# BSE Exchange's New Trading Architecture

# BSE Market Data Interfaces

**Manual**

| | |
|---|---|
| Version | V1.3.3 |
| Date | 12 July 2013 |

# Contents

# Part I

# General Overview

## 1 List of abbreviations

The table below shows all the abbreviations and definitions used in this document.

| | |
|---|---|
| **BSE EMDI** | BSE Enhanced Market Data Interface |
| **BSE MDI** | BSE Market Data Interface **BSE** |
| **BSE ETI** | BSE Enhanced Transaction Interface |
| **FAST** | FIX Adapted for Streaming (FAST Protocol) (FAST Protocol $^{SM}$). FIX Adapted for streaming is a standard which has been developed by the Data Representation and Transport Subgroup of FPLs Market Data Optimization Working Group. FAST uses proven data redundancy reductions that leverage knowledge about data content and data formats. |
| **FIX** | Financial Information eXchange. The Financial Information eXchange ("FIX") Protocol is a series of messaging specifications for the electronic communication of trade-related messages. |
| **In-band** | Incremental and snapshots are delivered in the same channel. |
| **Match event** | Part of the matching event having a unique match price. |
| **Out-of-band** | Incrementals and snapshots are delivered on different channels. |
| **Simple instruments** | Single leg outright contracts |
| **Complex instruments** | Any combination of single leg outright contracts, e.g. Future Time Spreads. |
| **Live-Live concept** | The concept whereby data is disseminated simultaneously via two separate channels called "Service A" and "Service B". |
| **PMAP** | Presence Map |
| **ToB** | Top of Book |

# 2 Introduction

BSE offers public market data via two interfaces as part of the BSE Exchange's new trading architecture. All two interfaces distribute information via UDP multicast; following FIX 5.0 SP2 semantics and are FAST 1.2[1] encoded. If any messages are lost, complete recovery is possible because every message is published on two identical services (A and B) with different multicast addresses (live-live concept). In the unlikely case that a message is lost on both services, participants can take advantage of the respective snapshot message and rebuild the order book.

There are two types of **Market Data Interface**:

- The BSE **Enhanced Market Data Interface (BSE EMDI):** This interface provides un-netted market data. The updates of the order book are delivered for all order book changes up to a given level;
- The **BSE Market Data Interface (BSE MDI):** This interface provides netted market data. The updates of the order book are sent at regular intervals; they are not provided for every order book change and are sent significantly less frequently than the BSE EMDI

The BSE EMDI and BSE MDI provide the following information to the participant:

- Price level aggregated order book depth and.[2]
- Product and instrument states.

## 2.1 Purpose of this document

The purpose of this document is to provide guidance for programmers during development of applications that read the BSE Market Data Interfaces.

It covers a complete reference for the two multicast based public interfaces, describes the general business behaviour and provides concepts for the implementation.

The most recent version is available at:

www.BSEchange.com > Technology > System Documentation > New Trading Architecture > Release 1.0 > Market Data Interfaces.

[1] FAST 1.1 templates are provided as well.

[2]

[3] With an update interval of 5 minutes.

## 2.2 Main audience

The target audience of this interface specification is experienced software developers support staff that may be involved in development/support activities for the BSE Market Data Interfaces.

Prior knowledge of developing for a derivatives market is beneficial but not a prerequisite. Knowledge in a programming language is expected. Programmers who have no experience in a market data interface environment can gain a basic understanding of the feed behaviour by reading Part II (How to guide). This manual does not attempt to cover basic knowledge of programming techniques and software development.

## 2.3 Data feeds

All interfaces deliver public market data in the form of snapshots and incrementals as can be seen in Figure 1. The two public market data interfaces, the **BSE EMDI** for a high bandwidth network and the **BSE MDI** for a low bandwidth network, disseminate information across the BSE network to the receiving application.

