteringFirm' field (tag 37501) is *FirmOptional*, whose 'nullValue' attribute is "0". For such field, the value "0" represents null, not zero.

There are two types of strings in SBE: variable-length strings and fixed-length strings. Null string and empty strings are encoded the same way. For variable-length strings, the field 'length' is 0 for null (empty) strings. For fixed-length strings, if the content is shorter than the specified length, must be delimited by NUL ('\0') character.

From the SBE specification, the *default* values for null value are:

Primitive Type	Value	Decimal	Hexadecimal
int8	-128	-128	0x80
uint8	255	255	0xFF
int16	-32768	-32768	0x8000
uint16	65535	65535	0xFFFF
int32	-2^{31}	-2147483648	0x80000000
uint32	$2^{32}-1$	4294967295	0xFFFFFFFF
int64	-2^{63}	-9223372036854775808	0x8000000000000000
uint64	$2^{64}-1$	18446744073709551615	0xFFFFFFFFFFFFFFFF
char	$0\ (ASCII\ NULL)$	0	0x00 ('\0')
decimal (int32 mantissa)	Mantissa: -2^{31}	-2147483648	0x80000000
decimal (int64 mantissa)	Mantissa: -2^{63}	-9223372036854775808	0x8000000000000000

5.5.1 Decimal ‘null’ value

For instance, the type ‘Price’ is defined as a Decimal; the mantissa type is ‘*int64*’ and the exponent is fixed as -4. A price like ‘+12.34’ is encoded as the long value ‘123400’ (it is 12.34 times 10^4, or 10000) but the null price (that can be found in market orders, that have no defined price) is encoded as the special value 0x8000000000000000, or -9223372036854775808.

For some decimal fields, it is possible to have null values encoded as ‘0’ instead. Typically, they are values that are strictly positive (cannot assume the value 0.0). Check the SBE template searching for ***nullValue*** attribute in type declaration.

5.6 SBE Templates

SBE templates provide the rules for an SBE decoder to be able to properly decode market data messages. SBE-encoded messages can only be interpreted correctly by using such templates.

The templates are all listed within a single XML file. The templates are subject to change by B3 as the system evolves and new functionality is added. When a change is done, B3 will notify market participants in advance for appropriate development and/or testing efforts.

The version of the schema file is shown in the “version” attribute of the `<sbe:messageSchema>` XML element. For example:

```
<sbe:messageSchema
        xmlns:ns2=”http://www.fixprotocol.org/ns/simple/1.0”
        xmlns:xsi=”http://www.w3.org/2001/XMLSchema-instance”
        xmlns:sbe=”http://fixprotocol.io/2016/sbe”
        package=” b3.umdf.mbo.sbe”
        id=”2” version=”6”
        semanticVersion=”1.5.4”
        description=”B3 Market Data UMDF SBE messages”
        byteOrder=”littleEndian”
        xsi:schemaLocation=”http://fixprotocol.io/2016/sbe sbe.xsd”>
```

The modifications to the template file are documented using the “*sinceVersion*” attribute. In the example below, a new value: “*RecoveryInProcess*” in the enumeration domain is only available after the version is rolled out in the production environment:

```
<validValue name=”RecoveryInProcess” description=”Recovery In Process”
sinceVersion=”6”>6</validValue>
```

Please note that the templates for **Market Data** messages and for **Order Entry** messages are listed in separate XML files and represent different schemas. Even though they could have messages with the same IDs, the schema IDs are different, so a program can import both Market Data and Order Entry templates without clashes.

5.7 FIX Message and SBE Templates

To better reflect different usages and save space, different templates must be defined.

The use of different templates for different *MDEntryType* combinations helps to keep space usage low, because a lot of fields are conditional in FIX (used for some *MDEntryTypes* but not for others) and would take a lot of space in SBE messages if they were all included.

If the original FIX Message has fields that are not found in the SBE template, these fields are not encoded or decoded.

5.8 Limiting the SBE message size

Unlike FIX/FAST messages (that have unlimited size), B3 limited the size of the SBE messages to fit a single datagram (UDP packet) – **1400** bytes.

It helps applications to optimize message processing because they can allocate fixed space for every message and easily reuse that space.

Applications do not need to collect the packets and consolidate them to decode messages. If a packet is received and it is not corrupted, messages can always be successfully decoded from this packet.

Limiting the message size is ideal for small messages, like incremental messages that update an order book; but large messages, like *News* messages, which can have potentially unlimited size but fortunately have low traffic, are handled in a special way. These large messages are broken into “parts” (smaller messages that have two additional fields: *partCount* and *partNumber*). The client application can collect such messages (*partNumber* = 1 to *partCount*) and consolidate fields of them (for instance, the News message has a ‘text’ field (tag 58) that must be consolidated).

5.9 Schema Extension Mechanism

5.9.1 Objective

It is not always practical to update all message publishers and consumers simultaneously. Within certain constraints, messages and repeating groups 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.

Message templates and repeating groups may be extended with new fields. However, the extension mechanism does not support extension of composite types that back existing fields.

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.

5.9.1.1 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.
- Inserting a new field in a reserved space not used before.
- Existing fields cannot change data type or move within a message.
- Message and repeating group byte alignment may not change.
- A repeating group may be added after existing groups at the root level or nested within another repeating group.
- A variable-length data field may be added after existing variable-length data at the root level or within a repeating group.
- Message header encoding cannot change.
- Adding a new choice for an enumeration or a set (the encoding type must remain the same).
- Adding a constant field at any position in the message.
- 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. A message template that has changed in an incompatible way must be assigned a new **template "id"** attribute.

5.9.2 Message schema features for extension

5.9.2.1 Schema version

The `<messageSchema>` root element contains a version number attribute. 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.

5.9.2.2 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.

5.9.2.3 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. 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.

5.9.2.4 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.

5.9.3 Wire format features for extension

5.9.3.1 Block size

The **length** of the root level of the message is sent on the wire in the SBE message header. 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.

5.9.3.2 Number of repeating groups and variable data

Message headers and repeating group dimensions carry a count of the number of repeating groups and a count of variable-length data fields on the wire. This supports a walk by a decoder of all the elements of a message, even when the decoder was built with an older version of a schema. As for added fixed-length fields, new repeating groups cannot be interpreted by the decoder, but it still can process the ones it knows, and it can correctly reach the end of a message.

5.10 Compatibility strategy to handle schema evolution

It is necessary to state that B3 will make any market data schema updates in tranches fashion following previously released plan. So, it's important to notice that during the roll-out period, not all channels will be publishing in the same schema version.

Basically, there are two strategies that B3 will adopt, based on the complexity of the changes. But to be make easier for client application be aware of changes, for every change to the message schema, **schema version will be increased**, so the decoder can handle them from reading the SBE header before starting to decode the body of the message.

5.10.1 Increment schema version and changes are made for some templates

This strategy is normally adopted for incremental changes that does not break template backward compatibility. Normally the type of changes that are backward compatible includes:

- Adding new optional fields at the end of root block or at the end of existing repeating group.
- Adding new optional fields in available unused spaces reserved previously.
- New repeating group block is added at the end of existing repeating group block but there is no variable length data after that.
- New variable length data is added at the end of the message.
- New values are added to existing enumeration types that has already been used by some existing message.
- New choices in unused bits are added to existing sets.

New fields, values and choices need to include “*sinceVersion”* attribute so modified decoders can support extensions smoothly. “Version” attribute in the root node of the “*messageSchema”* needs to be incremented as well so the SBE header of received message will have the updated schema version.

For clarification, we are adding this example: client application is adapted to process version “7” of trade message that includes a new optional field named *trdSubType*.

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. *trdSubType* field can be parsed, and if the business logic is correctly implemented, it can know what type of non-regular trade the trade is.

If the received version is less than the decoder's version (that is, the producer's encoder is older than the consumer's decoder from market data channel that has not updated yet), then only the fields of the old version may be parsed. This information is available through metadata as "*sinceVersion*" attribute of an added 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.

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 newly added fields. In most cases, the state of the client application probably will be in consistent state but without additional information provided by the updated version.

5.10.2 Schema version is incremented, new template is created for existing message

This strategy is normally adopted for disruption changes that breaks template backward compatibility. B3 will mark the old template as deprecated using "*deprecated*" attribute.

Because B3 will make any market data schema updates in tranches of channels, B3 strongly recommended that client applications support decoding/processing both templates: the old and the new one and process them accordingly.

To illustrate this strategy, we are giving this example: client application is adapted to process both templates: the new one (58) and also the old one (53). The new template "id" = 58 of trade message includes a new field named *trdSubType* but removes the *tradeCondition*. The older template "id" = 53 of trade message does not include the optional *trdSubType* field but has the *tradeCondition* field.

If the client application receives a trade message with the new template "id" (58) then all fields known to the decoder may be parsed, and no further analysis is required. *trdSubType* field can be parsed, and if the business logic is correctly implemented, it can know what type of trade from this field. *tradeCondition* field is not parsed.

If the received message has old template "id" (53), that is, the producer's encoder is publishing older template "id" than the consumer's decoder expected – market data channel that has not updated yet, then only the fields of the old version of the template may be parsed (in this example, all fields including *tradeCondition* field but not *trdSubType* field). How an application can handle new/modified fields that are unavailable in the old template and present in the newer template is an implementation detail. One strategy is for an application to provide a default value for unavailable/modified fields.

If the received version is greater than the decoder’s version (that is, the producer’s encoder is newer than the consumer’s decoder) for migrated market data channel, client application will not receive any messages with the old template “id” (in this example, template “id” = 53) and will receive a message with the newer template “id” (58) that does not know how to process them and possibly will be in **inconsistent state**.

Client systems must process SBE Header for each message.

It is extremely important that client applications that consume market data messages in SBE format must process *schemaVersion* and *blockLength* fields in the **SBE Header** instead of assuming them from the template definition because additional compatible versions of the same message (same template “id”) will likely have different schema version and greater block length to accommodate new optional fields. If the message has repeating groups or variable length data, these structures will be shifted to the right according to the block length of new optional fields added to the end of the root-level block.

6. System Architecture

The *Binary UMDF* platform will be available alongside the current FIX/FAST UMDF feed.

There are two principal aspects on this market data architecture: the concept of a "market data channel" – which defines how the feed is logically distributed according to a set of instruments and level of information of the book; and the "SBE engagement rules" – which define the transport of the information and how the client system should synchronize the data provided into the market data channels.

6.1 Market Data Channel

Binary UMDF is based on **UDP Multicast** to disseminate B3's market data information such as books, trades and statistics. A channel is a logical group of multicast IP addresses, UDP ports.

The UDP protocol itself trades reliability of performance and does not guarantee the datagrams delivery.

Therefore, the packet could be lost during the network transmission.

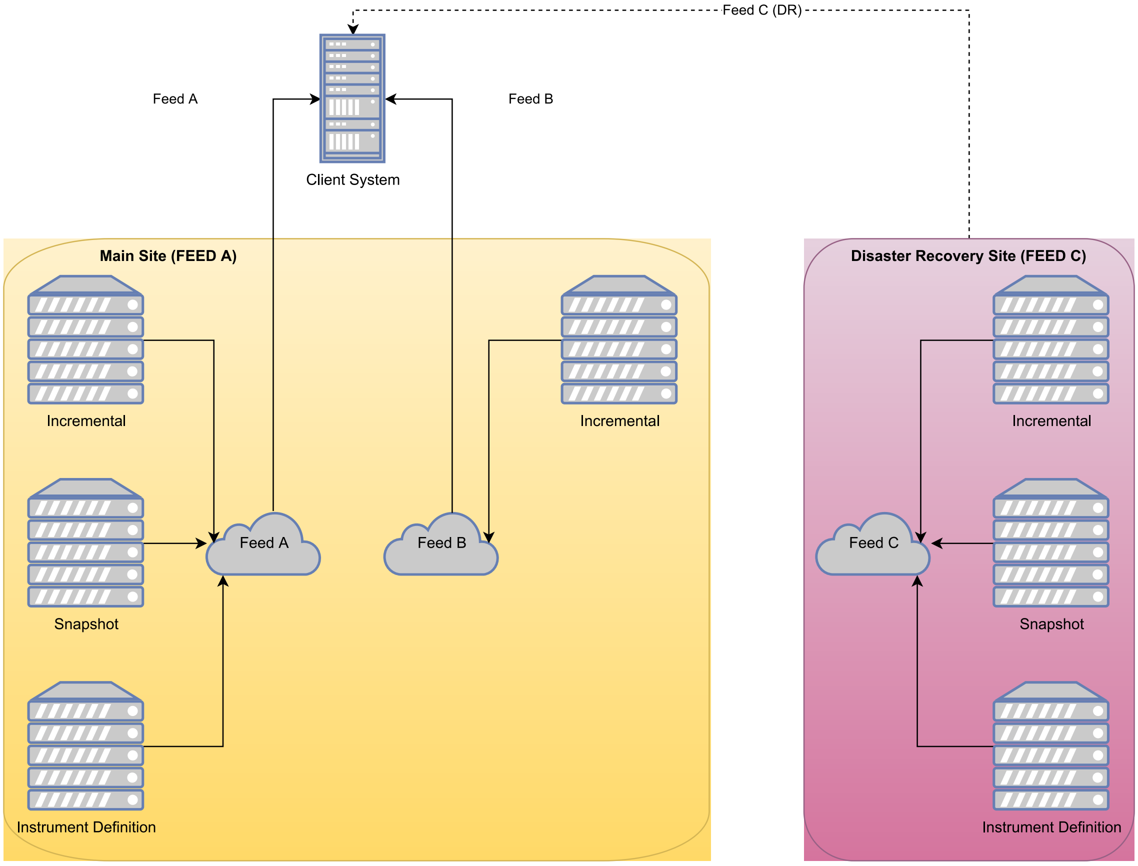
Even if the packet reaches the network node, it does not always mean that the application receives it because during processing the received packet goes through several levels, on each level there could be a loss.

Out-of-order packets can also be caused by UDP traffic. This issue occurs primarily due to stateless connections and the lack of flow control mechanisms that exist within UDP protocol.

Every channel provides market data of a list of instruments (or security list) that have common characteristics, as determined by the exchange.

A channel is logically broken up into 3 streams: an "Incremental Stream", an "Instrument Definition Stream" and a "Snapshot Recovery Stream".

There is a main feed (Feed A) for every stream. A secondary feed (Feed B) only has the incremental stream, and the purpose of this feed is to **mitigate the packet lost** if properly listening on both feeds (A and B) simultaneously and processes the first received packet, discarding the second one.



For contingency purposes, B3 provides a feed (Feed C) that is generated at its contingency site (DR – Disaster Recovery).

The secondary feed contains the **exact same data** that is sent over the primary feed, however with different connectivity information (different UDP multicast addresses and ports) and path (to mitigate network outage and instability).

B3 strongly suggest that customers sign up to receive both feeds (feed A and feed B) and process the first datagram received, because incremental messages that can be lost or out of sequence can be retrieved by the nature of UDP transmission.

6.2 Incremental Stream

Used to disseminate B3 incremental market data and other real time data such as news, instrument updates, instrument status using SBE encoded messages.

If no data is sent through the incremental stream for more than 1 (one) second (can be changed after notifications from TSG), B3 will issue a *Sequence* message for maintaining connectivity. If client systems do not receive any messages within 3 consecutive heartbeat intervals, the incremental stream should be considered not functional, and the book state should be considered inconsistent.

6.3 Snapshot Recovery Stream

Snapshot recovery is used to disseminate B3 market data snapshot message for instruments belonging to that channel.

The snapshot for one instrument comprises the state of each book (bids and asks), as well as some statistics (like *High Price* and *Last Trade Price*) and the security status; it is valid as of the sequence number in the Incremental Market Data feed (*lastMsgSeqNumProcessed*). It has only the most recent statistics; previous values (like past trades) cannot be recovered from the current value of the snapshot. It is transmitted using several SBE messages that can occupy one or more packets. It also keeps the last value of *rptSeq* (as *lastRptSeq*), that represents the sequence number per instrument.

The market data snapshot messages are replayed at a specific rate and should be used as the primary source for all book synchronization.

Once the books are synchronized and the client starts using only the incremental stream, the client should unjoin the stream as it would take up unnecessary bandwidth.

NOTE

The number of order book snapshots that is sent in the snapshot recovery stream in one loop could be less than the number of instruments to the related channel. Client systems must handle instruments with no snapshots as having empty order books and no statistical data before applying incremental data.

6.4 Instrument Definition Stream

The instrument definition stream is used to recover the list of all instruments that is currently assigned to that channel. The list is replayed periodically and starts over once the last instrument definition message is received.

NOTE

Each *SecurityDefinition* message conveys the definition of a single instrument.

6.5 Engagement Rules

This section contains an overview of engagement architecture for receiving the SBE market data feed.

6.5.1 Latest SBE Templates

SBE messages are not self-describing, and all metadata (message schemas) must be previously fetched before decoding.

Client systems must be prepared for template changes.

If schema evolution is required, B3 will release an updated version of the schema template file, with the new or updated templates.

Schema files are available at the B3 web site, at the following addresses:

In Portuguese: https://www.b3.com.br/pt_br/solucoes/plataformas/puma-trading-system/para-desenvolvedores-e-vendors/umdf-binario/

In English: https://www.b3.com.br/en_us/solutions/platforms/puma-trading-system/for-developers-and-vendors/binary-umdf/

6.5.2 Network Configuration

B3 will provide clients with the necessary network configuration information to allow them to receive all market data channels.

See the documents: Market Data Channels Definition or contact the TSG for the list of certification and production environments (multicast channels definitions).

Please note that FIX/SBE multicast data is available through the RCB (Rede de Comunicação B3, or B3 Communications Network).

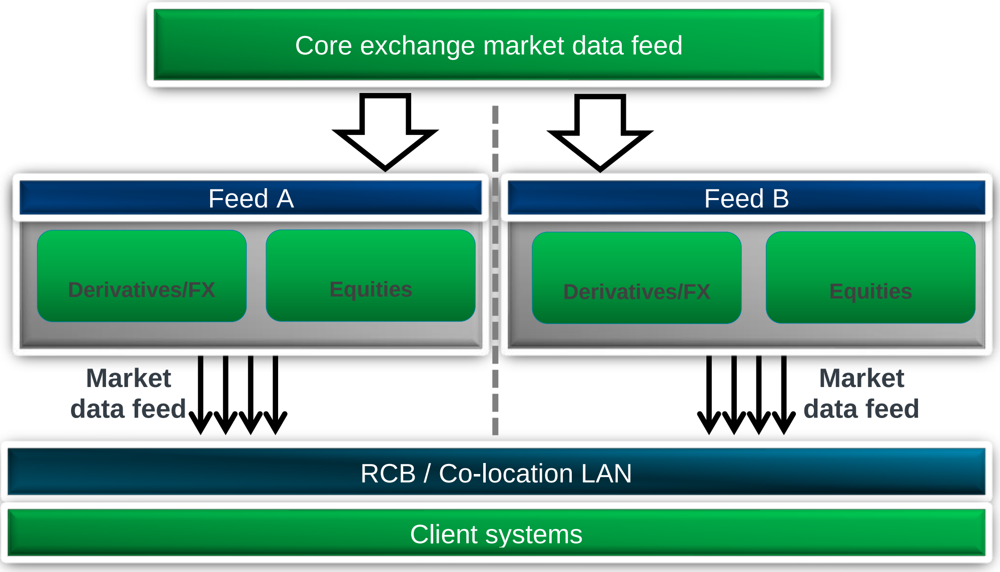
NOTE

SBE Multicast Data is available through the RCB and inside B3's datacenter for co-location customers.

6.5.3 Market Data Network Contingency Feed

B3 offers to all customers (RCB and Colocation) the ability to receive a network contingency feeds that passing on different internal route to strengthen stability and increase network fault tolerance.

The following diagram illustrates the primary and backup feeds distribution:



B3 suggests customers to sign up for both feeds, to increase stability.

There is a third feed intended to be used for disaster recovery purposes, called Feed C; clients are advised to sign up for the Feed C as well.

NOTE

On Binary UMDF, both Incremental feeds (A and B) share the exact same messages, so customers are encouraged to connect to **both feeds simultaneously** for better reliability and to **handle packet losses efficiently**.

6.5.4 Message Framing

All Market Data messages will use the SBE Format (Simple Binary Encoding), in *little-endian* format.

They need to be framed because the main network transport is UDP, that is datagram-based (data is sent in packets). The same framing will be used for stream-based protocols, like TCP (FIX Sessions).

To make the decoding easier and more amenable to hardware acceleration (like FPGAs), all messages must be smaller than a packet and kept as simple as possible.

The packet size will be smaller or equal to 1400 bytes, to be compatible with most VPNs.

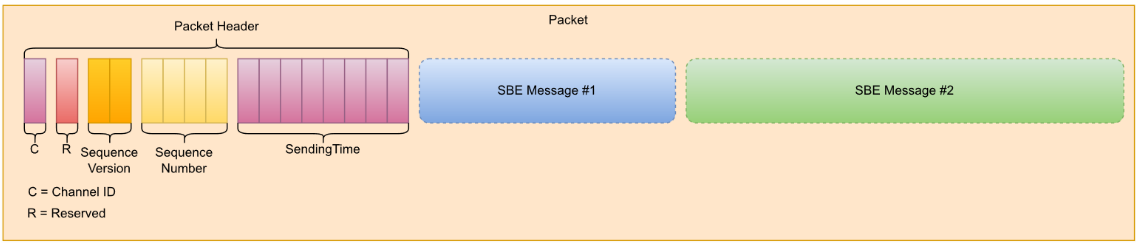
6.5.5 Packet Structure

The Packet structure described below is applied for all 3 streams: incremental, instrument definition and snapshot streams.