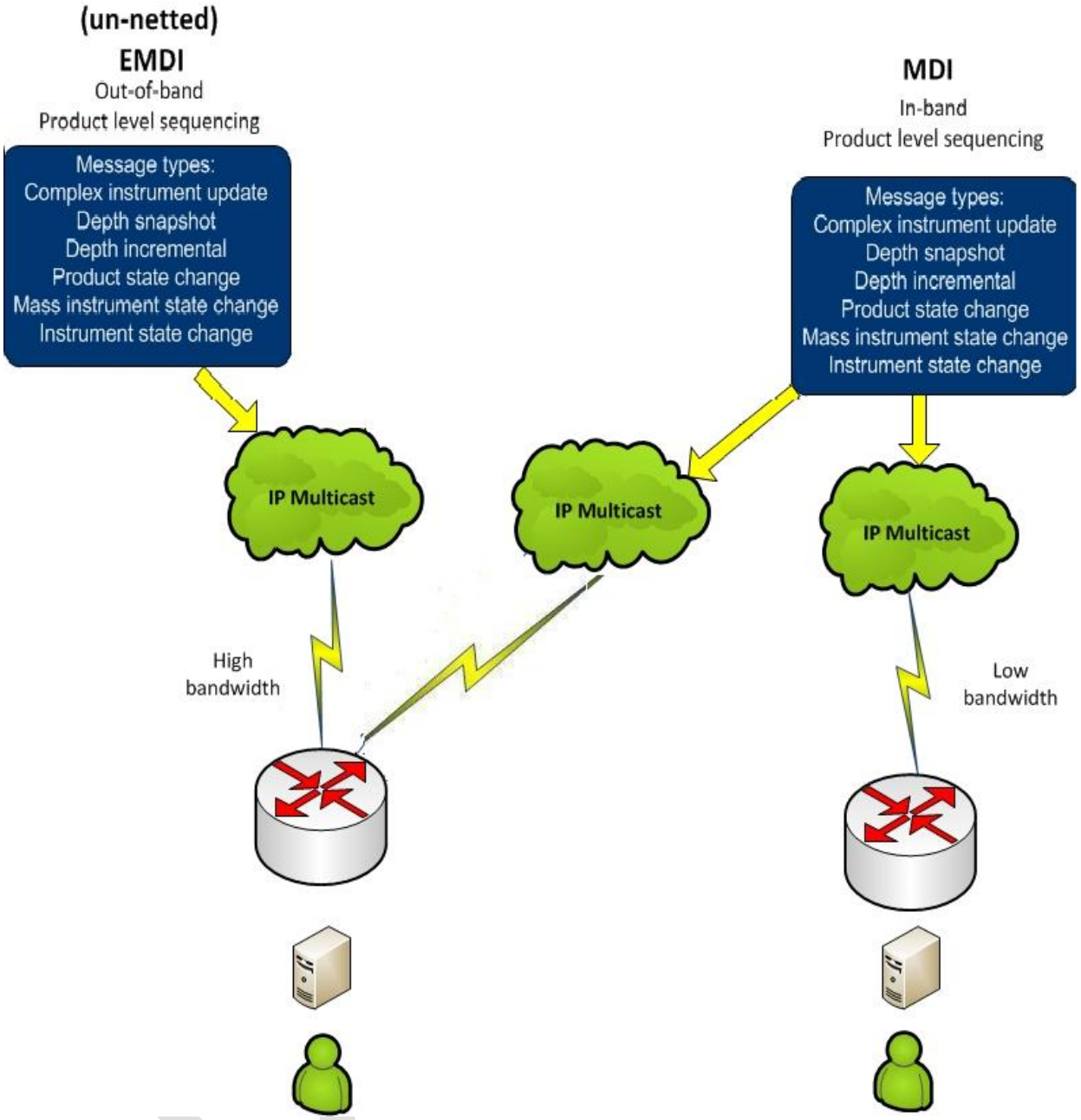


**Figure 1:** Market data interfaces

### 2.3.1 Market data interfaces

The BSE EMDI and the BSE MDI disseminate public market data information in the form of incrementals (event driven) and snapshots (time driven).

The **market data snapshot feed can** be used to recover lost market data or build up the current order book. Receiving applications are not expected to be permanently subscribed to this feed.

The **market data incremental feed should** be subscribed throughout the trading day for receiving order book updates. All incoming messages should be applied to the copy of the order book maintained by the member applications in order to have the latest information.

## 2.4 Interface version number

Each of the interfaces described in this manual has a version number. The version numbers are also listed within the FAST XML templates. This manual relates to the following interface version numbers:

- BSE EMDI: 000.000.021-1000510.83
- BSE MDI: 000.007.011-1000510.83

The version numbers for the interfaces are available at the beginning of the FAST XML files.

## 2.5 BSE customer support

BSE support is available 08hrs on business days and may be contacted as follows:

| | BSE Contact List | |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| | | |

**Table 1:** BSE contact list

## 2.6 Further reading matter for this topic

This document is designed as an independent learning and reference manual. However, for background information related to network connectivity, FAST/FIX messages further documents are recommended.

The documents listed below provide useful information.

**FAST- and FIX-related documents:**

- **FAST specification documents**: Explains all FAST rules in detail. FAST 1.2 is the summary of the FAST 1.1 specifications plus the extension Proposal. www.fixprotocol.org > fastspec
- **FIX specification documents**: FIX-messages and FIX-tags www.fixprotocol.org > Specifications
- **FIX-Tags**: Specifies all FIX-Tags www.fixprotocol.org > FIXimate3.0

## 2.7 How to read this document

This manual covers the BSE EMDI and BSE MDI. Differences in function-ality between the BSE EMDI and the BSE MDI are described in separate sub sections, while being represented by different text colors: **(BSE EMDI)** and **(BSE MDI)**.

For example, section 7.4.2, Recovery (BSE MDI), refers to the "netted" BSE MDI only. Participants who are interested in the "un-netted" BSE EMDI can ignore this sub chapter. This document consists of three parts:

- **Part** I (General Overview) introduces the interface for beginners.
- **Part** II (How to guide) provides methods and hands-on guidance.
- **Part** III (Reference) is a comprehensive reference with details on various message layouts in table format. A typical table would be the following:

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>35</td><td>MsgType</td><td>Y</td><td>string</td><td>User defined message<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>U0</td><td>Beacon</td></tr></table></td></tr>
<tr><td colspan="5"><GroupName > (optional) group starts</td></tr>
<tr><td colspan="5"><SequenceName > sequence starts</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td>...</td><td>...</td><td>...</td><td>...</td><td>...</td></tr>
<tr><td colspan="5"><GroupName > sequence ends</td></tr>
<tr><td colspan="5"><SequenceName > (optional) group ends</td></tr>
</table>

**Table 2:** Typical FIX message description

**Interpreting the fields above:**

- **Delivered on**: Specifies the feed which delivers the specific message. A message can be delivered on more than one feed.
- **Tag**: Describes the FIX Tags
- **Field Name**: Describes the FIX-name.
- **Req'd**: Describes whether or not the field is included within the message after FAST-decoding, purely from the FIX-point of view. This does not refer to a FAST-rule, e.g. operators or Presence Map (PMAP) in FAST.
- **Data Type**: FAST data type. This information is also provided in the XML FAST templates.
- **Description**: This column contains an explanation of the FIX-field and it's "valid values" in table format for this particular message.
- **GroupName, SequenceName**: The names correspond with the groups and sequences defined in the FAST XML templates.

Cross references to other chapters within this document and the glossary are provided in blue color.
**Example:** More information is provided in section 9.1, Reference data messages.

In this document, the terms "incrementals" and "snapshots" are used in various contexts. Within this document "incrementals" and "snapshots" refer either to messages of the market data feed. The actual meaning can be inferred from the context.