6.5.5.1 Packet Header

Each packet (datagram) has one Packet Header, and one or more messages inside. The Packet Header is in *little-endian* format (the least significant values come first). The total size is 16 bytes.

The figure below shows a packet with two messages:



One Packet Header has the following fields:

Name	Type	Size (bytes)	Description
ChannelID	uint8	1	Channel identification.
Reserved	uint8	1	Reserved.
SequenceVersion	uint16	2	Packet Sequence Version. For incremental stream, it starts with 1 at the rollout in the production environment and incremented on weekly basis or in case of failover events. For instrument definition and snapshot streams, its value changes for each new loop. its value will be rolled back to 1 if incremented beyond 65534.
SequenceNumber	uint32	4	Packet Sequence Number. Always incremented by one in the same channel and same *SequenceVersion*.
SendingTime	uint64	8	UTC date and time of message transmission, in nanoseconds since Unix epoch (Jan 1st. 1970), with microsecond-level precision*.

* - Precision is the fineness to which an event can be measured repeatably and reliably.

6.5.5.2 Example

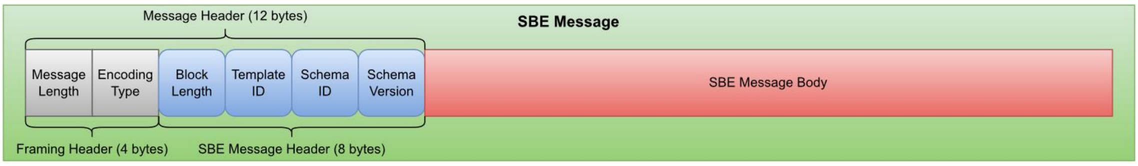
```
         +-------------------------------------------------+
         | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f |
+--------+-------------------------------------------------+----------------+
|00000000| 37 00 01 00 b1 68 de 3a 00 c8 98 65 f4 ac eb 15 |7....h.:...e....|
+--------+-------------------------------------------------+----------------+
```

Offset	Length	Field	Hex bytes	Encoded value
0000	1	ChannelID	37	55 (Channel #55)
0001	1	Reserved	00	-
0002	2	SequenceVersion	01 00	1
0004	4	SequenceNumber	b1 68 de 3a	0x3ade68b1 = 987654321
0008	8	SendingTime	00 c8 98 65 f4 ac eb 15	0x15ebacf46598c800 = 1579546260000000000 = Jan 20, 2020 18:51:00.000000000

6.5.5.3 Message Header

Each message in the packet starts with a *Message Header* that consists of the *Framing Header*, which is a compact form of **SOFH** – *Simple Open Framing Header*, and the *SBE Message Encoding* Header.

The message header is in *little-endian* format (the least significant values come first). The total size is 12 bytes.



One *Message Header* has the following fields:

Name	Type	Size (bytes)	Description
			(Framing Header)
messageLength	uint16	2	Overall message length **including** headers to support framing.
encodingType	uint16	2	Identifier of the encoding used in the message payload (always “50 EB – SBE 1.0 Little-Endian)
			(SBE Message Header)
blockLength	uint16	2	The total space reserved for the **root-level** of the message not counting any repeating groups or variable-length fields.
templateID	uint16	2	Identifier of the message template.
schemaID	uint16	2	Identifier of the message schema that contains the template.
schemaVersion	uint16	2	The version of the message schema in which the message is defined.

6.5.5.4 Example – A packet with a single message

```
          +-------------------------------------------------+
          | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f |
+---------+-------------------------------------------------+----------------+
|00000000| 37 00 01 00 b1 68 de 3a 00 c8 98 65 f4 ac eb 15 |7....h.:...e....|
|00000010| 48 00 50 eb 3c 00 32 00 02 00 03 00 a4 92 78 48 |H.P.<.2.......xH|
|00000020| 17 00 00 00 00 c8 98 65 f4 ac eb 15 20 00 31 45 |.......e.... .1E|
|00000030| 64 00 00 00 a8 5e bc 00 00 00 00 00 64 00 00 00 |d....^......d...|
|00000040| 00 00 00 00 d1 2f 01 00 00 00 00 00 00 00 00 00 |...../..........|
|00000050| 00 00 00 00 d1 e1 01 00                         |........        |
+---------+-------------------------------------------------+----------------+
```

Offset	Length	Field	Hex bytes	Decoded value
0000 … 000F	16	Packet Header	See above	
0010	2	messageLength	48 00	0x0048 = 72
0012	2	encodingType	50 eb	0xEB50 = SBE 1.0 Little-Endian
0014	2	blockLength	3C 00	0x003C = 60
0016	2	templateID	32 00	0x0032 = 50 (Order_50)
0018	2	schemaID	02 00	0x0002 = 2
001A	2	schemaVersion	03 00	0x0003 = 3
001C … 0057	60	SBE Message Body	a4 92 78…	

6.5.5.5 A packet with two SBE messages

This is the binary representation of a packet containing two SBE messages (an Order and a Trade):

```
          +-------------------------------------------------+
          | 0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f |
+---------+-------------------------------------------------+----------------+
|00000000| 37 00 01 00 b1 68 de 3a 00 c8 98 65 f4 ac eb 15 |7....h.:...e....|
|00000010| 48 00 50 eb 3c 00 32 00 02 00 03 00 a4 92 78 48 |H.P.<.2.......xH|
|00000020| 17 00 00 00 00 00 00 00 00 00 00 00 80 01 31 45 |..............1E|
|00000030| 0a 00 00 00 00 61 bc 00 00 00 00 78 e0 01 00 |.....a......x...|
|00000040| 00 00 00 00 38 56 4c 05 00 00 00 00 00 c8 98 65 |....8VL........e|
|00000050| f4 ac eb 15 39 30 00 00 40 00 50 eb 34 00 35 00 |....90..@.P.4.5.|
|00000060| 02 00 03 00 a4 92 78 48 17 00 00 00 00 c8 98 65 |......xH.......e|
|00000070| f4 ac eb 15 80 00 01 45 d0 07 00 00 00 61 bc 00 |.......E.....a..|
|00000080| 00 00 00 00 00 00 00 00 00 00 00 00 81 e3 01 00 |................|
|00000090| 39 30 00 00 04 00 74 4a                         |90....tJ        |
+---------+-------------------------------------------------+----------------+
```

Offset	Length	Field	Hex bytes	Decoded value
0000 … 000F	16	Packet Header	See above	
0010	2	messageLength	48 00	0x0048 = 72
0012	2	encodingType	50 eb	0xEB50 = SBE 1.0 Little-Endian
0014	2	blockLength	3C 00	0x003C = 60
0016	2	templateID	32 00	0x0032 = 50 (Order_50)
0018	2	schemaID	02 00	0x0002 = 2
001A	2	schemaVersion	03 00	0x0003 = 3
001C … 0053	60	SBE Message Body	a4 92 78…	
0058	2	messageLength	40 00	0x0040 = 64

Offset	Length	Field	Hex bytes	Decoded value
005A	2	encodingType	50 eb	0xEB50 = SBE 1.0 Little-Endian
005C	2	blockLength	34 00	0x0034 = 52
005E	2	templateID	35 00	0x0035 = 53 (Trade_53)
0060	2	schemaID	02 00	0x0002 = 2
0062	2	schemaVersion	03 00	0x0003 = 3
0064 … **0097**	52	SBE Message Body	a4 92 78…	

6.5.6 Examples of SBE Message Encoding

Please refer to the FIX SBE Technical Specification 1.0 with errata (November 2020).

It is available at:
https://www.fixtrading.org/packages/simple-binary-encoding-technical-specification-final/.

A few samples will be given below to provide a better understanding of the specification.

6.5.6.1 Message with no repeating groups

Encode a *Sequence* message whose template id is 2, schema id is 2, schema version is 0, and the value of *nextSeqNo* field is 27182818.

The SBE definition of the *Sequence* message is:

```
<sbe:message name="Sequence_2" id="2" semanticType="0">
    <field name="messageType" id="35" type="MessageType" presence="constant"
valueRef="MessageType.Sequence"/>
    <field name="applVerID" id="1128" type="ApplVerID" presence="constant"
valueRef="ApplVerID.FIX50SP2"/>
    <field name="nextSeqNo" id="35526" type="SeqNum"/>
</sbe:message>
```

Encoding the SBE Message: the first and second fields are constant (*messageType* and *applVerID*), so only the third field (*nextSeqNo*) must be encoded. Its type is '*SeqNum*', that is an alias for "unsigned integer with 32 bits'.

27182818 (decimal) = 019EC6E2 (hex) = E2 C6 9E 01 (reverse the order of the bytes, because it is little-endian)

The message payload has just 4 bytes: E2 C6 9E 01.

The whole message is encoded as

10 00 50 EB 04 00 02 00 02 00 00 00 E2 C6 9E 01

As seen in the table below:

Offset	Field	Value (decimal)	Value (Hexadecimal)	Value (Bytes)
00	messageLength	16	0010	10 00
02	encodingType	60240	(0xEB50)	50 EB
04	blockLength	4	0004	04 00
06	templateID	2	0002	02 00
08	schemaID	2	0002	02 00
0A	schemaVersion	0	0000	00 00
0C	*nextSeqNo*	27182818	019EC6E2	E2 C6 9E 01

6.5.6.2 Example – Message with one repeating group level

Let us encode a simpler version of the *SecurityDefinition* message. *SecurityDefinition* messages are very lengthy. A sample template for a special case will be shown instead. It could be used for encoding security definitions for index instruments.

Let us say that this *SecurityDefinitionForIndexInstruments* message whose template id is 777, schema id is 1, schema version is 0, is defined by:

```
<sbe:message name=”SecurityDefinitionForIndexInstruments” id=”777”>
    <field name=”messageType” id=”35” type=”MessageType” presence=”constant”
valueRef=”MessageType.SecurityDefinition”/>
    <field name=”symbol” id=”55” type=”Symbol6”/>
    <group name=”noUnderlyings” id=”711” dimensionType=”GroupSizeEncoding”>
        <field name=”underlyingSymbol” id=”311” type=”Symbol6”/>
        <field name=”indexPct” id=”6919” type=”Percentage8”/>
        <field name=”indexTheoreticalQty” id=”37021” type=”Quantity”/>
    </group>
</sbe:message>
```

Where “*Symbol6*” is a 6-character string, defined by:

```
<type name=”Symbol6” primitiveType=”char” length=”6” characterEncoding=”ASCII”
semanticType=”String” description=”Ticker symbol” />
```

If the symbol has less than 6 characters, it is padded with binary zeros. For instance: “B3SA3” (a 5-character symbol) is represented by the bytes 42 33 53 41 33 00.

“*Percentage8*” is a decimal fixed-point integer, defined by:

```
<composite name=”Percentage8” semanticType=”Percentage” description=”Percentage with constant
exponent -8”>
    <type name=”mantissa” description=”mantissa” presence=”optional” primitiveType=”int64”/>
    <type name=”exponent” description=”exponent” presence=”constant” primitiveType=”int8”>-8</type>
</composite>
```

The value “-8” tells us to get the value (mantissa) and divide it by 1 followed by 8 zeros (or multiply by 10^{-8}).

“*Quantity*” is a 4-byte integer, defined by

```
<type name=”Quantity” primitiveType=”uint32” semanticType=”Qty” description=”Quantity in
order/trade”/>
```

And “*GroupSizeEncoding*” is a type used for declaring repeating groups, defined by:

```
<composite name=”GroupSizeEncoding” description=”Repeating group dimensions”>
    <type name=”blockLength” primitiveType=”uint16”/>
    <type name=”numInGroup” primitiveType=”uint8” semanticType=”NumInGroup”/>
</composite>
```

The contents of the message are:

55=IBOV, 711=2,

311=PETR4, 6919=1.10863820, 37021=51860760,

311=VALE5, 6919=0.4702920, 37021=19792285

The encoded message is:

```
39 00 50 EB 06 00 09 03 01 00 00 00 49 42 4F 56 00 00 12 00 02 50 45 54 52 34
00 CC A5 9B 06 00 00 00 00 18 55 17 03 56 41 4C 45 35 00 D0 9B CD 02 00 00 00
00 9D 01 2E 01
```

Let us analyze it:

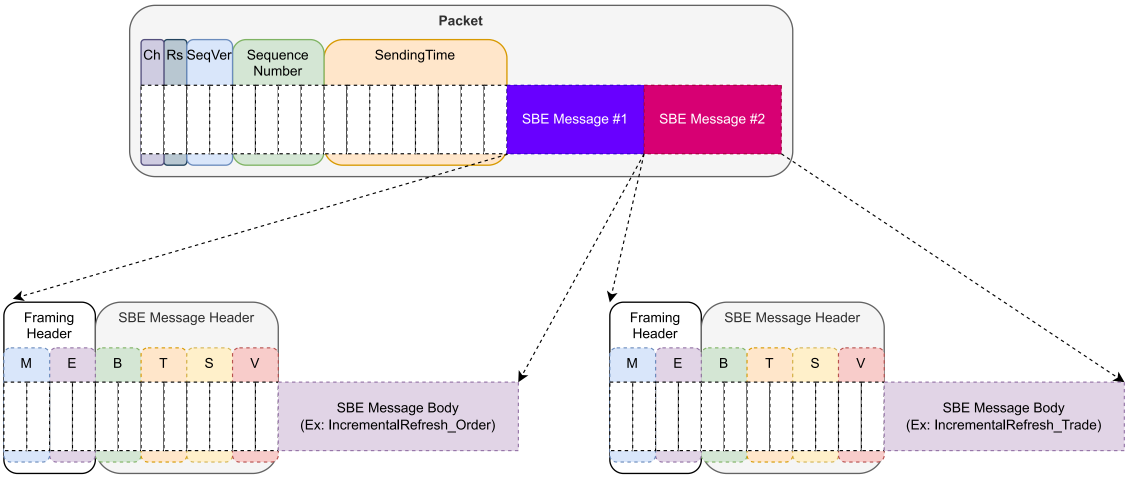
#	Field	Value (Bytes)	Decoded value
	messageLength	39 00	57
	encodingType	50 EB	60240
	blockLength	06 00	6
	templateID	09 03	777
	schemaID	01 00	1
	schemaVersion	00 00	0
	symbol (55)	49 42 4F 56 00 00	“IBOV”
	noUnderlyings (711): *GroupSizeEncoding.**blockLength***	12 00	18
	noUnderlyings (711): *GroupSizeEncoding.numInGroup*	02	2
1	*underlyingSymbol* (311)	50 45 54 52 34 00	“PETR4”
1	*indexPct* (6919)	CC A5 9B 06 00 00 00 00	1.10863820 (00 00 00 00 06 9B A5 CC = 110,863,820; divide by 100,000,000)
1	*indexTheoreticalQty* (37021)	18 55 17 03	51,860,760 (03 17 55 18 = 51860760)
2	*underlyingSymbol* (311)	56 41 4C 45 35 00	“VALE5”
2	*indexPct* (6919)	D0 9B CD 02 00 00 00 00	0.4702920 (00 00 00 00 02 CD 9B D0 = 47,029,200; divide by 10000000)
2	*indexTheoreticalQty* (37021)	9D 01 2E 01	19,792,285 (01 2E 01 9D = 19792285)

There are two *blockLength* fields in the message above: the first one appears in the *SBE Message Encoding* Header and tells us how many bytes are reserved for the “root level” of the message. The definition above tells that the root level has only the tag 55-*symbol* and the field occupies just 6 characters, so the value to use is 6 (06 00).

The second one (*noUnderlyings.blockLength*) tells us how many bytes are allocated for the fixed part of each repeating group. The definition above tells that each repeating group has the following fields: tag 311-*underlyingSymbol*, whose size is 6, tag 6919-*indexPct*, whose size is 8, and tag 37021-*indexTheoreticalQty*, whose size is 4. Therefore, the total size is 6 + 8 + 4 = 18, or 12 00.

6.5.7 Messages, Packets and Events

A packet can contain a single message, or several messages. To keep bandwidth low, the system can choose to send several messages in a single packet.



One event can generate a single message, or a series of messages.

It can extend for several packets.

The packet can contain messages from several distinct events (as described in chapter 4.2).

6.5.8 Instrument Definition Stream Processing

The instrument definition stream replays the list of instruments of a specific channel at an exchange-defined rate. To correctly process the full list of instruments for that channel, client systems must join the instrument definition stream.

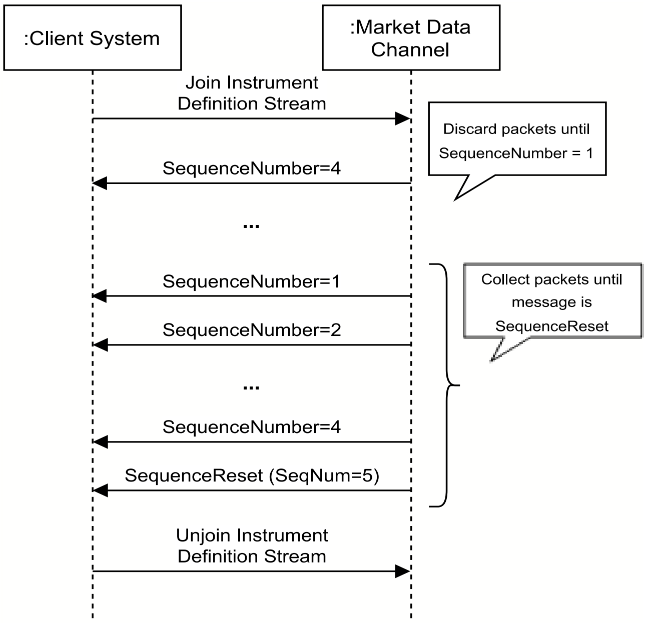
Client systems should begin recovering messages when *SequenceNumber* field (in the Packet Header) is 1 and use the tag *393-TotNoRelatedSym* (in the *SecurityDefinition* message) to retrieve the complete set of Security Definitions. *TotNoRelatedSym* field contains the number of *SecurityDefinition* messages to be published in the current loop.

Every loop ends with a *SequenceReset* message. The next message will be published with new *SequenceVersion* and *SequenceNumber* = 1.

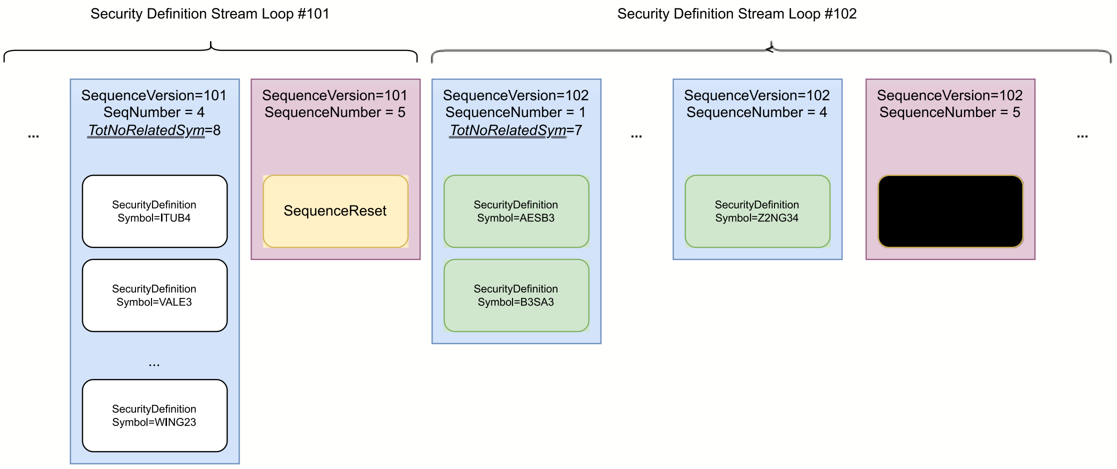
One instrument definition is defined by a single *SecurityDefinition* message. A packet can have several *SecurityDefinition* messages.

Deleted and expired instruments are not sent over the instrument definition stream; the application must process the *SecurityDefinition* message sent over the incremental stream.

The following diagram illustrates correct client system processing of the instrument definition stream:


:Client System
:Market Data Channel
Join Instrument Definition Stream
Discard packets until SequenceNumber = 1
SequenceNumber=4
...
SequenceNumber=1
Collect packets until message is SequenceReset
SequenceNumber=2
...
SequenceNumber=4
SequenceReset (SeqNum=5)
Unjoin Instrument Definition Stream

The packets and messages mentioned above are:



B3 will start issuing instrument definition messages in the instrument definition stream using the schedule described in the table below (all times are local unless stated otherwise):

Trading platform	Segment	Schedule
PUMA	Equities	Not restarted daily, brought down between Fri 22:00 and Sun 12:00 (local time)
PUMA	Derivatives/FX	Not restarted daily, brought down between Fri 22:00 and Sun 12:00 (local time)

The other feeds (incremental and snapshot recovery) are also activated at this time, but messages are only sent as they become available. In general, for PUMA Trading System, customers may connect every day or keep connected through the week. However, B3 recommends that customers remain disconnected during the weekends, unless when participating in scheduled Simulated Trading Sessions (a.k.a. mock tests).

6.5.9 Initial Market Data Synchronization Procedure

The process described below details the activities need to ensure a proper synchronization is complete, therefore all necessary market data is received by the client system. It is advisable to retrieve from B3's website the latest configuration parameters and template files or contact the TSG for more information.

1. Join the multicast address/UDP port of the incremental stream and start receiving the market data incremental messages. Queue them.