**Note**: Important statements made in this manual are highlighted with a shadow box.

# 3 Differences between the interfaces

A feed is a message flow of logically grouped messages, e.g. the depth incremental and product state change messages for a particular product are grouped together within the incremental feed of BSE EMDI. The following diagram illustrates the available feeds for the three multicast based public interfaces:

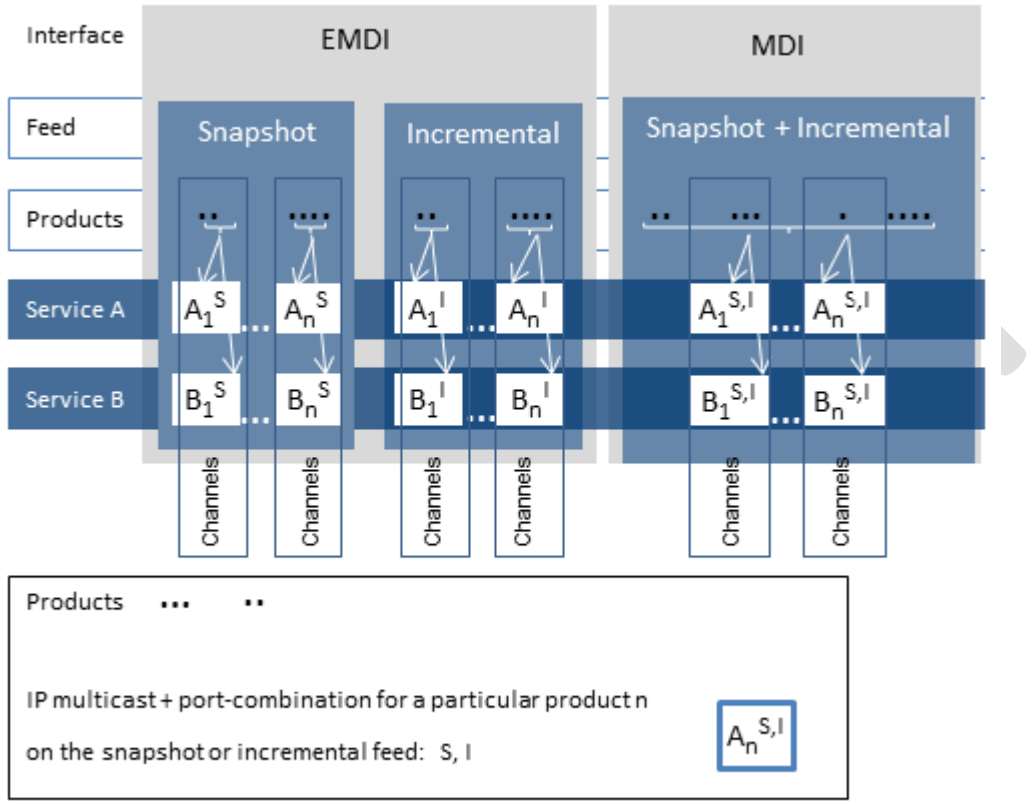


**Figure 2:** Overview of the three interfaces

The BSE EMDI has multiple channels that have either snapshots ($A_1^S$) to ($A_n^S$) and mul- tiple incremental channels ($A_1^I$) to ($A_n^I$). The BSE MDI has the snapshots and incrementals combined over multiple channels ($A_1^{S,I}$) to ($A_n^{S,I}$).

The snapshot and incremental messages for the **BSE EMDI** are delivered via separate feeds (out-of band) and need to be synchronized. Each feed consists of several channels, each of which delivers the information for a group of products.

Several partitions, each with a unique SenderCompID (49), may contribute to the same multicast address as shown in figure 20 on pg. 71. The SenderCompID (49) is unique across all partitions. However, it should not be relied upon as under unlikely but possible conditions on the exchange this is not true.

In contrast to the BSE EMDI, the snapshot and incremental messages for the **BSE MDI** are sent on one feed only (in-band), therefore there is no need to synchronize both messages. The feed is also di-

vided into several channels grouped on product basis.

All feeds are sent on two different multicast addresses via different physical connections (Service A and B). Service A and Service B are identical in terms of the information provided, i.e. the packet contents, sequence numbers and sequence in which packets are sent is the same. This is called "live-live" concept.

Product groups are distributed across several partitions on the BSE backend side. Service A and Service B cannot be published at exactly the same time.

## 3.1 Distribution sequence for BSE EMDI

The rule for the **distribution sequence** across partitions is as follows:

**Even partitions**: Publish on Service A first, then on Service B.
**Odd partitions**: Publish on Service B first, then on Service A.

The above rule is applied by using the field PartitionID (5948). It is available in the product snapshot message and in the packet header and contains the number of the partition for the product of interest. The PartitionID (5948) never changes intraday.

**Example:** A PartitionID = 8 indicates an even partition and therefore Service A is published before Service B.

The time difference between publication of Service A and B is currently not known; lab testing indicates an average time difference of about 10 - 15 µs; the cable length for both Service A and Service B within the co-location is the same, i.e. both services have the same propagation delay.

The multicast addresses for both of these services are disseminated in the product reference information. Due to the inherently unreliable nature of the UDP protocol, data packets may be lost in the transmission network. Therefore members are advised to join both services to reduce the probability of data loss.

## 3.2 Distribution sequence for BSE MDI

The rule for the **distribution sequence** across partitions is as follows:

**Even and odd partitions**: Publish on Service A first, then on Service B.

**Example:** The PartitionID (5948) for BSE MDI is not available in the packet header but in the product snapshot message. However, the PartitionID (5948) doesn't need to be considered because Service A is always published first regardless of the partition.

## 3.3 Choosing between the BSE EMDI and the BSE MDI

Both types of interface, un-netted and netted, provide market information via multicast using a price-level aggregated order book (as opposed to, for example, order-by-order feeds) but they have different bandwidth requirements and service levels.

- The **BSE Enhanced Market Data Interface (un-netted)** disseminates every order book change up to the configured depth and all on-exchange trades without netting. This interface is designed for participants that rely on low-latency order book updates and data completeness. The un-netted market data is partitioned over several channels; each channel provides information about a group of similar products. As the market becomes busier, the number of messages (and therefore bandwidth usage) increases.
- The **BSE Market Data Interface (netted)** has a lower bandwidth requirement compared to the un-netted version. This interface is designed for participants who do not need to see every order book update, this has the advantage of keeping the infrastructure costs low. Snapshot and incremental updates are sent via the same IP multicast address and port combination.

  This interface aggregates the order book changes over a specified time interval. Currently, BSE plans to provide market data for benchmark futures with a netting interval of 0.25 sec and depth of 10. For all other products, a netting interval of 2 sec and only top-of-book market data is envisaged. This interface has less price levels than the BSE EMDI. Furthermore, only statistical information is provided for on-exchange trades as well as the price and quantity of the last on-exchange trade in the netting interval.

The following table shows the main differences between the BSE EMDI and the BSE MDI:

| Area | BSE EMDI | BSE MDI |
|---|---|---|
| In-band/Out-of-band delivery | Incrementals and snapshots are delivered via different channels, i.e. out-of-band delivery. LastMsgSeqNumProcessed in the snapshot feed provides a link between incremental and snapshot feed, as it carries the sequence number of the last message sent on the incremental feed. Snapshots are needed only for start-up/recovery. | Incrementals and snapshots are delivered on the same channel, i.e. in-band delivery. Snapshots might contain new information. A flag (RefreshIndicator) within the snapshot indicates whether it has to be applied or not. LastMsgSeqNumProcessed is not used. |
| Sequence numbers on message level | Messages on the market data incremental feed have their own sequence number range per product; MsgSeqNum's exist on the depth incremental feed only as shown in table 13 on pg. 35. | Messages on the combined market data incrementals + snapshot feed have one sequence number range per product as shown in table 14 on pg. 37. |
| Trade Volume Reporting | Trade Volume Reporting is provided. Each on-exchange trade is reported individually. | Only statistical information (daily high/low price and total traded quantity) and last trade information is provided. |

| Area | BSE EMDI | BSE MDI |
|---|---|---|
| Packet header | A Performance Indicator[4] is provided for incrementals within the Packet Header as shown in figure 21 on pg. 74. | A Performance Indicator does not exist as shown in figure 22 on pg. 75. |
| Functional beacon message | A functional beacon message on a product level including the last valid MsgSeqNum is sent if no other message has been sent for a configured time period. | Snapshots act as functional beacon message, hence no separate functional beacon messages are provided. |

**Table 3:** Main differences between the BSE EMDI and the BSE MDI

# 4 Overview of the BSE Public Interfaces

This chapter describes the public market data provided by the market- and reference data interfaces.