2. Join the multicast address/UDP port of the security definition stream until all instruments have been received (monitor the tag 393- *TotNoRelatedSym*).

3. Unjoin the security definition stream, to avoid consuming unnecessary bandwidth.

4. Simultaneously with receiving security definition stream, join the multicast address/UDP port of the snapshot recovery stream until all snapshot messages have been received: monitor the Packet Header field *SequenceNumber*, whose value is cyclical, and the tag 911-*TotNumReports* = total number of snapshots in the current loop. Client systems could receive and queue SBE *SnapshotFullRefresh* related messages until total number of snapshots is equal to the value of *TotNumReports* field (tag 911) of the last snapshot message received and the older incremental data queued is greater than the next sequence of the lowest value of tag 369-*LastMsgSeqNumProcessed* of all snapshots stored. They also need to ensure that no packets are missed in the process: check the sequence numbers of those packets and make sure you received the packet that contains *SequenceReset* message (the last packet of the given loop).

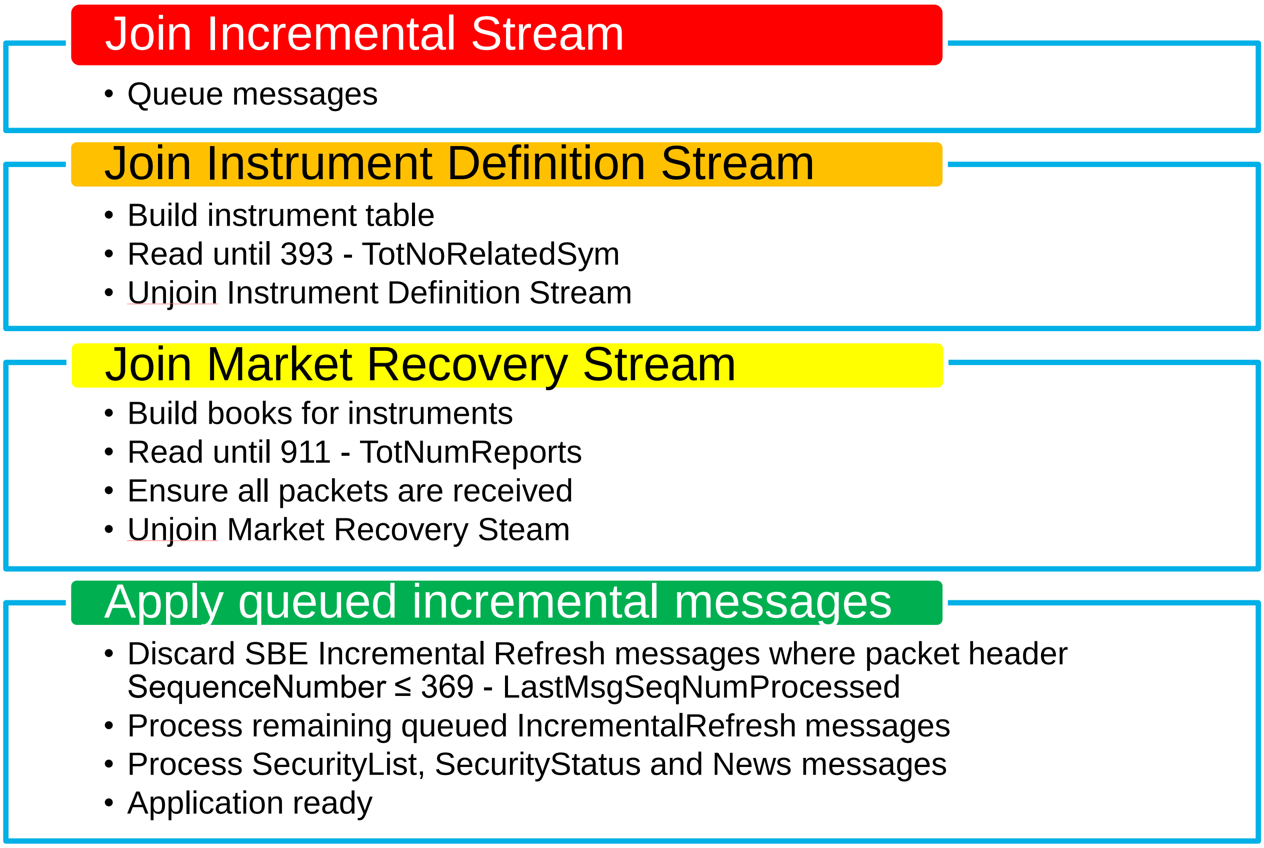
5. Unjoin the snapshot recovery stream, to avoid consuming unnecessary bandwidth.

6. Start by removing from the queue the incremental stream messages applying over related snapshots until consuming all the queued messages: discard queued messages from the incremental stream until Packet Header field *SequenceNumber* in the message has the **same value** as tag 369-*LastMsgSeqNumProcessed* in the snapshot for each instrument in the channel. The discarded messages contain information that was already included in the related snapshot message. **Do not discard messages of types: *SecurityDefinition* and *News*, as they are not reflected on the received snapshot.**
7. Start normal processing with incremental messages.

NOTE

One loop in the snapshot recovery stream can report a TotNumReports (tag 911) number that is different from the number of instruments assigned to the market data channel. Instruments with no snapshot data can be considered as having empty books.

The following diagram illustrates the graphical representation of the steps listed above.



Join Incremental Stream

- Queue messages

Join Instrument Definition Stream

- Build instrument table
- Read until 393 - TotNoRelatedSym
- Unjoin Instrument Definition Stream

Join Market Recovery Stream

- Build books for instruments
- Read until 911 - TotNumReports
- Ensure all packets are received
- Unjoin Market Recovery Steam

Apply queued incremental messages

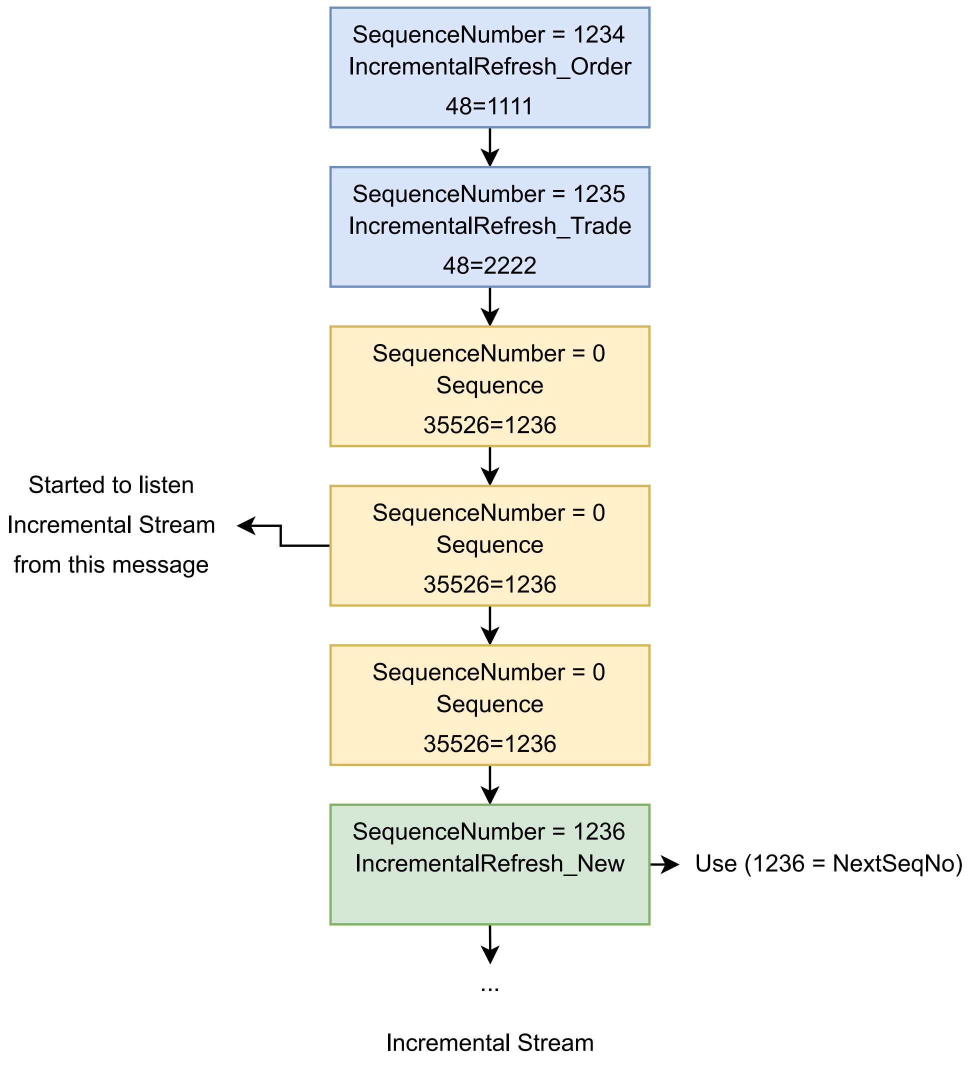
- Discard SBE Incremental Refresh messages where packet header SequenceNumber ≤ 369 - LastMsgSeqNumProcessed
- Process remaining queued IncrementalRefresh messages
- Process SecurityList, SecurityStatus and News messages
- Application ready

Figure 5.6 – Procedures for initial book synchronization.

6.5.10 Start of Day "Heartbeats" – Sequence messages

To provide clients with connectivity testing before the actual streams are activated, B3 will issue *Sequence* messages every 1 (one) second (this interval is configurable. If client systems do not receive any messages within an interval equivalent to 3 "Sequence" messages in a row, it should consider that the multicast is not active. Note that *Sequence* message is applicable to all three UDP multicast streams.

Sequence messages come in packets whose *SequenceNumber* field is 0; the *NextSeqNo* field (tag 35526) tells what the sequence number for the first business message that is will arrive in this stream. For instance, let us say that the last message in the last trading day has *SequenceNumber* = 1235; the system is inactive until morning and sends periodically "Sequence" messages with *NextSeqNo* field = 1236. The first message in the morning will have *SequenceNumber* = 1236.



6.5.11 Stream Reset Message

Client systems should be able to handle the *SequenceReset* message, which is sent by B3 in the incremental stream of any market data channel.

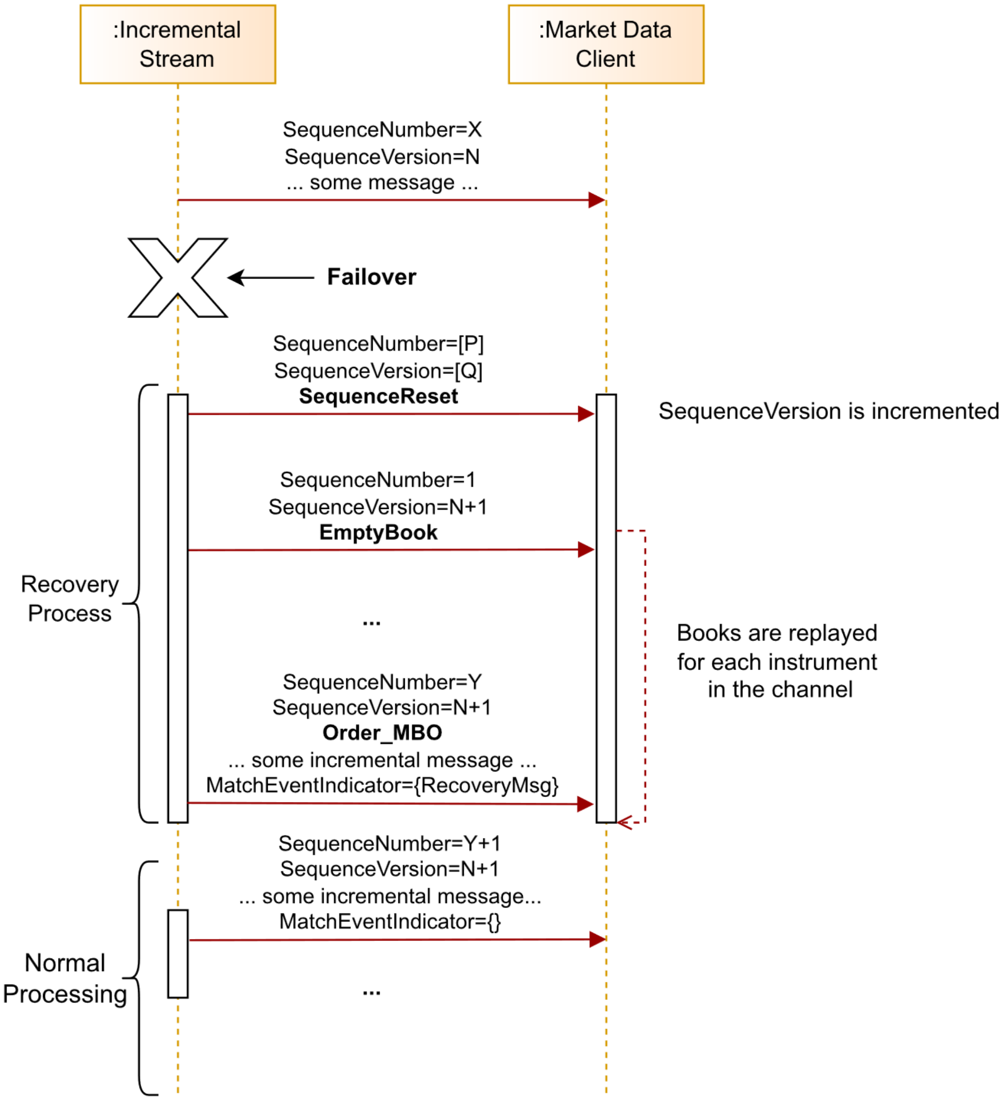
This message is issued in case of a severe failure in the exchange market data system, or regular start-up. This message will be sent individually for each site, i. e. if the failure occurs in the primary site, only that channel in the primary site is affected, likewise for the backup site.

This message is also sent on security definition and snapshot recovery streams just after a loop is finished to indicate a new loop will begin in the next packet. The stream reset is the *SequenceReset* message with *NewSeqNo* field (tag 36) = 1 (set new sequence number).

Upon receiving *SequenceReset* message in the incremental stream **during the trading session**, client systems should:

- Consider that the application sequence number has been reset and should be started from the value in *NewSeqNo* field.
- After that, replenish the book after receiving the *EmptyBook* and *Order_MBO* messages with the 5th bit set of the tag 37035-*matchEventIndicator* and the *sequenceVersion* field in the header incremented. The purpose of this behavior is to replenish the state of the order book for all active instruments in case of any missing state during the takeover process of the market data system.

The following diagram illustrates an example of the Stream Reset procedures:



[P] arbitrary sequence number (probably will be X+1)
[Q] arbitrary sequence version (probably will be the same as the last one)

6.5.12 Channel Reset

Channel Reset will provide a process to sync order books (by order) in the unlikely event of component failure, when books on the affected channel may be corrupted, or during the system initialization.

In case of component failure (or during the system initialization), B3 will issue a market data incremental message (*ChannelReset*) to notify client systems of order book reset events for all instruments of that channel.

The steps to detect the Channel Reset condition and proceed to recovery process are shown below:

1) The *ChannelReset* will be sent down the Incremental feed. It means that there has been a component failure and order books of all instruments on the channel are corrupted, or the system initialization is started.

2) The client system must empty order books of all instruments related to that impacted channel.

3) For all impacted instruments, the Snapshot Recovery data also will be removed on the Snapshot Recovery feed.

4) If the *ChannelReset* was sent during the **system initialization**, the list of instruments will be resent as well (messages *SecurityDefinition*). The last *SecurityDefinition* message will have its *LastFragment* field set to 'true'.

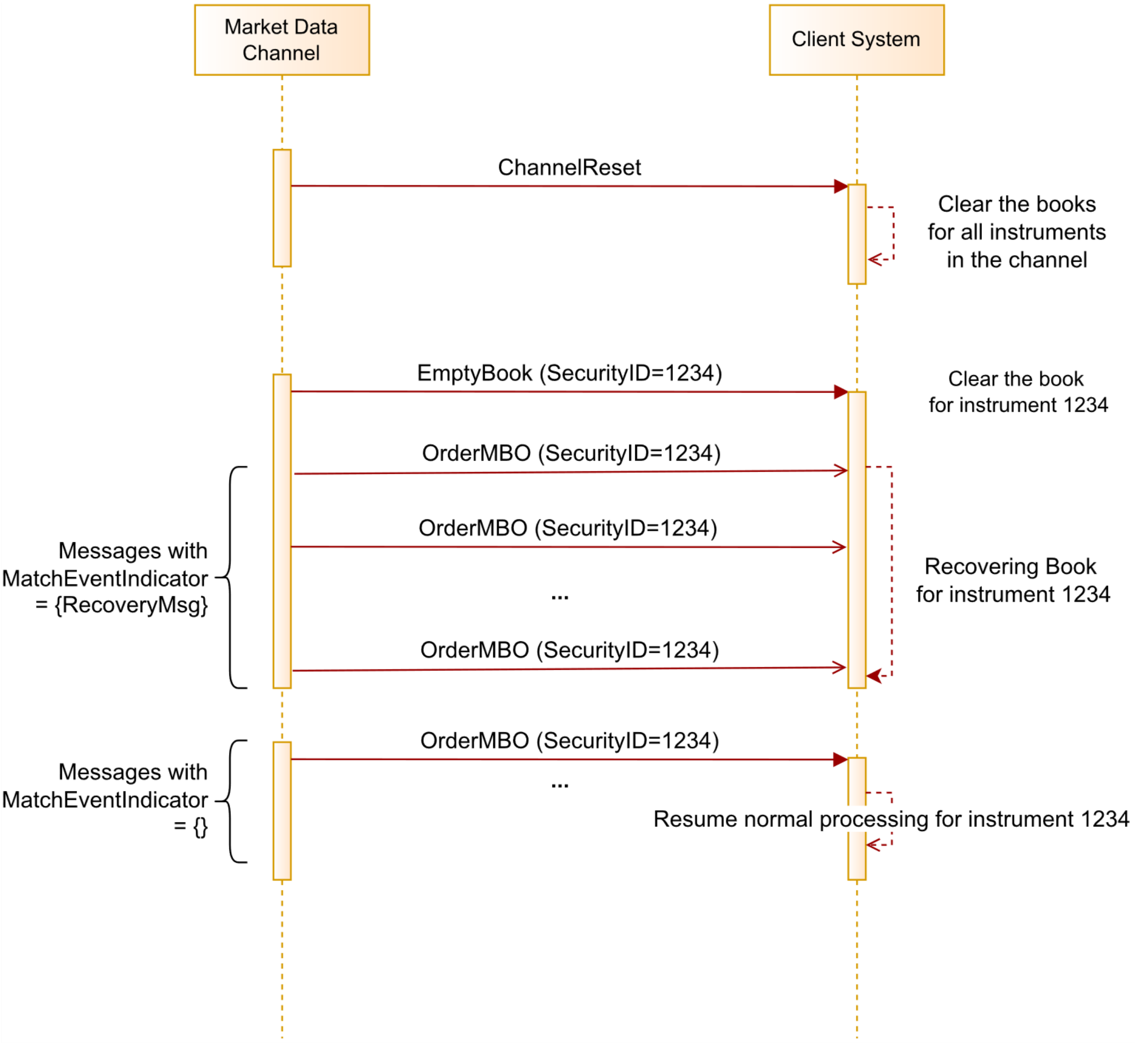
5) Incremental messages will be sent at the incremental stream to populate the order book for all instruments:

   a) The first incremental message will be an *EmptyBook,* whose *SecurityID* field indicates the instrument whose order books will be recovered (only for instruments that previously had a book). (Although the *ChannelReset* message resets the book of all instruments, the *EmptyBook* will be sent as well, to support clients that are interested only in a few instruments in the channel and filter messages by its field *SecurityID*).

   b) Then enough *Order_MBO* messages for reconstructing the order book for the instrument, with the 5th bit (*RecoveryMsg*) of the tag 37035-*matchEventIndicator* set, indicating that's a recovery message.

6) Once an order book for a specific instrument has been recovered, B3 will disseminate incremental real-time market data for that instrument (order book entry will not have the 5th bit of tag 37035-*matchEventIndicator* set), but other instruments on the channel may still be going through the recovery process.

The sequence diagram to illustrate the channel reset dynamics follows below.


Market Data Channel
Client System
ChannelReset
Clear the books for all instruments in the channel
EmptyBook (SecurityID=1234)
Clear the book for instrument 1234
OrderMBO (SecurityID=1234)
OrderMBO (SecurityID=1234)
...
OrderMBO (SecurityID=1234)
Messages with MatchEventIndicator = {RecoveryMsg}
Recovering Book for instrument 1234
OrderMBO (SecurityID=1234)
...
Messages with MatchEventIndicator = {}
Resume normal processing for instrument 1234

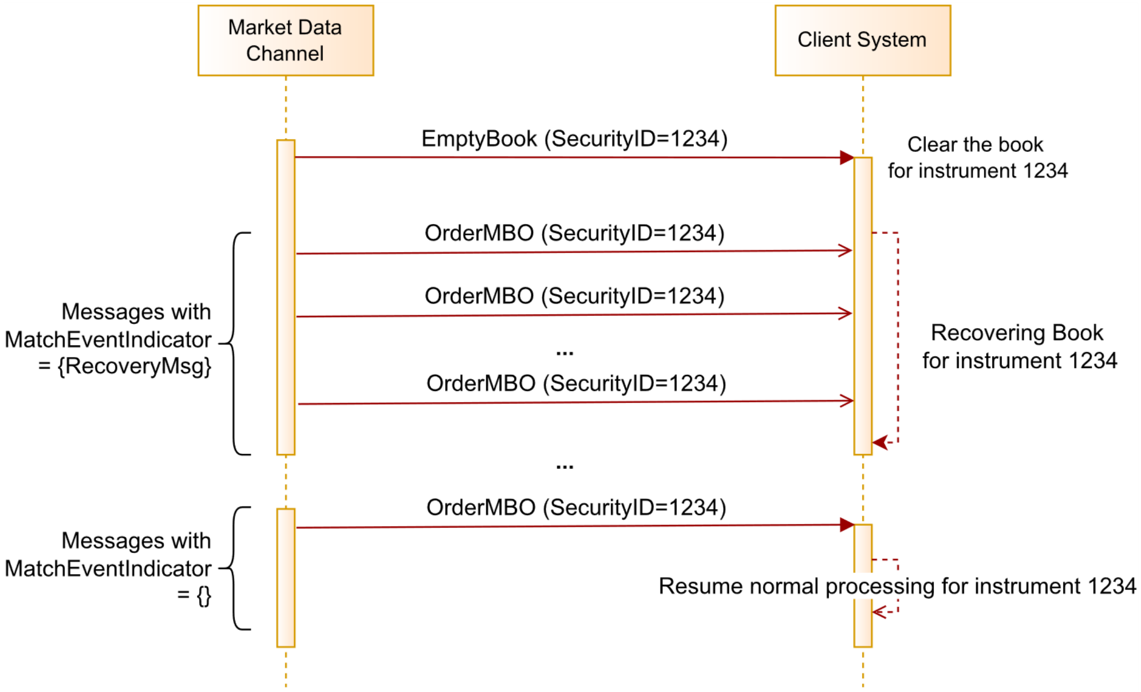
Notes:

- tag 48-*securityID*, tag 37035-*matchEventIndicator*; the value 32 (decimal) is, in binary, 10000 (the 5th bit is set – it indicates recovery).
- Security group phases, security status, statistics and band-related messages can be re-sent depending on the scenario that provoked the publication of *ChannelReset* message. It is recommended that client systems do not clear the statistics until **explicitly** receiving *SecurityGroupPhase* message with tag 1174-*SecurityTradingEvent* with value 4 (**TRADING_SESSION_CHANGE**).

6.5.13 EmptyBook (Book Reset)

EmptyBook will provide a process to sync order books (a.k.a. *Book Reset*) of specific instruments.

1) Messages will be sent at the incremental stream to populate the order book of each instrument that must be reinitialized.
    a) The first incremental message will be an *EmptyBook* whose *SecurityID* field indicates the instrument whose order books will be recovered (only for instruments that previously had a book).
    b) Then enough *OrderMBO* messages for reconstructing the order books for the instrument will be sent for every instrument, with the 5th bit (*RecoveryMsg*) of the tag 37035-*matchEventIndicator* set, indicating that's a recovery message.
2) Once an order book for a specific instrument has been recovered, B3 will disseminate incremental real-time market data for that instrument (order book entry will not have the 5th bit of tag 37035-*matchEventIndicator* set), but other instruments on the channel may still be going through the recovery process.


Market Data Channel
Client System
EmptyBook (SecurityID=1234)
Clear the book for instrument 1234
OrderMBO (SecurityID=1234)
OrderMBO (SecurityID=1234)
...
OrderMBO (SecurityID=1234)
Messages with MatchEventIndicator = {RecoveryMsg}
Recovering Book for instrument 1234
...
OrderMBO (SecurityID=1234)
Messages with MatchEventIndicator = {}
Resume normal processing for instrument 1234

Notes:

- Security status, statistics and band-related messages can be sent depending on the scenario that provoked the publication of *EmptyBook* message for the specific instrument.

- *RptSeq* is reset to one after the *EmptyBook* message related to that instrument is published. The next message after the *EmptyBook* will have the value of *RptSeq* field equals to one.

7. Recovery

7.1 Snapshot Messages

Instead of having a single, complex, lengthy *MarketDataSnapshotFullRefresh* for every instrument that has an order book or statistics, the Snapshot Recovery stream now broadcasts a series of messages for every instrument.

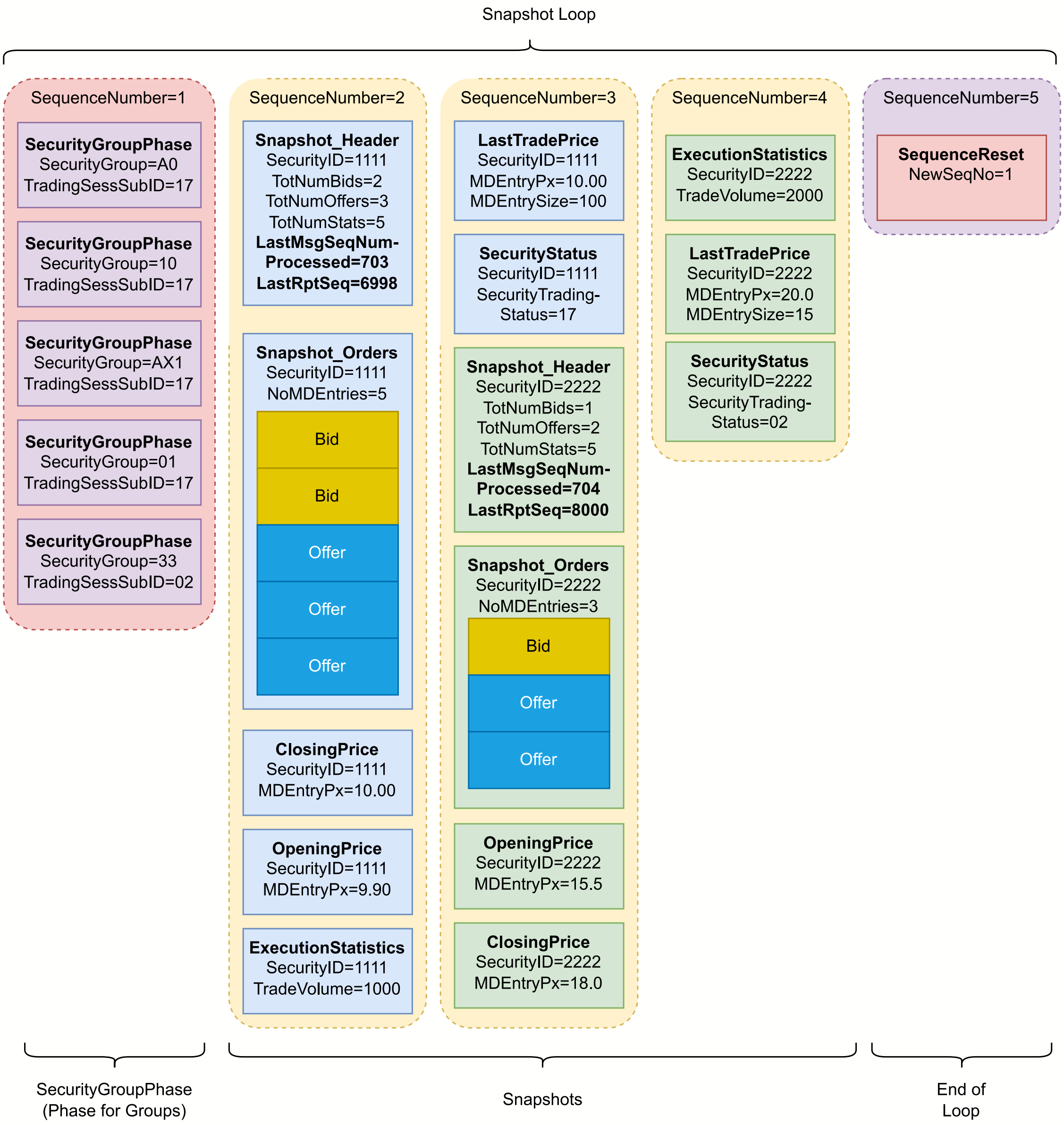
The following messages are broadcast in the Snapshot Recovery stream:

Message	Comment
Sequence	Sent periodically when the component is not initialized. The *SequenceNumber* (packet sequence number) must be ignored for this message and is usually 0.
SequenceReset	Sent at the end of every snapshot refresh loop.
SnapshotFullRefresh_Header	A header for the snapshot of a single instrument counting the messages related to a given instrument to follow.
SnapshotFullRefresh_Orders	A message containing only order book entries (bids and offers). It must not be larger than a packet. If the order books do not fit a single packet, this message will be used in the following packets until all order book entries are sent. If both order books are empty, this message is not sent for the instrument.
OpeningPrice, TheoreticalOpeningPrice, ClosingPrice, AuctionImbalance, PriceBand, QuantityBand, HighPrice, LowPrice, LastTradePrice, ExecutionStatistics, SettlementPrice, OpenInterest	For each distinct statistics, auction imbalance, band or reference price for the instrument, the corresponding message is sent. Miscellaneous statistics like TradeVolume and VWAP are sent in the *ExecutionStatistics* message. The value of the last trade price or the price and quantity values that were broadcast in the Incremental Stream in the messages "Trade", "ForwardTrade" and "LastTradePrice" are sent as the *LastTradePrice* message.
SecurityStatus	State of the instrument.
SecurityGroupPhase	Phase of the group.

Please note that some messages that are broadcast in the Incremental Stream are not broadcast in the Snapshot Stream – for instance, the *Trade* message.

For instance, only the last value of the price of a trade is sent (*LastTradePrice*); it is not possible to draw a OHLC chart (that require historical values for the trades) using only the Snapshot Stream.

News, *SecurityDefinition* and *ExecutionSummary* messages are not broadcast in the Snapshot Stream as well.



The picture above shows the *SecurityGroupPhase* messages that initialize the phase for each group, and a sequence of packets for a snapshot of 2 instruments; the first instrument (*SecurityID* = 1111) has one order book containing 5 orders (2 bids and 3 offers), four statistical values – *ClosingPrice* (Px=10.00), *OpeningPrice* (Px=9.90), *ExecutionStatistics* (TradeVolume=1000) and *LastTradePrice* (Px=10.00); and one *SecurityStatus* for the trading state of the instrument. The second instrument (security id = 2222) has one book containing 3 orders (1 bid, 2 offers), four statistical values – *OpeningPrice* (Px=15.50), *ClosingPrice* (Px=18.0), *ExecutionStatistics* (TradeVolume=2000) and *LastTradePrice* (Px=20.0); and *SecurityStatus* for the trading state of the instrument.

The messages are distributed in 5 packets. The last packet signals the end of the loop and contains only *SequenceReset* message. Its *SequenceNumber* value may be used to detect any gap in the end of loop, as it turns out to indicate the total number of packets in the loop.

Please note that usually there are more security groups that fit in a single UDP packet; the diagram above does not imply that all security group information will be broadcast in a single packet.

7.2 SnapshotRefresh_Header

This message starts a sequence of messages that describe a snapshot for a single instrument. The relevant fields are:

Field	Tag	Comment
SecurityID	48	Identification of the security.
LastMsgSeqNumProcessed	369	Sequence number of the last incremental feed packet processed, e.g., incorporated in this snapshot.
TotNumReports	911	The number of instruments that have market data for this loop. It is NOT the number of packets for this snapshot loop.
TotNumBids	37071	Total number of bid orders that constitute this snapshot.
TotNumOffers	37072	Total number of ask orders that constitute this snapshot.
TotNumStats	37070	Total number of statistics (incremental and security status messages) that constitute this snapshot.
LastRptSeq	37083	Last processed *RptSeq* (sequence number per instrument update) for this instrument. Can be used to synchronize the snapshot with the incremental feed if the client is only interested in a subset of the channel’s instruments.

For instance, in the example given in the picture above, there are two messages *SnapshotRefresh_Header*.

The first one has:

Tag	Tag Name	Value	Comments
48	securityID	1111	
369	lastMsgSeqNumProcessed	703	The *SequenceNumber* of the packet (in the Incremental feed) for the last incremental message used to update the snapshots was 703.
911	totNumReports	2	There are two instruments in loop.
37071	totNumBids	2	Two bids.
37072	totNumOffers	3	Three offers.
37070	totNumStats	5	1 *ClosingPrice*, 1 *OpeningPrice*, 1 *LastTradePrice*, 1 *ExecutionStatistics*, 1 *SecurityStatus*.
37083	lastRptSeq	6998	The *RptSeq* of the last incremental message used for updating the snapshot of instrument 1111 was 6998.

The second one has:

Tag	Tag Name	Value	Comments
48	securityID	2222	
369	lastMsgSeqNumProcessed	704	The *SequenceNumber* of the packet (in the Incremental feed) for the last incremental message used to update the snapshots was 704.
911	totNumReports	2	There are two instruments in loop.
37071	totNumBids	1	One bid.
37072	totNumOffers	2	Two offers.
37070	totNumStats	5	1 *ClosingPrice*, 1 *OpeningPrice*, 1 *LastTradePrice*, 1 *ExecutionStatistics*, 1 *SecurityStatus*.
37083	lastRptSeq	8000	The *RptSeq* of the last incremental message used for updating the snapshot of instrument 2222 was 8000.

There is a case that *SnapshotFullRefresh_Header* is the only message for the snapshot of an instrument, and it is filled with:

Tag	Tag Name	Value
48	securityID	SecurityID of the instrument
369	lastMsgSeqNumProcessed	The last processed packet sequence number of the incremental channel as of the time the snapshot was generated.
911	totNumReports	Same as the other snapshots in the loop
37071	totNumBids	**0**
37072	totNumOffers	**0**
37070	totNumStats	**0**
37083	lastRptSeq	**0**

The scenario of the case above has a rare occurrence and may happen during the a given loop when the affected instrument has just been deleted, or an *EmptyBook* message related to that instrument has just been published in the incremental stream before the publication of its snapshot to maintain the consistency of tag 911-*TotNumReports* for the entire loop.

7.3 SnapshotRefresh_Orders

This message contains a list of orders (bids and offers) that are used to recreate the order books. Once all messages must be smaller than a packet, this message – *SnapshotRefresh_Orders* – must be small as well, so it can contain just a few orders (about 21). To transmit the full set of order books, a series of *SnapshotRefresh_Orders* messages may be sent.

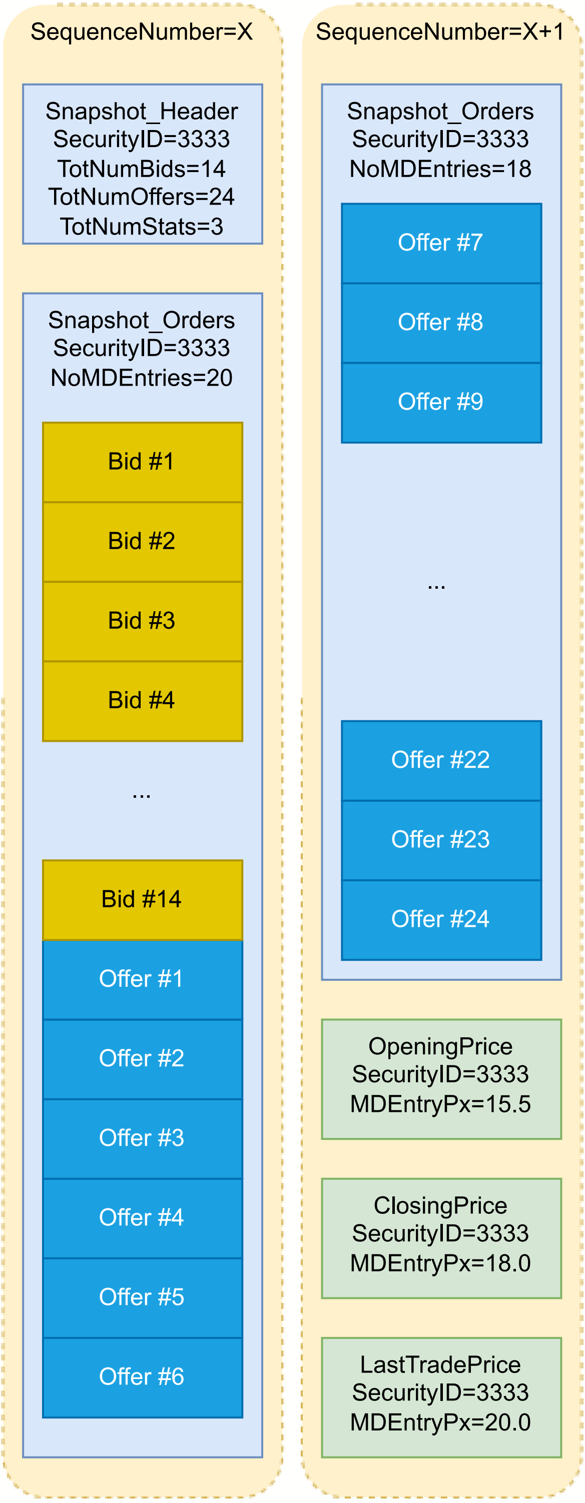
The following fields are relevant:

Field		Tag	Comment
securityID		48	Identification of the security.
noMDEntries		711	A repeating group containing a list of orders.
→	mDEntryType	269	0-Bid or 1-Offer.
→	mDEntryPositionNo	290	(MBO) Position of the order, starts with 1.
→	mDEntryPx	270	Price of the order (null for market orders)
→	mDEntrySize	271	Quantity of the order

Field		Tag	Comment
→	secondaryOrderID	198	Exchange-generated order identifier that changes for each order modification event, or quantity replenishment in disclosed orders.
→	mDInsertTimestamp	37034	The date and time when the order was inserted or re-inserted into the order book or manually altered by MktOps.
→	enteringFirm	37501	Replaces MDEntryBuyer or MDEntrySeller.
→	matchEventIndicator	37035	Identifies if this order is implied or not: Bit 4: Implied generated order.

The *MDEntryTimestamp* value of the original order is not transmitted in the snapshot message because it is not relevant anymore, only the *MDInsertTimestamp* value (instant the order was inserted or re-inserted into the order book).

In the example below, there are 38 entries in the order books (14 bids and 24 asks). The entries were sent in 2 messages *SnapshotRefresh_Orders*, one with the first 20 orders, the second one with the remaining 18 orders.



7.4 Snapshot Recovery

This recovery method should be used for large-scale data recovery (i.e. major outage or late joiners) to synchronize client systems to the latest state maintained by B3.

Client systems can use the Snapshot Recovery stream on each channel to determine the state of each book in affected channels.

Each Snapshot Recovery stream constantly loops and sends a sequence of messages, starting with SecurityGroupPhase messages for each Security Group belongs to the channel, then for each instrument, a message *SnapshotFullRefresh_Header*, then one or more *SnapshotFullRefresh_Orders* (containing the book orders), then incremental messages, representing the statistics, security status, and bands related to that instrument. It ends with a packet that contains only a *SequenceReset* message.

The Snapshot Recovery feed Is known to be valid as of a sequence number on the Incremental Market Data feed, which is found in tag 369-*LastMsgSeqNumProcessed*. This sequence number (tag 369-*LastMsgSeqNumProcessed*) is found on the *SnapshotRefresh_Header* message. Client systems will recover the most recent statistics on the Snapshot Recovery stream. Any intermediary statistics (for example trades) will not be recovered.

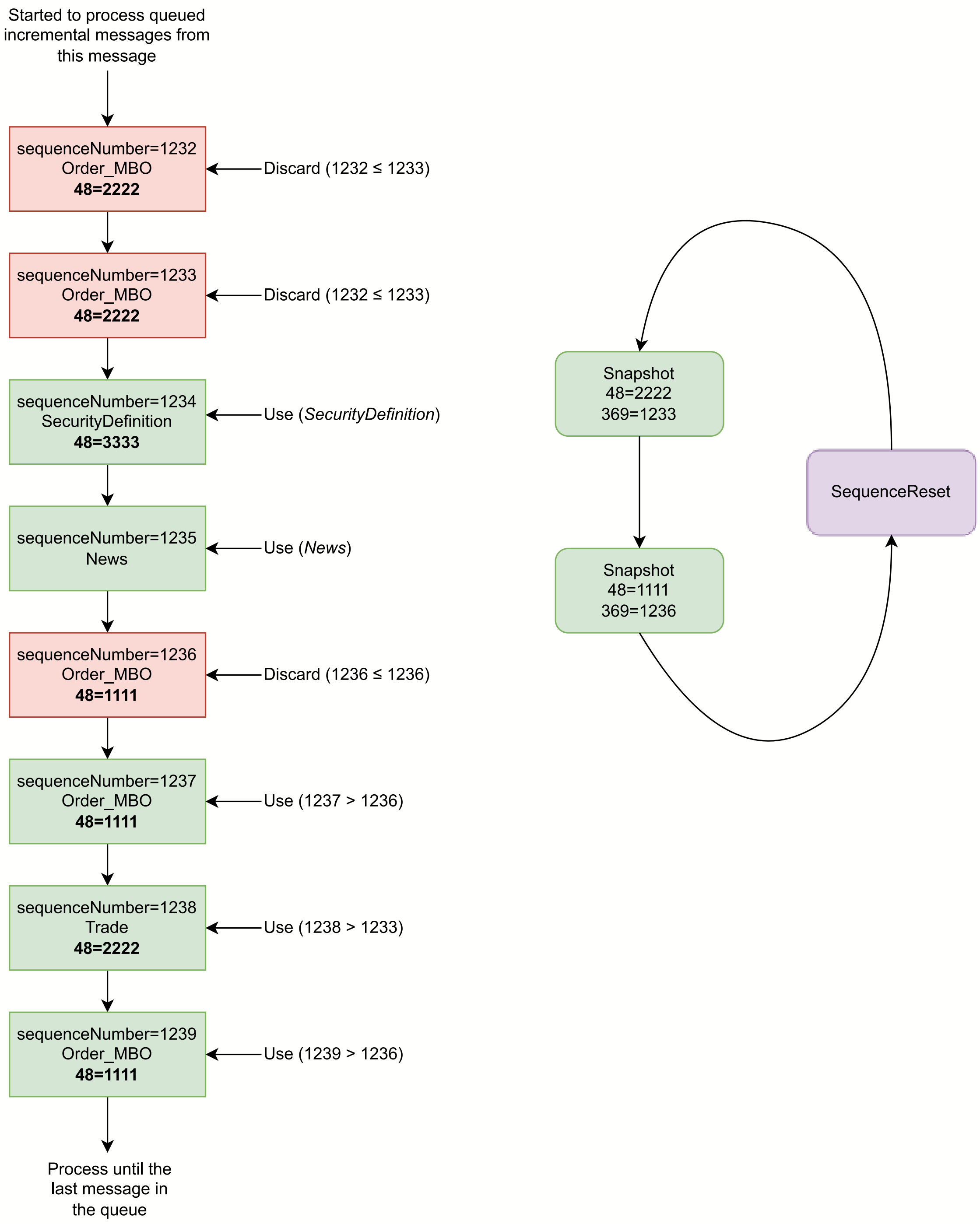
Some considerations:

1. Client systems should queue real-time data until all snapshot data is retrieved from a given channel. After this, the queued data should then be applied as necessary, where all queued incremental message with Packet Header field *SequenceNumber* less or equal than the value of tag 369-*LastMsgSeqNumProcessed* of processed snapshot should be discarded.
2. B3 strongly recommends that the Snapshot Recovery streams be used for recovery purposes only. Once client systems have retrieved recovery data, client systems should stop listening (unjoin associated multicast IP/port) to the Snapshot Recovery streams.