## 4.1 Infrastructure requirements

The BSE market data interfaces disseminate market data over the BSE multicast network. A router which is capable of handling IP multicast is required for accessing this inter- face. The multicast management protocol is IGMPv2. When utilizing IGMPv3, the IGMPv2 compatibility mode must be enabled.

## 4.2 Trading states

State changes are disseminated over both the BSE EMDI and the BSE MDI market data feeds. Trading state information is not communicated over the BSE Enhanced Transaction Interface (BSE ETI) or FIX interface.

The BSE EMDI and the BSE MDI market data feeds follow the FIX protocol for the publication of trading state information. The BSE product and instrument states are displayed by these interfaces as shown in the following tables.

Section 9.8, Trading states for a sample business day illustrates state messages for a typical business day. The hours of operations for the BSE system is provided in Section 4.8, Hours of operation/availability of messages.

### 4.2.1 Product State Changes

The product state is published with a product state change message (FIX TradingSessionStatus, MsgType = h). In this message, the product state can normally be found in the field TradingSessionSubID (625). Only for quiescent product states, the field TradingSessionID (336) must be evaluated additionally to determine the actual product state.

| product state change message | | | |
|---|---|---|---|
| **BSE Product State** | **FIX TradingSessionID (336)** | **FIX TradingSessionSubID (625)** | **FIX TradeSesStatus (340)** |
| Start of Day | 3 = Morning | 7 = Quiescent | 3 = Closed |
| Pre-Trading | 3 = Morning | 1 = Pre-Trading | 2 = Open |
| Trading | 1 = Day | 3 = Trading | 2 = Open |
| Closing | 1 = Day | 4 = Closing | 2 = Open |
| Post-Trading | 5 = Evening | 5 = Post-Trading | 2 = Open |
| End of Day | 5 = Evening | 7 = Quiescent | 3 = Closed |
| Halt | 1 = Day | 7 = Quiescent | 1 = Halted |
| Holiday | 7 = Holiday | 7 = Quiescent | 3 = Closed |

**Table 5:** Product states

A Halt state is additionally indicated by the FIX field TradSesStatus (340) containing the value 1 = Halted. A Fast Market is reported with the same message type using the new FIX field FastMarketIndicator (28828) which can take the values 0 = No or 1 = Yes.

### 4.2.2 Instrument State Changes

The instrument state is published with an instrument state change message (FIX SecurityStatus, MsgType = f) in case of a single instrument, or with a (FIX SecurityMassStatus, MsgType = CO) message in case that all or most of the instruments of a product and of a specific instrument type[7] change their state.

- In the instrument state change message (FIX SecurityStatus, MsgType = f), the instrument state can be found directly in the field SecurityTradingStatus (326).
- In the mass instrument state change message (FIX SecurityMassStatus, MsgType = CO), the instrument state can be found in the field SecurityMassTradingStatus (1679). This message may contain an exception list of instruments that have a different instrument state. The exception list contains the instrument state in the field SecurityTradingStatus (326) for each of these instruments.

| **BSE Instrument State** | **instrument state change message / mass instrument state change message** |
|---|---|
| | **FIX SecurityTradingStatus (326) / FIX SecurityMassTradingStatus (1679)** |
| Closed | 200 = Closed |
| Restricted | 201 = Restricted |
| Book | 202 = Book |
| Continuous | 203 = Continuous |

**Table 6:** Instrument states

The field FastMarketIndicator (28828) is also contained in the mass instrument state change message; each instrument state message also contains the information about whether the product that the instrument belongs to is in a Fast Market state. This implies that a mass instrument state change message is sent when a product is set to Fast Market (or back) without a change in the instrument states.

The status of the instrument (as opposed to the instrument state) distinguishes active and published instruments and is contained in the field SecurityStatus (965).