Recommended procedure for recovering:

1. Identify channel(s) in which the client system is out of sync.
2. Listen to and queue all the messages from incremental stream.
3. Join the multicast address/UDP port of the snapshot recovery stream until all snapshot messages have been received: monitor the header field *Sequence Number* whose value is cyclical and the tag 911-*TotNumReports* = total number of snapshots of instruments in the current loop. Client systems could receive and queue snapshots until total number of snapshots received and stored is equal to the value of tag 911-*TotNumReports* of the last snapshot message received and the *Sequence Number* of the older packet in the queue is less than of the lowest value of tag 369-*LastMsgSeqNumProcessed* of all snapshots received.

4. Start by removing from the queue the incremental stream messages applying over related snapshots until consuming all the queued messages: discard queued incremental messages from the incremental stream whose *Sequence Number* in the packet has the same value or less than tag 369-*LastMsgSeqNumProcessed* in the snapshot for each instrument in the channel. The discarded messages contain information that was already included in the related snapshot message.
5. *News* and *SecurityDefinition* messages could be processed or not. It depends on your algorithm/system. Snapshots are not based on those types of messages.
6. Unjoin the snapshot recovery stream, to avoid consuming unnecessary bandwidth.
7. Start normal processing with incremental messages.



The picture above shows the process for the Incremental and Snapshot Streams. There are two instruments in the snapshot loop (*TotNumReports* = 2); instead of showing individual SBE messages for each snapshot, we just show the *SnapshotHeader* messages for short. Starting from message 1232, the messages from the Incremental Stream are queued; the *News* and *SecurityDefinition* messages are used immediately, but the incremental messages are kept until the *SequenceNumber* value of the first message in the queue is lower than the lowest *LastMsgSeqNumProcessed value* from all snapshots received in the loop. After that, iterate from the first incremental message in the queue until the last one and apply (use) it, discarding the incremental whose sequence number is lower or equal to the *LastMsgSeqNumProcessed* value for each snapshot (Discarded messages are highlighted in pink in the picture above).

7.4.1 Using *RptSeq* and *LastRptSeq* for synchronization

If the client system is interested in selected instruments from a given channel, instead of all instruments, it can use *RptSeq* (a field from incremental messages, like *OrderMBO, Trade*, statistics and *SecurityStatus*) and *LastRptSeq* (a field from *SnapshotFullRefresh_Header* message) for synchronization instead of *SequenceNumber* and *LastMsgSeqNumProcessed*.

The *RptSeq* field (tag 83) represents a sequence number *per instrument*. By inspecting this tag, and checking the gaps between *RptSeq* values:

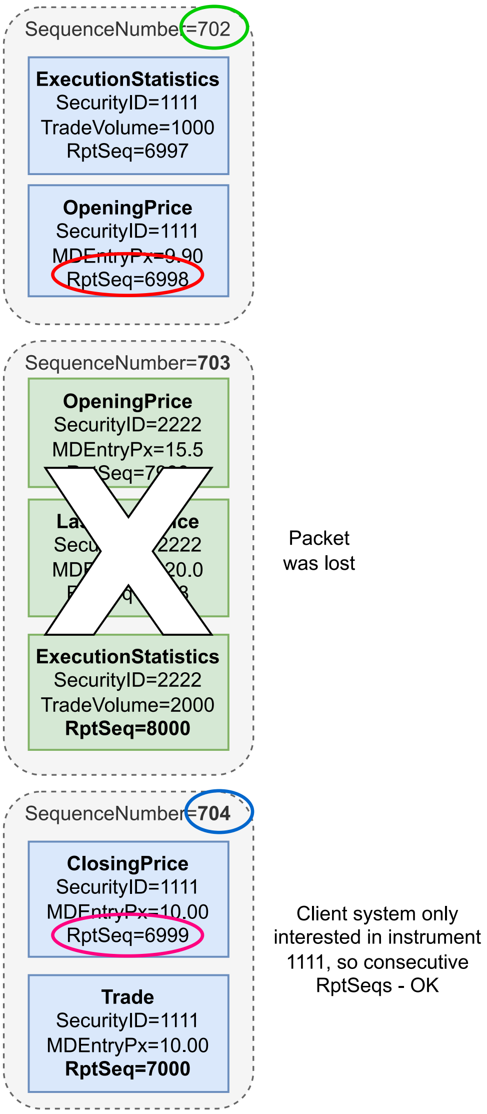
- If there is a gap, data was missed for the instrument when packet loss occurred.
- If there is no gap, the data can be used immediately, and the book for this instrument still has a correct and current state.

There is a case where *LastRptSeq* value is not present in the snapshot: illiquid instruments that haven't received any updates yet from incremental stream, explicitly related to that instrument. The only statistic present in the snapshot is the *SecurityStatus* derived from *SecurityGroupPhase* message which the instrument belongs. In this particular case, it is safe to say that the client system can process the incremental messages related to that instrument without discarding them.

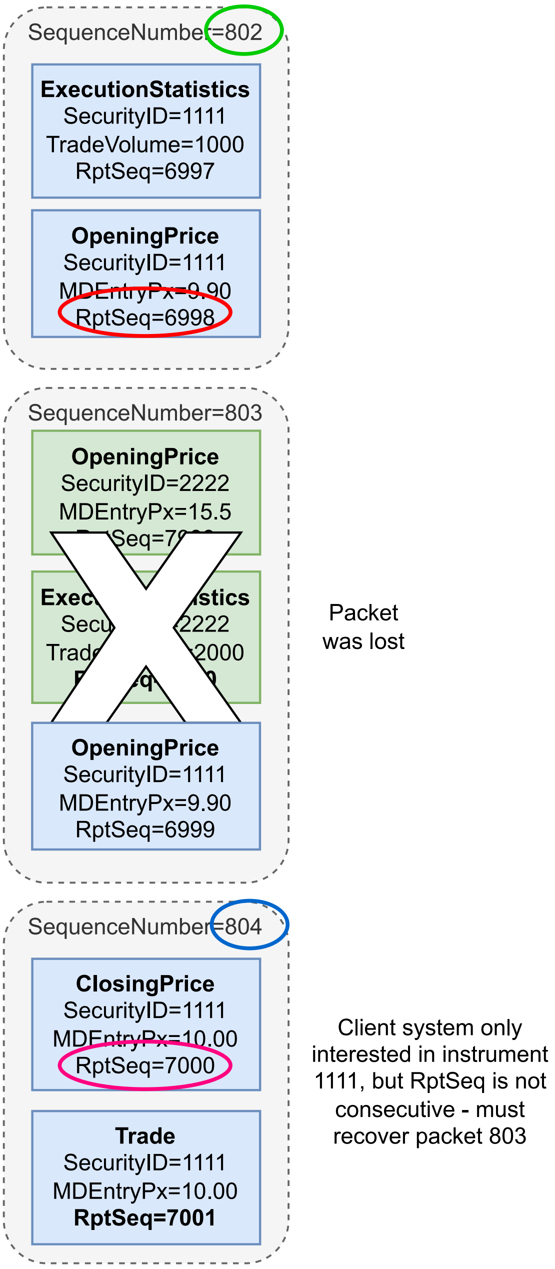
RptSeq of a given instrument is reset to one after the *EmptyBook* message related to that instrument is published. The next message after the *EmptyBook* will have the value of *RptSeq* field equals to one. It occurs in the following scenarios:

1. When the platform starts up at the beginning of the week or the secondary instance of the market data system takes over in case of failure of the primary instance.
2. There is an inconsistence in the state of the book of a given instrument, and surveillance team tries to work around it, resending the state of that instrument (*EmptyBook* + *Order* messages).

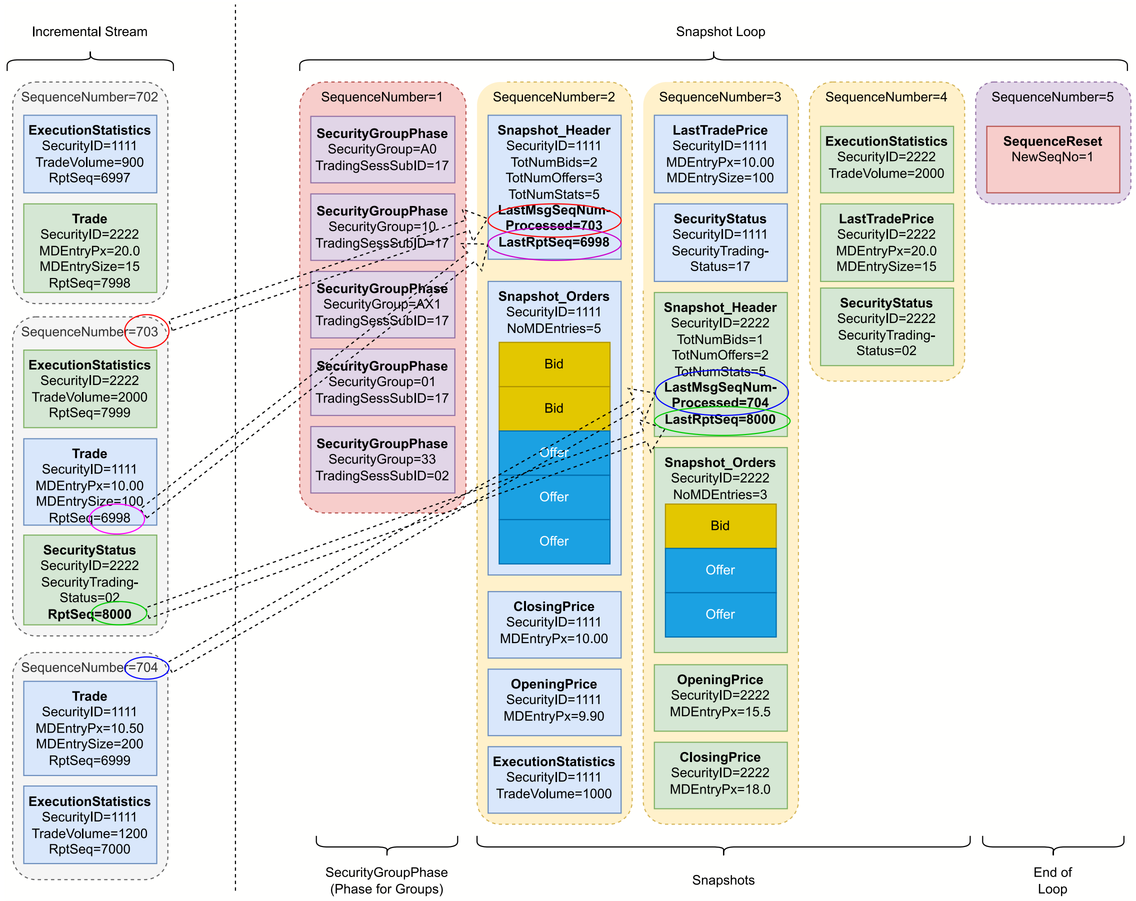
In the diagram below, the client system is interested only in instrument 1111, and the packet 703 was lost. Because there is no gap in *RptSeq* for the instrument 1111, there is no need to recover packet 703.



In the diagram below, the client system is interested only in instrument 1111, and the packet 803 was lost. Because there is a gap in *RptSeq* for the instrument 1111 (it went from 6998 to 7000), the packet 803 must be recovered.


SequenceNumber=802
ExecutionStatistics
SecurityID=1111
TradeVolume=1000
RptSeq=6997
OpeningPrice
SecurityID=1111
MDEntryPx=9.90
RptSeq=6998
SequenceNumber=803
OpeningPrice
SecurityID=2222
MDEntryPx=15.5
ExecutionStatistics
SecurityID=2222
TradeVolume=2000
OpeningPrice
SecurityID=1111
MDEntryPx=9.90
RptSeq=6999
Packet was lost
SequenceNumber=804
ClosingPrice
SecurityID=1111
MDEntryPx=10.00
RptSeq=7000
Trade
SecurityID=1111
MDEntryPx=10.00
RptSeq=7001
Client system only interested in instrument 1111, but RptSeq is not consecutive - must recover packet 803

The *LastRptSeq* field (tag 37083) from the *SnapshotFullRefresh* message represents the value of *RptSeq* for the last incremental message that updated the snapshot for the instrument.


Incremental Stream
Snapshot Loop
SequenceNumber=702
ExecutionStatistics SecurityID=1111 TradeVolume=900 RptSeq=6997
Trade SecurityID=2222 MDEntryPx=20.0 MDEntrySize=15 RptSeq=7998
SequenceNumber=703
ExecutionStatistics SecurityID=2222 TradeVolume=2000 RptSeq=7999
Trade SecurityID=1111 MDEntryPx=10.00 MDEntrySize=100 RptSeq=6998
SecurityStatus SecurityID=2222 SecurityTrading-Status=02 RptSeq=8000
SequenceNumber=704
Trade SecurityID=1111 MDEntryPx=10.50 MDEntrySize=200 RptSeq=6999
ExecutionStatistics SecurityID=1111 TradeVolume=1200 RptSeq=7000
SequenceNumber=1
SecurityGroupPhase SecurityGroup=A0 TradingSessSubID=17
SecurityGroupPhase SecurityGroup=10 TradingSessSubID=17
SecurityGroupPhase SecurityGroup=AX1 TradingSessSubID=17
SecurityGroupPhase SecurityGroup=01 TradingSessSubID=17
SecurityGroupPhase SecurityGroup=33 TradingSessSubID=02
SequenceNumber=2
Snapshot_Header SecurityID=1111 TotNumBids=2 TotNumOffers=3 TotNumStats=5 LastMsgSeqNum-Processed=703 LastRptSeq=6998
Snapshot_Orders SecurityID=1111 NoMDEntries=5
Bid
Bid
Offer
Offer
Offer
ClosingPrice SecurityID=1111 MDEntryPx=10.00
OpeningPrice SecurityID=1111 MDEntryPx=9.90
ExecutionStatistics SecurityID=1111 TradeVolume=1000
SequenceNumber=3
LastTradePrice SecurityID=1111 MDEntryPx=10.00 MDEntrySize=100
SecurityStatus SecurityID=1111 SecurityTrading-Status=17
Snapshot_Header SecurityID=2222 TotNumBids=1 TotNumOffers=2 TotNumStats=5 LastMsgSeqNum-Processed=704 LastRptSeq=8000
Snapshot_Orders SecurityID=2222 NoMDEntries=3
Bid
Offer
Offer
OpeningPrice SecurityID=2222 MDEntryPx=15.5
ClosingPrice SecurityID=2222 MDEntryPx=18.0
SequenceNumber=4
ExecutionStatistics SecurityID=2222 TradeVolume=2000
LastTradePrice SecurityID=2222 MDEntryPx=20.0 MDEntrySize=15
SecurityStatus SecurityID=2222 SecurityTrading-Status=02
SequenceNumber=5
SequenceReset NewSeqNo=1
SecurityGroupPhase (Phase for Groups)
Snapshots
End of Loop

In the diagram above, the snapshot for the instrument 1111 was last updated with a SBE incremental message (*RptSeq* = 6998) that came in the packet (*SequenceNumber* = 703), and the snapshot for the instrument 2222 was last updated with a SBE incremental message (*RptSeq* = 8000) that came in the packet (*SequenceNumber* = 703) but has just been serialized and published after receiving incremental packet (*SequenceNumber* = 704) that incidentally does not have any market data for instrument 2222.

Here follows a variation of the synchronization algorithm, using *RptSeq/LastRptSeq* instead of *SequenceNumber/LastMsgSeqNumProcessed*, if the client system is interested in a few instruments of the channel. (The recovery time is not speed up because the snapshot cycle time is not changed). It is given for supplementary purposes only, because in practice *RptSeq* is most useful for detecting gaps in incremental messages for a few instruments, not for snapshot synchronization.

- Identify channel(s) in which the client system is out of sync.

- Listen to and queue the messages from incremental stream the client system is interested on (use *SecurityID* and *RptSeq* fields for filter purposes).

- Join the multicast address/UDP port of the snapshot recovery stream until all snapshot messages have been received, ignoring the instruments the client system is not interested: monitor the *SnapshotFullRefresh_Header* message fields *SecurityID* and the field *LastRptSeq* (and the field 911–*TotNumReports* – to check the total number of snapshots in the current loop). Client systems could receive and queue snapshots until total number of snapshots received is equal to the value of tag 911–*TotNumReports* of the last snapshot message received.

- Start by removing from the queue the incremental stream messages applying over related snapshots until consuming all the queued messages: discard queued incremental messages from the incremental stream whose *RptSeq* field has the same value or less than tag 37083–*LastRptSeq* in the snapshot for the instrument of interest.

- The discarded messages contain information that was already included in the related snapshot message.

- *SecurityDefinition* and *News* messages have no *RptSeq* and are not reflected on the received snapshot, so they must be processed immediately, or discarded if the client system is not interested on them.

- Unjoin the snapshot recovery stream, to avoid consuming unnecessary bandwidth.

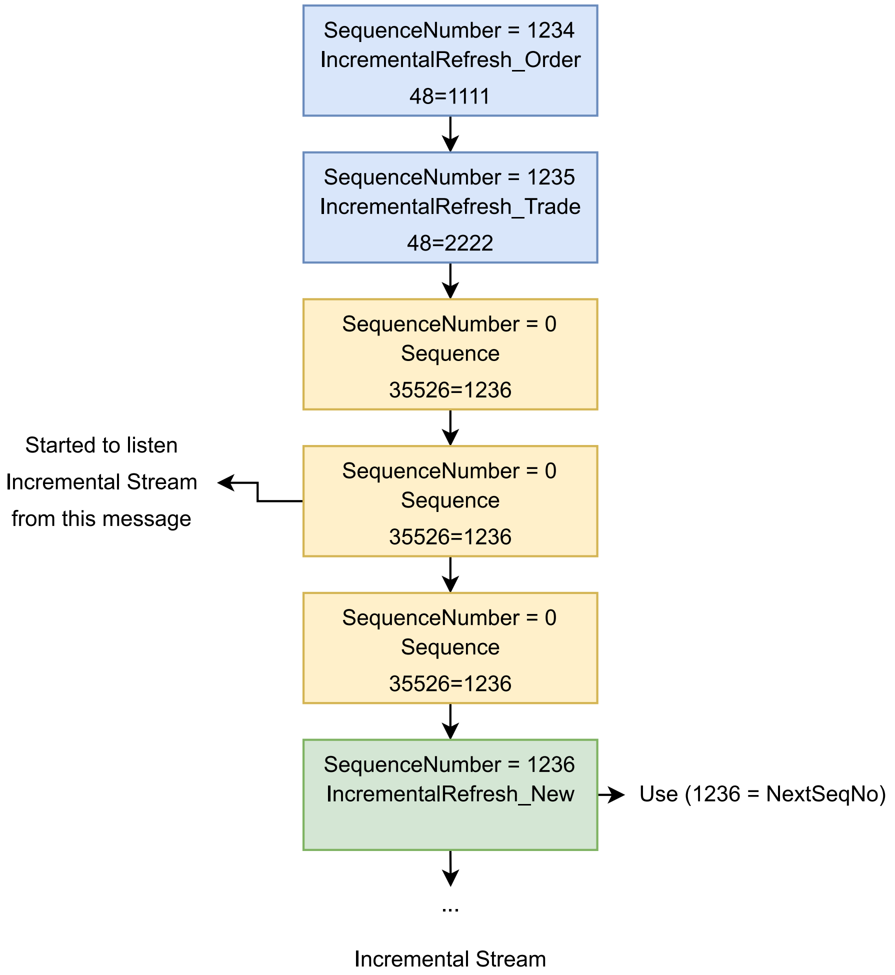
- Start normal processing with incremental messages.