[7] Instrument types distinguish simple instruments (option series, futures contracts) and various types of complex instruments

## 4.3 Overview of the various message types

The various message types can be divided into "Service Messages" and "Data Messages".

**Service messages:**

- Technical heartbeat message is sent out periodically by the BSE system on every multicast address and on a specific port assigned for the technical heartbeat; it consists of a FAST reset message only. The purpose of the heartbeat message is network related only[8].
- Functional beacon message (BSE EMDI) contains the last valid MsgSeqNum of each product and is only sent on the market data incremental feed when there is no activity in a product for a certain amount of time. No functional beacons are sent for the BSE MDI because the snapshots act as a functional beacon.

**Data messages:**

- Depth snapshot message is used to send a snapshot of all price levels of the order book and statistical information about on-exchange trades. This message can be used whenever the order book needs to be rebuilt.
- Depth incremental message is used to receive updates on the initial order book.
- Product state change message is used to publish the state of the BSE products.
- Mass instrument state change message provides the state information for all instruments of a product. This message can publish different states for instruments of the same product, e.g. in case of a volatility interruption the front month could be in a different state than the back month.
- Instrument state change message provides state information for a single instrument.
-

[8] It is used to keep the Spanning Tree alive.

- 

A detailed description of the message types listed above is given in section 11, Detailed data feed description and layout.

## 4.4 What is not included in these interfaces

The following information is **not** provided via the new interfaces:

- 
- Market Supervision News is **not** provided. This information is available via the BSE ETI in recoverable form.
- 
- Retransmission functionality is **not** provided, but recovery is possible from the respective other service (A or B). In case a message is lost a snapshot can be used to rebuild the order book.

## 4.5 FIX over FAST

FIX messages are sent out in FAST 1.2 encoded format. The receiving software decodes the FAST messages according to the FAST 1.2 rules.

**Note:** FAST 1.2 templates and FAST 1.1 compatible templates are provided.

After the decoding process, the actual FIX message can be built by applying the FIX structure to the decoded message. The detailed process is shown in Part II, FIX/FAST-Implementation.

Participants need a standard FAST template based decoder in order to be able to use the BSE EMDI, BSE MDI. Alternatively participants can use their own FAST decoder implementation.

## 4.6 Freedom of choice

BSE does not need to provide any software for accessing the services offered. The BSE market and reference data interfaces can be accessed using any platform capable of receiving multicast data feeds. Participants can use any operating system, compiler version or programming language in order to develop or use specific third party applications that are tailored to their requirements.

## 4.7 Testing

It is recommended to test the functionality application logic sufficiently in a simulation environment.

Receiving applications must be able to cope appropriately with a variety of BSE service fail-over scenarios. For this purpose, special test scenarios are offered in a simulation environment.

## 4.8 Hours of operation/availability of messages

- BSE is available from approximately **6:00 IST**. It is recommended to start applications between **6:30 IST** and **7:20 IST**
- Market data messages are sent from the time a product changes to the state "Start-Of-Day" and stops when it changes to the state "End-Of-Day". During that period depth snapshots are sent.

  The reference data is independent to any one product state so it has its own schedule.
- Receiving applications are expected to stay connected from product state "Start-Of-Day" until product state "End-Of-Day".

The following table provides further details about the availability of messages per instrument state:

| State | Market Data Orderbook | Market Data State Info |
|---|---|---|
| **Continuous** | Yes | Yes |
| **Freeze** | Yes | Yes |
| **Book** | No | Yes |
| **Restricted** | No | Yes |
| **Closed** | No | Yes |

**Table 7:** Availability of messages per instrument state

# Part II

# How to guide

## 5 FIX/FAST-Implementation

This chapter describes the message structure for the three interfaces. It also provides the basic FAST-rules used by the interface and describes the basic steps from receiving a FAST datagram, decoding it and building FIX-messages out of it.