7.5 Sequence Message

The *Sequence* message works like a heartbeat and has a special field *NextSeqNo* (tag 35526) that can be useful in the snapshot recovery process. This field contains the next value of *SequenceNumber* for a packet that is not a heartbeat.

The value of *SequenceNumber* field (in Packet Header) for this message is 0 (**zero**) and must be ignored (it is not meant to be saved or recovered). This message only tells the client application that the system is idle, but still running.

Suppose that the client program is trying to synchronize with an idle market data channel that is currently sending *Sequence* messages in the incremental stream. Using the *NextSeqNo* field, the algorithm can get the *SequenceNumber* for the next incremental message that will be sent in the incremental stream. In the picture below, the message 1236 and subsequent messages can be used.



Sequence messages also are published on the Instrument Definition and Snapshot streams and tells the client application that the recovery system is idle, but still running. In this situation, client application needs to wait until *ChannelReset* message is published in the Incremental stream, and after that, book-related messages are replenished. Recovery system will also be fuelled with those messages and will start to replay them on Instrument Definition and Snapshot streams.

8. Market Data Entry Types

This section lists the market data entry types supported in the B3 feed. Each entry type contains relevant trading information such as order book, trades, and statistical data. Note that availability of each of these types is subject to the trading platform functionality for Equities and Derivatives.

For every *MDEntryType* as defined by FIX, there is a different SBE Message:

MDEntryType	SBE Message	Description	Comment
0 or 1	Order_MBO	New / Change Bid / Offer	The book on the buy (bid) or sell (offer) side for the security. The book is always order-depth based (each

MDEntryType	SBE Message	Description	Comment
			individual order appears as a separate book entry). This message includes or changes an order in the order book.
0 or 1	DeleteOrder_MBO	Delete Bid / Offer	This message deletes a single order in the order book.
0 or 1	MassDeleteOrders_MBO	Delete From / Delete Thru Bid / Offer	This message deletes a set of orders in the order book.
2	Trade	Trade	The completed trades for the security.
2	ForwardTrade	Trade for a "Termo" (Forward) instrument	Specialized version of "Trade" that has two additional fields for forward ("termo") instruments.
2	LastTradePrice	Last Trade Price	A statistical message that reports the last trade price and quantity. Incremental stream: it is sent very rarely (the Trade message already conveys the last trade price and quantity). Snapshot Stream: it is sent for each instrument.
4	OpeningPrice	Opening Price	The opening price of the security (first trade).
4	TheoreticalOpeningPrice	Theoretical Opening Price	Theoretical opening price, calculated and updated based on the orders presented in the book during every auction including the pre-opening / pre-closing auction.
5	ClosingPrice	Closing price	The closing price of the security (previous day's last trade).
6	SettlementPrice	Settlement price	Settlement price or the previous day's adjusted closing price.
7	HighPrice	Trading Session High Price	The highest price traded for the security in the trading session.
8	LowPrice	Trading Session Low Price	The lowest price traded for the security in the trading session.
A	AuctionImbalance	Imbalance	Information related to imbalance of auctions such as side and quantity.
C	OpenInterest	Open Interest	Total number of contracts in a commodity or options market that are still open.
J	EmptyBook	Empty Book	Indicates that the order book for the related instrument (or for all instruments of the channel) is no longer valid.
g	PriceBand	Price band	Contains price banding information.
h	QuantityBand	Quantity band	Contains quantity banding information.
9	ExecutionStatistics	Execution Statistics	Contains some statistics (like VWAP and TradeVolume) from the trading event.
s	ExecutionSummary	Execution Summary	Summarizes an event that resulted in one or more trades. It is sent as the first message

MDEntryType	SBE Message	Description	Comment
			from all market data messages that resulted from that event.

9. Trading Event Processing – *matchEventIndicator*

All incremental messages have a field – *matchEventIndicator* (tag 37035) – that is a set of flags indicating the end of a trading event (**EndOfEvent**, bit 7) and/or if this message is a retransmission (**RecoveryMsg**, bit 5) and also for order and trade/trade bust messages: the implied indicator (that informs if the related order was generated from an implied mechanism or if the related trade is the result of a match that involves an implied order). The "RecoveryMsg" flag is sent only for *Order*, *ChannelReset* and *EmptyBook* messages. There are other flags that are currently reserved.

Let us review the example given in chapter 5.4 before.

For instance, let us say that an ask order was sent to the Matching Engine that matched the resting order whose *secondaryOrderID* is 727042222275. The resulting market data could be expressed in the former MBO diffusion as the following 35=X FIX message – this represents the first trade of an instrument in the day, followed by the opening price, high price, low price, VWAP and the matching (removal) of the resting order of *secondaryOrderID* = 727042222275 (that is represented using the tag 37, not the tag 198).

```
35=X|268=6|
   279=0|269=2|288=85|289=88|1003=10|271=1|270=65265|48=100988|
   279=0|269=4|270=65265|48=100988|
   279=0|269=7|270=65265|48=100988|
   279=0|269=8|270=65265|48=100988|
   279=0|269=9|270=65265|48=100988|
   279=2|269=0|48=100988|37=727042222275|290=1|
```

In *Binary UMDF*, we will send two additional messages that will help to make faster trading decisions. The first one is *ExecutionSummary* and will be sent as the first message. The second one is *ExecutionStatistics* and will report some trade statistics that apply to the trade execution, like VWAP price, Trade Volume, and number of the trades in the session:

Message Name	FIX equivalent	Template ID	Message Size (bytes)	SBE message contents
ExecutionSummary		55	64	`SecurityID=100988, LastPx=65265, FillQty=1, CxlQty=0, TradedHiddenQty=0, AggressorSide=1`
Trade	269=2, 279=0	53	52	`SecurityID: 100988, MDEntryBuyer: 85, MDEntrySeller: 88, TradeID: 10, MDEntrySize: 1, MDEntryPx: 65265, MatchEventIndicator: {}`
OpeningPrice	269=4, 279=0	15	40	`SecurityID: 100988, MDEntryPx: 65265, MatchEventIndicator: {}`

Message Name	FIX equivalent	Template ID	Message Size (bytes)	SBE message contents
HighPrice	269=7, 279=0	24	32	`SecurityID: 100988, MDEntryPx: 65265, MatchEventIndicator: {}`
LowPrice	269=8, 279=0	25	32	`SecurityID: 100988, MDEntryPx: 65265, MatchEventIndicator: {}`
DeleteOrder	269=0, 279=2	51	40	`SecurityID: 100988, SecondaryOrderID: 727042222275, MDEntryPositionNo: 1, MatchEventIndicator: {}`
ExecutionStatistics	269=9	55	64	`SecurityID: 100988, TradeVolume: 1, NumberOfTrades: 100, VwapPx: 65265, MatchEventIndicator: {EndOfEvent}`

The *matchEventIndicator* field (tag 37035) is a 'set', that is a set of bits representing a collection of non-exclusive choices. All messages belong to the same "trading event". The last message is marked with **EndOfEvent**, as you can see in the table below:

Decimal Value	RecoveryMsg (bit 5)	EndOfEvent (bit 7)
0	0	0
128	0	1 (i.e., 2^7, or 1 << 7)
32	1 (i.e., 2^5, or 1 << 5)	0
160 = 128 | 32	1 (i.e., 2^5, or 1 << 5)	1 (i.e., 2^7, or 1 << 7)

If the messages were repeated due to a recovery process, the order messages would be marked with the "**RecoveryMsg**" bit as well, so the values for the tag 37035 would be 32 ({RecoveryMsg}) and 160 ({RecoveryMsg, EndOfEvent}) respectively. The other messages (statistics and trading state) report the current status of the instrument, so they are not marked with "RecoveryMsg" flag.

10. Order Book

There are two ways to view the same order book:

- By Price Level and Order Priority
- By Position Number

The Matching Engine works with order books by price level and order priority; the position number is inferred. MBO uses a view of the book by position number.

10.1 Book: Position Number

Order books are internally kept in price-level and priority order, but the position number is synthesized by Market Data components to relay this information. The position number starts by 1.

The examples given above will be shown with the Position Numbers (tag 290-*MDEntryPositionNo*).

Example 1: Order books have only "Limit orders":

Bid				Ask		
Price	**Secondary OrderID**	**Quantity**	**Position**	**Price**	**Secondary OrderID**	**Quantity**
7.28	330	100	**1**	7.31	800	300
7.28	700	200	**2**	7.32	320	100
7.20	100	100	**3**	7.32	600	200
			4	7.50	200	100
			5	7.52	300	400

Example 2: A book has some "market orders":

Bid					Ask	
Price	**Secondary OrderID**	**Quantity**	**Position**	**Price**	**Secondary OrderID**	**Quantity**
			1	N/A	900	300
			2	N/A	920	400
			3	7.32	600	200
			4	7.50	200	100
			5	7.52	300	400

11. Incremental Order Book Management

Order books received via the SBE feed are incremental, i. e. changes to the book are relayed on individual messages providing "deltas" of the previous state of the book.

The actions to be executed by the client system receiving the incremental message are determined by the type of the incremental message.

MBO	Action	FIX Equivalent
Order	Add or /modify an order.	279=0 (NEW) or 1 (CHANGE), 269=0 (BID) or 1 (OFFER)
DeleteOrder	Delete a single existing order.	279=2 (DELETE), 269=0 (BID) or 1 (OFFER)
MassDeleteOrders	Delete a set of orders (DELETE FROM, DELETE THRU).	3 (DELETE_THRU) or 4 (DELETE_FROM) 269=0 (BID) or 1 (OFFER)
EmptyBook	Empty the books (bid and ask) and clear the statistics of an instrument.	269=J

11.1 Incremental Book Management–- MBO

Order books received via the MBO feed are incremental, i.e. changes to the book are relayed on individual messages providing "deltas" of the previous state of the book.

The actions to be executed by the client system receiving the incremental message are determined by tag *279-MDUpdateAction*, whose value carries an instruction that can be either add, delete, change, delete_from or delete_thru. The items in the order book that are affected by the action stated in tag 279 are stated in tag 290-*MDEntryPositionNo*, which contains a position in the order book.

For bid or offer book entries, the deletion is based on the entry's position (tag *290-MDEntryPositionNo*). For example, assume ten bids for a security. Adding a bid with tag 290-*MDEntryPositionNo* = 4 requires the receiver to shift down other Market Data Entries, i.e. the Market Data Entry in the 4th display position will shift to the 5th, the 5th shifts to the 6th, etc. until the 10th shifts to the 11th. B3 will not send a modification of all entries in the 4th through 10th positions just to update the tag 290-*MDEntryPositionNo* field; the receiver of the market data must infer the change.

Similarly, deleting a Market Data Entry in the 7th position causes the 8th Market Data Entry to move into the 7th position, the 9th to shift into the 8th position, etc. B3 will not issue a change action to modify the position of an entry in the order book. Change updates are only sent when a value applicable to a specific tag 290-*MDEntryPositionNo* – such as total quantity or number of orders – changes.

11.1.1 Order depth book

Order depth book contains order by order information, where each entry represents an individual order. For example, this is how an order-depth book looks like:

Bid			Offer		
PosNo	**Size**	**Px**	**Px**	**Size**	**Po**
1	5000	10.58	11.03	7000	1
2	4000	10.58	11.03	2000	2
3	3000	10.57	11.05	1000	3
4	4000	10.54			4

One entry per order: same price on more than one entry.

B3 provides the full depth of the book for order-depth book, i.e. the client will always receive updates for all the orders that are in the order book, even if it is the last one (worst price).

In general, if a trade occurs, B3 will send a delete or change data block to update the book. The trade data block itself is not used to update the order book.

Below are the messages sent for order depth book update:

11.1.1.1 Order_MBO (new or change existing order)

Tag	Tag Name	Presence	Comments
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the *SecurityDefinition* message in Market Data feed.
37035	matchEventIndicator	R	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 4: Implied generated order. Bit 5: Message is sent during recovery process. Bit 7: Last message for the event.
279	mDUpdateAction	R	Update Action (NEW, CHANGE).
269	mDEntryType	R	Entry Type (BID or OFFER).
270	mDEntryPx	O	Price per share or contract. Conditionally required if the order type requires a price (not market orders).
271	mDEntrySize	R	Displayed quantity or volume represented by the Market Data Entry.
290	mDEntryPositionNo	R	Display position of a bid or offer, numbered from most competitive to least competitive, per market side, beginning with 1.
37501	enteringFirm	O	Identifies the broker firm.
37034	mDInsertTimestamp	R	The date and time when the order was inserted or re-inserted into the order book or manually altered by MktOps.
198	secondaryOrderID	R	Exchange-generated order identifier that changes for each order modification event, or quantity replenishment in disclosed orders.
37033	mDEntryTimestamp	R	Timestamp when the order event occurred.

11.1.1.2 DeleteOrder_MBO (delete existing order)

Tag	Tag Name	Presence	Comments
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 4: Implied generated order. Bit 7: Last message for the event.
269	mDEntryType	R	Entry Type (BID or OFFER).
290	mDEntryPositionNo	R	Display position of a bid or offer, numbered from most competitive to least competitive, per market side, beginning with 1.
271	mDEntrySize	O	Quantity of the deleted order. Absent if the deletion is the result of a matching event.
198	secondaryOrderID	R	Exchange-generated order identifier that changes for each order modification event, or quantity replenishment in disclosed orders.
37033	mDEntryTimestamp	R	Timestamp when the order event occurred.

11.1.1.3 MassDeleteOrders (delete several orders in given side of the book)

Tag	Tag Name	Presence	Comments
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
279	mDUpdateAction	R	Update Action (DELETE_FROM, DELETE_THRU).
269	mDEntryType	R	Entry Type (BID or OFFER).
290	mDEntryPositionNo	R	Display position of a bid or offer where orders will be deleted (up or down from this position).
37033	mDEntryTimestamp	R	Timestamp when the mass delete order event occurred.

For more details, please check the *Binary UMDF Message Reference* document.

11.1.2 Delete From

The message is *MassDeleteOrders* with *MDUpdateAction* = **DELETE_FROM**.

This message allows for more efficient book management by providing an extension to tag 279-*MDUpdateAction* that allows the deletion of all orders from a certain position.

When an order is entered that causes several executions and sweeps the order book, causing several price levels to be deleted, instead of sending deletions for several price levels, the *MDUpdateAction* "*Delete From*" (*tag 279* = 4) is used, along with the tag *MDEntryType* (tag 269 = BID–- '0' for the BID book, OFFER–- '1' for the ASK or OFFER book). It indicates that all positions from the position stated in tag 290-*MDEntryPositionNo* up until position 1 must be deleted. This will cause the market data entry that was in position *MDEntryPositionNo* + 1 to be the first position now.

The following example of an order-depth book illustrates this behavior:

Bid			Offer		
PosNo	**Size**	**Px**	**Px**	**Size**	**PosNo**
1	5000	10.58	11.03	7000	**1**
2	4000	10.58	11.03	2000	**2**
3	3000	10.57	11.05	1000	**3**
4	4000	10.54			**4**

A sell order is sent with quantity 12000 and price 10.57, which executes against the 3 existing buy orders in the order book. B3 will send an incremental market data message with the following characteristics:

MassDeleteOrders	
MDUpdateAction	DELETE_FROM (4)
MDEntryType	BID (0)
MDEntryPositionNo	3

The resulting order book as displayed by the client system should be:

Bid			Offer		
PosNo	**Size**	**Px**	**Px**	**Size**	**PosNo**
1	4000	10.54	11.03	7000	**1**
2			11.03	2000	**2**
3			11.05	1000	**3**

11.1.3 Delete Thru

The message is *MassDeleteOrders* with *MDUpdateAction* = **DELETE_THRU**.

This message allows for more efficient order book management by providing an extension to tag 279-*MDUpdateAction* allowing delete through a position. All entries of related side of the order book (Bid or Offer) are deleted; the value of tag 290-*MDEntryPositionNo* is always 1, and it does not remove the statistics.

The following example of an order-depth book illustrates this behavior:

Bid			Offer		
PosNo	**Size**	**Px**	**Px**	**Size**	**PosNo**
1	5000	10.58	11.03	7000	**1**
2	4000	10.58	11.03	2000	**2**
3	3000	10.57	11.05	1000	**3**
4	4000	10.54			**4**

The market supervisor decided to cancel all bid entries, so B3 will send an incremental market data message with the following characteristics:

MassDeleteOrders	
MDUpdateAction	DELETE_THRU (3)
MDEntryType	BID (0)
MDEntryPositionNo	1

The resulting order book as displayed by the client system should be:

Bid			Offer		
PosNo	**Size**	**Px**	**Px**	**Size**	**PosNo**
1			11.03	7000	1
2			11.03	2000	2
3			11.05	1000	3

12. Trade and real-time statistical data

There is a number of statistics (market data events) which are related to changes in an order book but these events are not used to update the order book. The following types of information fit this category: last best price, trade, high/low trade price, and pre-opening statistics. These events describe the behavior of the trading sessions and allow a user to know when the market is moving in a certain direction and provide historical information on how the market has performed.

The following messages are related to trade and statistical data:

MBO	Action	FIX Equivalent
OpeningPrice **TheoreticalOpeningPrice** **ClosingPrice**	Relays the information of Opening, Theoretical Opening, and Closing Prices.	279=0, 269=4 279=0, 269=4 279=0, 269=5
Trade **ForwardTrade**	Describes a single trade. (ForwardTrade is used with forward (“termo”) instruments).	279=0, 269=2
LastTradePrice	The value of the last trade.	279=0, 269=2, 277=U
ExecutionSummary	Relays execution summary information on one instrument.	
ExecutionStatistics	Relays execution summary statistics information on one instrument.	279=0, 269=9
AuctionImbalance	Relays auction details on one instrument.	279=0, 269=A
PriceBand **QuantityBand**	Disseminates price and quantity bands information on one instrument.	279=0, 269=g 279=0, 269=h
HighPrice **LowPrice**	Trading session high price and low price.	269=7 / 8
EmptyBook	Empty the order book for one instrument.	269=J

For more details on each of the messages, please refer to Binary UMDF Message Reference document.

Prices with different decimals!

Closing prices differ from other prices generated by the matching engine, as these values can be adjusted based on corporate events and, depending on the applied factor, can have up to 8 decimal places. Therefore, *mdEntryPx* field from *ClosingPrice* message and *netChgPrevDay* field from *OpeningPrice* and *ExecutionStatistics* messages all use types that have exponential part = -8.

12.1 ExecutionSummary

The *ExecutionSummary* message is sent in four scenarios:

1. When an incoming (*aggressor*) order matches passive orders in the book.
2. When a cross order is successfully registered.
3. When a trade is generated when a retail order aggresses a hidden RLP order from its own brokerage firm (RLP trade).
4. Block trades for the following types: RFQ, Block Book and Midpoint.

The following conditions **do not** generate *ExecutionSummary* message:

1. Trades originated when auction ends (generated by matching orders that forms the theoretical price during the auction).
2. Trades published by manual insertion/change from market operation due to some operational reason.
3. Trades originated from forward agreement (*Termo*): trades on the *Termo* instrument and trades on the underlying instrument (cash instrument) in case of forward + cash ("*Termo Vista*").

It contains the instrument identifier, aggressor side and timestamp, worst price, total executed quantity, traded hidden quantity, and cancelled quantity (for instance, due to self-trade prevention). The individual trades and order deletions or modifications are sent after this message. That's why *matchEventIndicator* (tag 37035) field is not present in the message: this information is never the last one belonged to the current matching event.

This message has no corresponding equivalent in FIX/FAST diffusion.

Here are the main fields of the *ExecutionSummary* message:

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the *SecurityDefinition* message in Market Data feed.
2446	aggressorSide	R	AggressorSide Enum	Which side is aggressor of all fills.
31	lastPx	R	Price	Price of the last fill (i.e., worst price of this match).
1365	fillQty	R	Quantity	Quantity of all fills.
37779	tradedHiddenQty	O	QuantityOptional	Total quantity of matched passive orders that is not displayed to the market. That

Tag	Tag Name	Pres.	Data Type	Comments
				includes matched iceberg orders, RLP trades, block trades.
84	cxlQty	O	QuantityOptional	Total quantity canceled during matching process (e.g., due to self-trade).
2445	aggressorTime	R	UTCTimestampNanos	Timestamp of aggressive order resulting in match event. If the resulting matches are due to an aggressive order, the field reflects the instant this order enters the FIFO queue in the order entry flow.

Here are some scenarios to clarify how Execution Summary works:

12.1.1 Matching disclosed quantity orders (iceberg orders)

Disclosed Quantity allows participants to trade a large lot of a given security without exposing the whole lot in the market at once.

ExecutionSummary message provides a way to inform all hidden quantity traded for the whole event (transaction) independently if several different orders are involved, even if they are matched for each replenishment.

On the order entry side, *MaxFloor* (tag 111) field determines the largest amount which is shown in the order book at a time.

Suppose a sell order matches several passive orders in the buy side (all orders have limit price = 20.50), including an ordinary order with higher priority and an iceberg order described below:

Existing higher priority order in the book is filled first

Msg	Side	OrderID	Secondary OrderID	Qty	Max Floor	Last Qty	Leaves Qty	Ord Status	Exec Type
D	BUY	-	-	10000	500	-	-	-	-
Iceberg order is accepted									
8	BUY	ORD_1	SORD_1	10000	500		10000	New	New
D	SELL	-	-	1800	-	-	-	-	**-**
8	SELL	ORD_11	SORD_11	1800	-	-	1800	New	New
Disclosed quantity of 500 shares is totally filled by an aggressive order									
8	BUY	ORD_21	SORD_21	800	0	800	0	Filled	Trade
8	SELL	ORD_11	SORD_11	1800	-	800	1000	Partially Filled	Trade
8	BUY	ORD_1	SORD_1	10000	500	500	9500	Partially Filled	Trade
8	SELL	ORD_11	SORD_11	1800	-	500	500	Partially Filled	Trade
Order is replenished and a new Order ID (SORD_2) is sent									
8	BUY	ORD_1	SORD_2	10000	500		9500	Partially Filled	Restated
Disclosed quantity of 500 shares is totally filled by the same aggressive order									
8	BUY	ORD_1	SORD_2	10000	500	500	9000	Partially Filled	Trade
8	SELL	ORD_11	SORD_11	1800	-	500	0	Filled	Trade
Order is replenished and a new Order ID (SORD_3) is sent									
8	BUY	ORD_1	SORD_3	10000	500		9000	Partially Filled	Restated

On the market data side:

ExecutionSummary message is highlighted for better clarification:

MDEntryType	AggresorSide	LastPx	FillQty	TradedHiddenQty	CxlQty
s	SELL	20.50	1800	1000	0

Three trades were generated in this event:

MDEntryType	MDEntryPx	MDEntrySize	TradeID
2	20.50	800	10
2	20.50	500	20
2	20.50	500	30

Order book updates in this event:

Message	MDEntry Type	MDUpdate Action	MDEntryPx	MDEntry Size	MDEntryPositionNo	Secondary OrderID
DeleteOrder_MBO	1 (Offer)	2 (DELETE)	-	800	1	SORD_21
DeleteOrder_MBO	1 (Offer)	2 (DELETE)	-	500	1	SORD_1
Order_MBO	1 (Offer)	0 (NEW)	20.50	500	1	SORD_2
DeleteOrder_MBO	1 (Offer)	2 (DELETE)	-	500	1	SORD_2
Order_MBO	1 (Offer)	0 (NEW)	20.50	500	1	SORD_3

For better understanding of the dynamics of matching event, we omitted the publication of related statistics such as *HighPrice*, *LowPrice*, *ExecutionStatistics* messages.

NOTE

Each time the disclosed quantity of an iceberg order is replenished, it enters in the end of the order list with the same price-level with the lowest priority (like a new order), to preserve the hidden nature of iceberg orders.

12.1.2 Triggering self-trade prevention during a matching

Self-trading prevention at customer level is a functionality that aims to restrict matching between buying and selling orders from the same customer, regardless of firm.

For this purpose, the customer must be identified with a unique Investor ID, included within the order message.

Self-trading prevention gives the opportunity to choose which order should be canceled when identifying a potential match between an aggressor and a resting order. The options available are cancel aggressor order (default), cancel resting order and cancel both orders.

ExecutionSummary message provides a way to inform canceled quantity due to triggering self-trade prevention mechanism for the whole event (transaction) independently if several different orders are involved.

See the order entry side below for better clarification. We will use the “cancel resting order” option, for simplification:

Msg	Side	OrderID	Price / LastPx	OrderQty / Remaining Qty	Self-Trade Prevention Instruction	Investor ID	Exec Restatement Reason	Ord Status	Exec Type
D	BUY	-	21.00	100	-	-			
8	BUY	ORD_1	21.00	100 / 100	-	-	-	New	New
D	BUY	-	20.50	200	2 (Cancel Resting Order)	123			
8	BUY	ORD_2	20.50	200 / 200	2	123	-	New	New
Sell order is entered to match those 2 orders above									
D	SELL		20.50	300	2	123			
8	SELL	ORD_3	20.50	300 / 300	2	123	-	New	New
8	SELL	ORD_3	20.50 / 20.50	300 / 200	2	123	-	Partially Filled	Trade
8	BUY	ORD_1	21.00 / 20.50	100 / 0	-	-	-	Filled	Trade
8	BUY	ORD_2	20.50	200 / 0	2	123	107	Canceled	Canceled

On the market data side:

First of all, there are the insertion of two orders in the book before the aggression:

Message	MDEntry Type	MDUpdate Action	MDEntryPx	MDEntry Size	MDEntryPosition No	Secondary OrderID
Order_MBO	0 (Bid)	0 (NEW)	21.00	100	1	Not relevant
Order_MBO	0 (Bid)	0 (NEW)	20.50	200	2	Not relevant

Below are the messages published during the aggression:

ExecutionSummary message is highlighted for better clarification:

MDEntryType	AggresorSide	LastPx	FillQty	TradedHiddenQty	CxlQty
s	SELL	20.50	100	0	200

The cancelation of passive order due to triggering self-trade prevention mechanism is published before the *Trade* messages:

Message	MDEntry Type	MDUpdate Action	MDEntryPx	MDEntry Size	MDEntryPosition No	Secondary OrderID
DeleteOrder_MBO	0 (Bid)	2 (DELETE)	-	200	2	Not relevant

After that, trade was published related the only matching that occurred:

MDEntryType	MDEntryPx	MDEntrySize	TradeID
2	20.50	100	40

Order book updates related to the matching event:

Message	MDEntry Type	MDUpdate Action	MDEntryPx	MDEntry Size	MDEntryPosition No	Secondary OrderID
DeleteOrder_MBO	0 (Bid)	2 (DELETE)	-	100	1	Not relevant
Order_MBO	1 (Offer)	0 (NEW)	20.50	200	1	Not relevant

12.1.3 Triggering several stop orders/trades in a single matching event

This is a special case, where several stop orders are triggered by a previous match, and those orders, when added in the open book, match with orders in the other side of the book. In this scenario, all of trades originated from the stop orders added in the open book belonged to the same matching event that is composed by one *ExecutionSummary* message and several *Trade*, *DeleteOrder* and *Order* messages, and also related statistics messages with the last message in this event with *matchEventIndicator* flag set to true indicating the end of the current event.

On the market data side:

Here is the state of the open book before the aggression:

Bid				Ask		
Price	Secondary OrderID	Quantity	Position	Price	Secondary OrderID	Quantity
7.28	330	100	1	7.31	800	300
7.28	700	200	2	7.32	320	100
7.20	100	100	3	7.32	600	200
			4	7.50	200	100
			5	7.52	300	400

There are stop orders that haven’t triggered yet:

Bid		
Quantity	Price	Stop Price
100	7.31	7.31
200	7.32	7.31
200	7.32	7.31

Then, a bid order (price = 7.32, orderQty = 200) is added and instantly matches with the top of the offer book. A matching event that consists of *ExecutionSummary*, *Trade*, *DeleteOrder* messages and also statistics messages are published, and it is not represented here.

The resulting book before the stop orders entering in the open book is:

Bid				Ask		
Price	Secondary OrderID	Quantity	Position	Price	Secondary OrderID	Quantity
7.28	330	100	1	7.31	800	100
7.28	700	200	2	7.32	320	100
7.20	100	100	3	7.32	600	200
			4	7.50	200	100
			5	7.52	300	400

This action triggers the stop orders, and in this example, those 3 orders aggress several offers in the top of the book in one matching event as show below:

ExecutionSummary:

MDEntryType	AggresorSide	LastPx	FillQty	TradedHiddenQty	CxlQty
s	BUY	7.32	400	0	0

See that the traded quantity of those stop orders is **not** reflected in the *tradedHiddenQty* field.

Here are the other relevant messages that belongs to the event:

Trades:

MDEntryType	MDUpdateAction	TradeCondition	TradeID	MDEntryPx	MDEntrySize
2 (Trade)	NEW	Regular trade	1210	7.31	100
2 (Trade)	NEW	Regular trade	1220	7.32	100
2 (Trade)	NEW	Regular trade	1230	7.32	100
2 (Trade)	NEW	Regular trade	1240	7.32	100

After the matches, there is a remaining quantity from the last stop order that enters in the open book (bid side):

Message	MDEntry Type	MDUpdate Action	MDEntryPx	MDEntry Size	MDEntryPosition No	Secondary OrderID
Order_MBO	0 (Bid)	0 (NEW)	7.32	100	1	**720**

The final state of the book is:

Bid				Ask		
Price	Secondary OrderID	Quantity	Position	Price	Secondary OrderID	Quantity
7.32	**720**	**100**	**1**	7.50	200	100
7.28	330	100	**2**	7.52	300	400
7.28	700	200	**3**	7.52	320	100
7.20	100	100	**4**	7.55	360	300
			5	7.55	370	200

For better understanding of the dynamics of matching event, we omitted the publication of related statistics such as *HighPrice*, *LowPrice*, *ExecutionStatistics* messages.

12.2 ExecutionStatistics

The *ExecutionStatistics* message is sent as one of the statistics related to the trade execution. It conveys some useful statistics like *VWAP*, *TradeVolume* and number of trade events that are updated for each execution and are related to the whole trading session and will be reset after the end of the related trading session. Here are the main fields of the *ExecutionStatistics* message:

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
336	tradingSessionID	R	TradingSessionID Enum	Identifier for trading session.
1020	tradeVolume	R	QuantityVolume	Total traded volume for the session.
37778	vwapPx	O	PriceOptional	Volume-weighted average price.
451	netChgPrevDay	O	PriceOffset8Optional	Net change from previous trading day's closing price vs. last traded price.
37073	numberOfTrades	R	NumberOfTrades	Number of trades executed in the session.
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which a set of market data applies.

There is a new field "numberOfTrades" that just counts the number of trades executed in the session for the instrument. It is a counterpart of "tradeVolume", that is the total traded volume for the session. It has no equivalent in the FIX/FAST diffusion.

12.3 Trade

The *Trade* message is sent when a trade occurs to provide volume and trade statistics.

Here are the main fields of the *Trade* message:

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 4: Trade resulted from an implied generated order. Bit 7: Last message for the event.
336	tradingSessionID	R	TradingSessionID Enum	Identifier for trading session.
277	tradeCondition	R	TradeCondition Set	Set of conditions describing a trade.
270	mDEntryPx	R	Price	Price of the Market Data Entry.
271	mDEntrySize	R	Quantity	Quantity or volume represented by the Market Data Entry.
1003	tradeID	R	TradeID	Contains the unique identifier for this trade per instrument + trading date, as assigned by the exchange.
288	mDEntryBuyer	O	FirmOptional	For reporting trades (buying party).
289	mDEntrySeller	O	FirmOptional	For reporting trades (selling party).
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which a set of market data applies.
829	trdSubType	O	TrdSubType Enum	Sub type of trade assigned to a trade.
37033	mDEntryTimestamp	R	UTCTimestampNanos	Timestamp when the trade event occurred.

NOTE

Unless the bit-3 (*OutOfSequence*) of *TradeCondition* is set, the field *MDEntryPx* value is the last traded price.

12.3.1 TradeCondition

Each trade has a set of properties ("conditions") that are listed below:

Condition Name	FIX Equivalent	Bit Number	Condition
OpeningPrice	"R"	0	This is one of the trades that forms the opening trade event that indicates when an instrument is traded for the first time in the trading session in progress.
Crossed	"X"	1	Trade resulted from a cross order.
LastTradeAtTheSamePrice	"L"	2	Trade's price is the same as the last trade price.
OutOfSequence	"S"	3	This trade is out of sequence and its price must not be used for determining the last trade price.
TradeOnBehalf	"2"	6	Marketplace entered trade.
RegularTrade		13	1=Regular Trade, 0=Special trade type (see *TrdSubType* enum).
BlockTrade		14	1=Block Trade (see *TrdSubType* enum for details), 0=Not.

12.3.2 TrdSubType

If a trade is not a regular one, i.e., *RegularTrade* condition (13-bit) is not set in the *TradeCondition* set, then *TrdSubType* enum (tag 829) informs one of the following sub type of trade:

Enum Type	Value	Is Block Trade?	Description
MULTI_ASSET_TRADE	101		Multi Asset Trade. Trade in the cash instrument originated from a forward + cash agreement (Termo Vista).
LEG_TRADE	102		Leg Trade. Indicates trade of leg originated from a trade of related UDS or EDS.
MIDPOINT_TRADE	103	✓	Midpoint Trade.
BLOCK_BOOK_TRADE	104	✓	Block Book Trade.
RFQ_TRADE	105	✓	Request for Quote (RFQ) Trade.
RLP_TRADE	106		Retail Liquidity Provider (RLP) Trade.
TAC_TRADE	107		Trade at Close Trade.
TAA_TRADE	108		Trade at Average Trade.
SWEEP_TRADE	109		Sweep Trade. Can be combined with *TradeCondition*: Crossed ("X").

12.3.3 Relationship between TradeCondition and TrdSubType

What is a Regular Trade?

- Regular Trade is everything except: RFQ trade, Block Book trade, Midpoint trade, Leg trade, Multi Asset trade and RLP (Retail Liquidity Provider) trade.
- So, crossed, sweep trade and trade on behalf are both considered a regular trade.

- Sweep Trade indicates that the trade is part of a Sweep and Cross operation, when a trader sends and special cross order that tries to sweep the other side of the book, and after that, if there is any remaining quantity, tries to cross with the other side account within the same firm.

Please check the following table, summing up possible values for each trade type that can have others trade condition and trade sub type:

Regular Trade (bit 13=1)		Block Trade (bit 14=1)		No Block / No Regular (bit 13=0 and bit 14=0)	
Trade Condition	**Trade SubType**	**Trade Condition**	**Trade SubType**	**Trade Condition**	**Trade SubType**
Opening Price		Last Trade At The Same Price	RFQ trade	Opening Price	Leg trade
Crossed		Trade On Behalf	Block Book trade	Crossed	Multi Asset trade
Last Trade at The Same Price			Midpoint trade	Last Trade at The Same Price	RLP
Trade on Behalf				Trade on Behalf	
	Sweep Trade				
Crossed	Sweep Trade				

So, in the other hand when you receive a trade condition Regular Trade equals to zero (bit 13=0) it means that exists a *TrdSubType* value associated with, and it is a non-regular trade as well.

You can also check it on the table in section 12.3.2.

12.4 Trade Bust

The *TradeBust* message is sent when a trade is cancelled (busted) by Market Supervision. The fields are almost equal to the *Trade* message.

Tag	Tag Name	Presence	Comments
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 4: Bust of a trade resulted from an implied generated order. Bit 7: Last message for the event.
336	tradingSessionID	R	Identifier for trading session.
270	mDEntryPx	R	Price of the Market Data Entry.
271	mDEntrySize	R	Quantity or volume represented by the Market Data Entry.

Tag	Tag Name	Presence	Comments
1003	tradeID	R	Contains the unique identifier for this trade per instrument + trading date, as assigned by the exchange.
75	tradeDate	R	Used to specify the trading date for which a set of market data applies.
37033	mDEntryTimestamp	R	Timestamp when the trade bust event occurred.

12.5 Forward Trade

Trades for Forward ("Termo") instruments use the message *ForwardTrade*. There are two additional fields:

Tag	Name	Pres.	Comments
37014	mDEntryInterestRate	O	Interest Rate of the Termo Trade. Expressed in decimal form. For example, 1% points is expressed and sent as 0.01. One basis point is represented as 0.0001.
287	sellerDays	O	Specifies the number of days that may elapse before delivery of the security.

12.6 Last Trade Price

The *LastTradePrice* message is sent mainly in snapshot stream with the last trade price and other statistics. In incremental stream, it is sent very rarely, to adjust the last trade price after a catastrophic failure in the publishing service.

In the scenario where this message is been published in the incremental stream, *tradeCondition* field does not reflect the previous conditions of the related trade event. So, all the bits of this field are 0 (zero).

Here are some of the FIX tags sent for a *LastTradePrice* message:

Tag	Name	Pres.	Comments
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 4: Trade resulted from an implied generated order. Bit 7: Last message for the event.
270	mDEntryPx	O	Last Trade Price. In snapshot stream, copied from lastPx field from *ExecutionSummary* message.
37033	sellerDays	O	Specifies the number of days that may elapse before delivery of the security. Only used for trades in forward market.
37014	mDEntryInterestRate	O	In snapshot stream, copied from this field from *ForwardTermo* message; only sent for forward ("termo") instruments.

12.7 Trading Session High/Low Price

These messages are sent in incremental and snapshot streams.

Tag	Name	Values	Comments
269	mDEntryType	C	“7” (High) – for HighPrice message. “8” (Low) – for LowPrice message.
48	securityID	R	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
270	mDEntryPx	R	High price, low price.
75	tradeDate	R	Used to specify the trading date for which a set of market data applies.

12.8 Calculation of Trading Session VWAP Price

VWAP Price comes in *ExecutionStatistics* messages as the *VwapPx* field (tag 37778).

VWAP is calculated as below:

The Volume-Weighted Average Price is the ratio of the value traded to the total volume traded over the trading session. It is calculated by the formula:

$$P_{VWAP} = \frac{\sum_j P_j Q_j}{\sum_j Q_j}$$

where:

P_{VWAP} is the Volume Weighted Average Price

P_j is the price of trade j

Q_j is the quantity of trade j

j is each individual trade that takes place over the defined period (including cross trades)

12.9 Opening Price / Theoretical Opening Price / Closing Price

The message *OpeningPrice* carries the summary information about opening trading session events per market data stream.

The message *TheoreticalOpeningPrice* carries the theoretical opening price, that is calculated and updated based on the orders presented in the book during every auction, including the pre-opening and pre-closing auction.

The message *ClosingPrice* carries the summary information about closing trading sessions per market data stream, including the previous day’s adjusted closing price.

The messages are summarized below (R= required, O=optional, C = constant):

12.9.1 OpeningPrice message

Here are the main fields of the *OpeningPrice* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.

Tag	Tag Name	Pres.	Data Type	Comments
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
286	openCloseSettlFlag	R	OpenCloseSettlFlag Enum	Identifies if the opening price represents or not a daily opening price.
270	mDEntryPx	R	Price	Value of the statistics.
451	netChgPrevDay	O	PriceOffset8Optional	Net change from previous trading day’s closing price vs. last traded price.
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which a set of market data applies.

12.9.2 TheoreticalOpeningPrice message

Here are the main fields of the *TheoreticalOpeningPrice* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
286	openCloseSettlFlag	C	OpenCloseSettlFlag Enum	Indicates this is a theoretical opening price. Constant: 5 (Theoretical Price)
270	mDEntryPx	O	PriceOptional	Theoretical Opening Price.
271	mDEntrySize	O	QuantityOptional	Theoretical Opening Quantity.
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which a set of market data applies.

12.9.3 ClosingPrice message

Here are the main fields of the *ClosingPrice* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
286	openCloseSettlFlag	R	OpenCloseSettlFlag Enum	Identifies if the closing price represents a daily or entry from previous business day.
270	mDEntryPx	R	Price8	Closing price.

75	tradeDate	R	LocalMktDate	Used to specify the trading date for which a set of market data applies.
9325	lastTradeDate	O	LocalMktDateOptional	Date the instrument last traded.

12.10 Auction Imbalance

The message *AuctionImbalance* relays auction imbalance information, indicating the remaining quantity and to which side (buyer or seller) the auction is pending towards.

12.10.1 *AuctionImbalance* message

Here are the main fields of the *AuctionImbalance* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
277	imbalanceCondition	R	ImbalanceCondition Set	IMBALANCE_MORE_BUYERS, IMBALANCE_MORE_SELLERS, All bits off => BALANCED.
271	mDEntrySize	O	QuantityOptional	Remaining auction quantity.

NOTE

Client applications are responsible for deleting the existing theoretical opening price and imbalance information from memory after trading phase changes from Pre-Open (tag 625-TradingSessionSubID = 21) for each instrument in the group whose trading status is different from Pre-Open status (tag 326-SecurityTradingStatus = 21).

12.11 Price and Quantity Bands Information

The message *PriceBand* relays most of the information regarding price tunnels and bands; the message *QuantityBand* relays most of the information regarding quantity tunnels and bands. They are relayed on the Market Data channel for each specific instrument.

The following types of bands and limits are supported:

PriceBand (PriceBandType)	QuantityBand
Hard Limits (HARD_LIMIT)	
Rejection Band (REJECTION_BAND)	
Auction Band (AUCTION_LIMITS)	
Static Limits (STATIC_LIMITS)	
Reference Price (PriceBandType is omitted)	Quantity Limit (Equities only)

12.11.1 PriceBand message

Here are the main fields of the *PriceBand* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
6939	priceBandType	O	PriceBandType Enum	Indicates the type of price banding (tunnel).
1306	priceLimitType	O	PriceLimitType Enum	Describes how the price limits are expressed.
37008	priceBandMidpointPriceType	O	PriceBandMidpoint-PriceType Enum	Band Midpoint Type, used with Auction Price Banding. Only sent for Rejection and Auction Bands when *PriceLimitType* (1306) equals to 2 (Percentage).
1148	lowLimitPrice	O	PriceOptional	Allowable low limit price for the trading day. A key parameter in validating order price. Used as the lower band for validating order prices. Orders submitted with prices below the lower limit will be rejected.
1149	highLimitPrice	O	PriceOptional	Allowable high limit price for the trading day. A key parameter in validating order price. Used as the upper band for validating order prices. Orders submitted with prices above the upper limit will be rejected.
qsnip p1150	tradingReferencePrice	O	PriceOptional	Reference price for the current trading price range. The value may be the reference price, settlement price or closing price of the prior trading day. Sent only for Economic Indicators.

If a client system tracks bands for a purpose, it is important to know that the *PriceBandType* attribute is a qualifier of the price band information: Hard Limits, Reject Band, Auction Band, and Static Limits can coexist simultaneously. An update must be applied specifically to the informed type, keeping the previously defined band attributes for the other types.

12.11.2 QuantityBand message

Here are the main fields of the *QuantityBand* message.

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
37003	avgDailyTradedQty	O	QuantityVolumeOptional	Daily average shares traded within 30 days – equity market only. Always 0 for Derivatives.
1140	maxTradeVol	O	QuantityVolumeOptional	The maximum order quantity that can be submitted for a security. The value is the minimum between % of shares issued and % of average traded quantity within 30 days.

The tunnels don´t change intraday. It is also known as “oscillation tunnel” establishing the price limits (lower and higher) of an instrument. Any order submitted with a price below the low limit or above the high limit will be rejected.

12.11.3 SettlementPrice message

Here are the main fields of the *SettlementPrice* message:

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which settlement price applies.
270	mDEntryPx	R	Price	Settlement Price.
286	openCloseSettlFlag	R	OpenCloseSettlFlag	Identifies if the settlement price represents a daily, preliminary or an entry from previous business day.
423	priceType	R	PriceType	Code to represent the price type: PERCENTAGE, PU or FIXED AMOUNT.
731	settlPriceType	R	SettlPriceType	Type of settlement price: FINAL, THEORETICAL or UPDATED.

It’s advised that the client application is able to process any combination of the *openCloseSettlFlag* (tag 286) and *settlPriceType* (tag 731) fields.

The following table illustrates the regular schedule:

Event Time	OpenCloseSettlFlag	SettlPriceType	Description
Before trading session	ENTRY_FROM_PREVIOUS_BUSINESS_DAY	FINAL	Previous day final settlement.
During trading session	ENTRY_FROM_PREVIOUS_BUSINESS_DAY	UPDATED	Previous day updated settlement.
During trading session	DAILY	UPDATED	Current day preview settlement.
After trading session	DAILY	FINAL	Current day final settlement.

12.11.4 OpenInterest message

Here are the main fields of the *OpenInterest* message:

Tag	Tag Name	Pres.	Data Type	Comments
48	securityID	R	SecurityID	Security ID as defined by B3. For the SecurityID list, see the Security Definition message in Market Data feed.
37035	matchEventIndicator	R	MatchEventIndicator Set	Set of indicators that identify some market data events. Here are the possible bits applied to this message when it is set: Bit 7: Last message for the event.
75	tradeDate	R	LocalMktDate	Used to specify the trading date for which open interest applies.
271	mDEntrySize	R	Quantity	Indicates volume of contracts currently open.

12.11.5 Specific usage for each type of Band and Tunnel

Tag	Name	Reference Price	Hard Limits	Rejection Band	Auction Band	Static Limits	Quantity Limits
48	SecurityID	X	X	X	X	X	X
207	SecurityExchange	X	X	X	X	X	X
37033	MDEntryTimestamp	X	X	X	X	X	X
6939	PriceBandType	-	1 (HARD_LIMIT)	3 (REJECTION_BAND)	2 (AUCTION_LIMITS)	4 (STATIC_LIMITS)	-
1306	PriceLimitType	-	0 (PRICE_UNIT)	2 (PERCENTAGE 0 (PRICE_UNIT))	2 (PERCENTAGE 0 (PRICE_UNIT))	0 (PRICE_UNIT)	-
1148	LowLimitPrice	-	X	X	X	X	-
1149	HighLimitPrice	-	X	X	X	X	-
1150	TradingReferencePrice	C	-	-	-	-	-
37008	PriceBandMidpointPriceType	-	-	LAST_TRADED_PRICE, COMPLEMENTARY_LAST_PRICE, THEORETICAL_PRICE (When 1306= PERCENTAGE)	LAST_TRADED_PRICE, COMPLEMENTARY_LAST_PRICE, THEORETICAL_PRICE (When 1306= PERCENTAGE)	-	-
37003	AvgDailyTradedQty	-	-	-	-	-	X
1140	MaxTradeVol	-	-	-	-	-	X

*Tags marked with an “X” are required, those marked with “-“ are not sent, otherwise they have the specified values.

13. Group Phase/Instrument State Information

B3 will relay the state of an individual instrument or a group of instruments using the following messages:

- **SecurityStatus** – in incremental and snapshot stream. Used to relay instrument state changes intraday.

- **SecurityGroupPhase** – in incremental and snapshot stream. Used to relay security group phase changes intraday.

When the client system starts up, it should consider that all snapshots contain the current phase of the security group and current state of the individual instrument if it is detached from the security group it belongs, informed by tag 1174-*SecurityTradingEvent* = 101-"Security Status maintained separately from Group Status". Intraday updates may be done on the instrument level or group level.

NOTE

Group codes may repeat amongst different exchanges, hence it is advisable that client systems use the key group code (tag 1151 – *SecurityGroup*) + exchange (tag 207 – *SecurityExchange*).

When processing the *SecurityGroupPhase* message in the incremental stream, client systems must first look for tag *1151-SecurityGroup*. This tag contains the group identification of a set of instruments. That being the case, all individual instruments of that set will have their status changed to the value of tag 625–*TradingSessionSubID*. The following message example illustrates the change of trading phase of the group "XX" to "PAUSE":

SecurityGroupPhase message		
Tag	**Tag Name**	**Value**
1151	securityGroup	XX
207	securityExchange	BVMF
625	tradingSessionSubID	2 (PAUSE)

A *SecurityStatus* message refers to an instrument, referred to tag *48-SecurityID*. Suppose that the group phase is 17 (**OPEN**) but the instrument state changes to 2 (**PAUSE**), separating from the group phase. In this case, the tag *SecurityTradingEvent* will be 101 (**SECURITY_STATUS_CHANGE**):

SecurityStatus message		
Tag	**Tag Name**	**Value**
48	securityID	99999999
207	securityExchange	BVMF

SecurityStatus message		
326	securityTradingStatus	2 (PAUSE)
1174	securityTradingEvent	101 (SECURITY_STATUS_CHANGE)

Now if the instrument state return to 17 (**OPEN**), following the group phase, the *SecurityTradingEvent* will be 102 (**SECURITY_REJOINS_SECURITY_GROUP_STATUS**):

SecurityStatus message		
Tag	**Tag Name**	**Value**
48	securityID	99999999
207	securityExchange	BVMF
326	securityTradingStatus	17 (OPEN)
1174	securityTradingEvent	102 (SECURITY_REJOINS_SECURITY_GROUP_STATUS)

An important note is that in snapshots, if the value of *securityTradingStatus* field is null, it means the instrument status follows the status of the security group. For example, if the value of *tradingSessionSubID* field is OPEN for the group of the related instrument, it means the security status of that instrument is also OPEN even the value of the *securityTradingStatus* field is null.

Please see the complete *SecurityStatus* and *SecurityGroupPhase* message format at UMDF SBE Message Reference document.

NOTE

Whenever an instrument state rejoins the group phase (1174=102), it is safe to infer the group phase (tag 625) from the current instrument state (tag 326).

13.1 Instrument States

The list of the different instrument states available on the tag ***326****-SecurityTradingStatus* is indicated in the following table:

Name	Tag Value	Description
Trading halt **(*PAUSE*)**	2	This instrument state is used by surveillance to prevent order entry and matching by market operations or schedule. Orders in the book are not eliminated when instrument entering this state.
No-Open **(*CLOSE*)**	4	This instrument state is used by market surveillance to perform a limited number of functions, including in particular, consultations. Users have no order entry, modification or cancel capability during this phase.
Ready to trade **(*OPEN*)**	17	This instrument state is used by subscribers and surveillance to enter, modify, and cancel orders, subject to cancellation and modification rules. The orders entered during this period result in immediate trading if counterparty is matched and the specific instrument status also equals to "Open". Otherwise, the more restrictive status rules.
Not available for trading **(*FORBIDDEN*)**	18	This instrument state is used by surveillance to prevent order entry and matching by market operations command or schedule. Users have no order entry, modification or cancel capability during this phase.
Pre-Open **(*RESERVED*)**	21	Reserved state is the auction functionality that can be triggered (auction band and self-trading for illiquid instrument, for example) or started by Market Operations by command. Time of state can be pre-defined. The reserve state can have a defined time with the fixed closed (opening) time or random closed (opening) time. This state is used by subscribers and surveillance to enter, modify, and cancel (subject to cancellation and modification rules) orders. The orders entered during this period do not result in immediate trading but are used to determine a Theoretical Opening Price. State ended when trades are created based on auction algorithm prior to transition to another state.
FINAL_CLOSING_CALL	101	This state indicates when the instrument is on the final closing call for the trading day (Equities only). It behaves similarly to Reserved state.

13.2 Trading Phases

A trading phase identifies the “state” of a whole group of instruments in terms of trading session. By default, all instruments follow the trading phase of the group they belong to.

For example, group “XX” may be in trading phase “*Open”*, but instrument ABCD that belongs to group “XX” is in the “*Pause”* status – due to market surveillance command. This information is especially useful when client systems want to determine the state of the group altogether and outlining the individual state of the instrument.

Trading phase information is relayed to client systems using tag 625–*TradingSessionSubID*.

The following table presents the domain of possible trading phases:

Name	Tag Value	Description
PAUSE	2	This trading phase is used by surveillance to prevent order entry and matching by market operations or schedule.
CLOSE	4	This trading phase is used by market surveillance to perform a limited number of functions, including in particular, consultations. As a rule, during this phase, surveillance checks the consistency of data and post-market state process results before the start of the trading day. Users have no order entry, modification or cancel capability during this phase.
OPEN	17	This trading phase is used by subscribers and surveillance to enter, modify, and cancel orders, subject to cancellation and modification rules. The orders entered during this period result in immediate trading if counterparty is matched and the specific instrument status also equals to “Open”. Otherwise, the more restrictive status rules.
FORBIDDEN (Pre-Close)	18	This trading phase is used to indicate an intervention by surveillance. If the specific instrument is in “Reserved” state, the auction continues, otherwise users have no order entry, modification or cancel capability during this phase.
RESERVED (Pre-Open)	21	This trading phase is used to indicate that all instruments belong to the group is in “Reserved” state except the status of the instrument indicates “Forbidden” state.
FINAL_CLOSING_CALL	101	This phase indicates when the instruments on this group are on the final closing call for the trading day (Equities only). It behaves similarly to Pre-Close phase, however when entering this phase, MOC orders are triggered.

13.3 Trading Statistics Reset

If customer systems receive a *SecurityStatus* message with the tag 1174-*SecurityTradingEvent* with value 4 (**TRADING_SESSION_CHANGE**), it is an event of "end of day trading statistics reset".

On receipt of this message and flag, client systems are expected to reset the information that is received in the following messages:

MDEntryType	Description
LastTradePrice	Last trade
ExecutionStatistics	Execution statistics information (VWAP, TradeVolume, NumberOfTrades)
OpeningPrice	Opening price
TheoreticalOpeningPrice	Theoretical opening price
HighPrice	Trading session high price
LowPrice	Trading session low price

The statistics above will also be reset in the snapshot stream.

13.4 Group Phase and Instrument State in the Snapshot Stream

The snapshot stream publishes, for each security group, the current security group phase as a *SecurityGroupPhase* message, and for each instrument, the current state of an instrument as a *SecurityStatus* message.

The "snapshot" reflects the last state of the instrument and the last correlated phase that affected the group to which the instrument belongs to.

14. Derivatives/FX Specific Market Data Functionality

This section refers to the Derivatives/FX segment functionality only, and important technical information for B3 customers on how to process this data.

14.1 Option Strike Price

Some instruments disseminated on the options channels are not options per se, but spreads on options, this happens with Rollovers and Strategies. The decision was made to keep them on the same channel as their underlying options, even though they are not proper options.

Hence, as such instruments are not options, **the strike price field (tag 202-*StrikePrice*) will not be sent or sent as zero**.

The proper way to identify these instruments is checking their security subtype (tag 762-*SecuritySubType*). The following subtypes are used to identify them:

Product Description	Tag 762 (SecuritySubType) value
Strategies	90 (STRATEGY)
Financial Rollover	140 (FINANCIAL_ROLLOVER)
Agricultural Rollover	141 (AGRICULTURAL_ROLLOVER)

15. Trade Volume, VWAP and Number of Trades

On Binary UMDF, the total traded volume on the electronic platform, as well as the VWAP price and the number of trades in the current session, are reported on the *ExecutionStatistics* message.

16. Implieds

The Implied functionality consists of deriving tradable orders (a synthetic order) generated from orders of two or more legs of an instrument (spreads or strategies), i.e., it allows orders to be created in a book from two other orders from different books. Messages that directly handle that synthetic order (*Order_MBO* and *DeleteOrderMBO*) can be identified with bit-4 (*Implied*) of the *matchEventIndicator* field. Messages that indirectly handle synthetic orders (*MassDeleteOrders_MBO*) do not have bit-4 (*Implied*) set to true because there could be regular orders also involved in the operation.

An implied order can only be created from orders from outright book in a spread/strategy book (Implied In) or from orders from an outright book and orders from a spread/strategy book in an outright book (Implied Out).

If one of the orders of the outright instrument that generated the implied order is fully or partially executed, cancelled or modified, the implied order is automatically cancelled. If the new calculated quantity of the implied order still exists after that, a new synthetic order is restated in same event in case of modification/cancellation or another event in case of any full/partial execution.

Instruments that are eligible to be used in the Implied mechanism have the *impliedMarketIndicator* field (tag 1144) enabled in the *SecurityDefinition* message.

NOTE

An implied order has always the least priority at the same price-level in the book. So, when a new normal order enters in the book and has the same price as the existent implied order, it has the precedence over (position number is less than existing implied order) even though it enters after that.

16.1 Implied In orders

These orders are derived from existing outright orders in individual contracts (legs). This means that an outright order in a spread can be matched with other outright orders in the spread/strategy OR with a combination of orders in the legs of the spread/strategy based on price and the rules corresponding to the method of allocation being used.

Example:

Moment #1: Client X inserts a buy order with 10 @ 6.99 in DI1F24:

DI FUT – DI1F24 (maturity: Jan/24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
10	6.99 (A)		

Moment #2: Client Y inserts a sell order with 10 @ 7.34 in DI1N24:

DI FUT – DI1N24 (maturity: Jul/24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
		7.34 (B)	10

Moment #3: the combination of 2 orders above generates an implied (IN) order in offer book with 10 @ 0.35 in DIIF24N24 (spread of maturities of F24 and N24 with ratio = 1):

Spread of Sell DI1F24 and Buy DI1N24 (DIIF24N24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
		0.35 (B - A)	10

Order book updates for the Implied In scenario:

Message	Instrument	MDEntry Type	MDUpdate Action	MDEntry Px	MDEntry Size	MDEntryPosition No	MatchEvent Indicator
Order_MBO	DI1F24	0 (Bid)	0 (NEW)	6.99	10	1	0
Order_MBO	DI1N24	1 (Offer)	0 (NEW)	7.34	10	1	0
Order_MBO	DIIF24N24	1 (Offer)	0 (NEW)	0.35	10	1	16 (4-bit)

To illustrate the least priority nature of implied order, imagine that a trader enters now enters a new order with the same price as the existing implied order at the same side of the book:

Moment #4: Client Z inserts a sell order with 15 @ 0.35 in DIIF24N24:

Message	Instrument	MDEntry Type	MDUpdate Action	MDEntry Px	MDEntry Size	MDEntryPosition No	MatchEvent Indicator
Order_MBO	DIIF24N24	1 (Offer)	0 (NEW)	0.35	15	1	0

The resulted book is:

Spread of Sell DI1F24 and Buy DI1N24 (DIIF24N24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
		0.35 (Normal)	15
		0.35 (Implied)	10

16.2 Implied Out orders

These orders are derived from the combination of an existing outright order in a spread/strategy and an existing outright order in one of the individual underlying legs. These two outright orders are utilized to create a contingent outright order on the other underlying leg of the spread. This means that an outright order in a leg can be matched with other outright orders for this specific leg OR with a combination of orders from **any** spread/strategy composed of this leg and orders of the other corresponding leg of the spread. **This feature is not yet available, and the availability will be announced through appropriate channels in time.**

16.3 Match that involves an implied order

A match event that involves an implied order generates trades and book order updates in the market data. The trade message that has resulted from a match that involves any implied order has the implied flag set to true in *matchEventIndicator* field (bit-4). Just like a cancellation of a trade (trade bust) that involved an implied order also has the implied flag set to true in *matchEventIndicator* field (bit-4).

When the partial match occurs that involves the synthetic order, the remaining quantity of the synthetic order is cancelled in the same event to avoid any inconsistencies because the availability of any quantity of the synthetic order depends on some verifying the common factors of the *outrights*, and it is expensive enough for latency perspective to compute in the same event.

After that, **in another event**, the synthetic order may be restated in the book of the spread/strategy with another *secondaryOrderID* value.

In continuation of the first example (Implied In):

Moment #5: client Z does an aggression of the normal and implied order inserting a bid order with 20 @ 0.35 in DIIF24N24:

Spread of Sell DI1F24 and Buy DI1N24 (DIIF24N24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
20	**0.35**	0.35 (Normal)	15
		0.35 (Implied)	10

Trades generated in this event:

Instrument	MDEntryType	MDEntryPx	MDEntrySize	MatchEventIndicator
DIIF24N24	2	0.35	15	0
DIIF24N24	2	0.35	5	16 (bit-4)
DI1F24	2	6.99	5	16 (bit-4)
DI1N24	2	7.34	5	16 (bit-4)

Resulted book:

Spread of Sell DI1F24 and Buy DI1N24 (DIIF24N24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
		0.35	**5**

DI FUT – DI1F24 (maturity: Jan/24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
5	6.99		

DI FUT – DI1N24 (maturity: Jul/24)			
Bid Qtd	Bid Price	Offer Price	Offer Qtd
		7.34	5

Order book updates (**same event**):

Message	Instrument	MDUpdate Action	MDEntry Type	MDEntryPx	MDEntry Size	MDEntryPosition No	MatchEvent Indicator
MassDeleteOrders _MBO	DIIF24N24	4 (Delete From)	1 (Offer)	-	25	2	**0**
Order_MBO	DI1F24	1 (Change)	1 (Bid)	6.99	5	1	0
Order_MBO	DI1N24	1 (Change)	0 (Offer)	7.34	5	1	0

Restate of the synthetic order in the spread (**another event**):

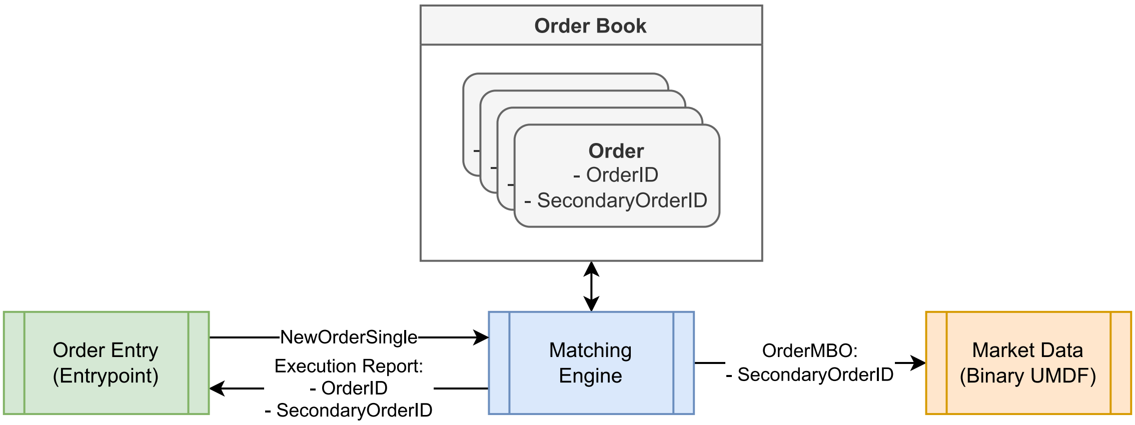
Message	Instrument	MDUpdate Action	MDEntry Type	MDEntryPx	MDEntry Size	MDEntryPosition No	MatchEvent Indicator
Order_MBO	DIIF24N24	0 (New)	1 (Offer)	0.35	5	1	16 (4-bit)

17. Miscellaneous Remarks

- Trades marked with 277-*TradeCondition* with bit 3 set (**OutOfSequence**) represent trades that are reported out of sequence (like market operation entered trades); they must not be considered if your objective is to get the “last trade price”.

- When processing leg trades (277-*TradeCondition* has bit 5 set – **LegTrade**), the trade price should not be used to infer the Last Trade Price.

- In FIX/FAST market data diffusion, B3 maps the *secondaryOrderID* field (198) from Order Entry to the *OrderID* field (37) in Market Data. In Binary UMDF, the *secondaryOrderID* tag (198) from Order Entry and the *secondaryOrderID* field (198) in Market Data have the same content. The behavior continues the same: a Market Data client cannot see the real *OrderID* from Order Entry. Whenever an order loses priority in the book (for instance, because it was modified and the quantity was increased or the price was changed), it appears that the order gets deleted and a new order is added with the new parameters, and the value of tag 198 – *secondaryOrderID* changes. For Iceberg orders (display quantity is different from real quantity), when the order is refilled, the value of tag 198 – *secondaryOrderID* also changes, in order to make more difficult to distinguish between regular and iceberg orders).



The diagram above what happens with a single order that is entered: the value of the field *secondaryOrderID* in *ExecutionReport* message is broadcast as the *secondaryOrderID* field in *OrderMBO* message.

- When a trade deletion (“trade bust”) is sent, i.e., a *TradeBust* message, no prices are resent. The other prices are only changed automatically if all trades are cancelled. In this case some of the prices can be deleted as well.