The FAST 1.2 specification is provided as an extension to the FAST 1.1 specification. The documents can be found under the following links:

FAST Specification (Version 1.1)
www.fixprotocol.org > Technical Specifications > FAST Protocol > FAST Protocol Specifications > FAST Specification Version 1.1

FAST version 1.2 Extension Proposal
www.fixprotocol.org > Technical Specifications > FAST Protocol > FAST Protocol Specifications > FAST Extension Version 1.2

### 5.1 Structure of Messages

The three public interfaces disseminate data in UDP datagrams in network byte order also known as big endian byte order. This includes vector encoded numbers. A UDP datagram has the following structure:

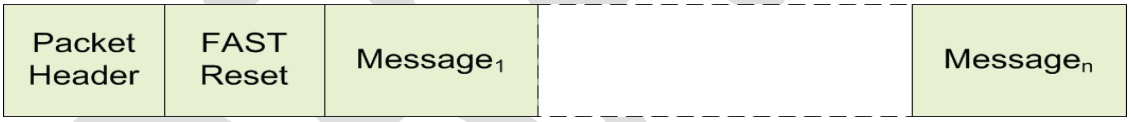


**Figure 3:** Structure of a UDP datagram

- The UDP datagram starts with the packet header message as shown in section 11.1.2.
- Followed by a FAST reset message.
- Followed by the actual message ($Message_1$).
- Possibly followed by one or more messages ($Message_2$ - $Message_n$).

Each message shown in the picture above has the following sub structure:

- PMAP (Presence Map).
- TID (Template ID).
- Data Part.

This is shown in the following diagram:

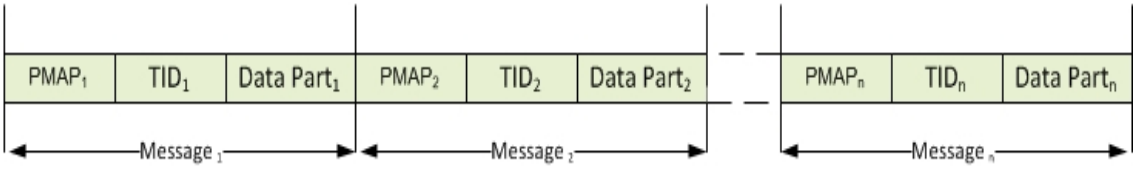


**Figure 4:** Structure of consecutive messages within one datagram

One UDP datagram contains one or more FAST encoded FIX 5.0 SP2 messages. The UDP protocol adds a 28 byte header to every packet (20 byte IP header plus 8 byte UDP protocol header). Due to the unreliable nature of UDP, every UDP datagram is self contained; there is no dependency across datagrams.

## 5.2 FAST terminology

### 5.2.1 FAST reset message

The BSE Market Data Interfaces use **global dictionary** scope for FAST operators[9]. All operators share the same dictionary regardless of the template and application type. The FAST reset message is inserted at the start of every datagram to explicitly reset all the dictionaries.

### 5.2.2 Presence Map (PMAP)

The presence map is a bit combination indicating the presence or absence of a field in the message body, one bit in the PMAP for each field that uses a PMAP bit according to the FAST type. The allocation of a bit for a field in the presence map is governed by the FAST field encoding rules.

### 5.2.3 Template ID (TID)

The template identifier is represented by a number (integer) and points to a specific FAST template which describes the layout and characteristics of the message to be decoded. The FAST XML files are provided in section 13, FAST templates.

FAST uses templates to reduce redundancies within a message by using the following methods:

- The order of fields within the FAST message is fixed, so the field meaning is defined by its position in the message and there is no need to transfer the field tag to describe the field value.
- The templates specify the order and occurrence of message fields like type, presence and operators.

The following list contains the message types and their corresponding template identifiers used with the three BSE interfaces:

[9] The dictionary scope should always be derived from the template definition.

| Message | TID BSE EMDI | TID BSE MDI |
|---|---|---|
| Functional Beacon | 109 | - |
| Packet header for BSE RDI / EMDI / MDI | 111 | 116 |
| FAST Reset Message | 120 | 120 |
| MarketDataReport | - | - |
| ProductSnapshot | - | - |
| InstrumentSnapshot | - | - |
| InstrumentIncremental | - | - |
| | 124 | 105 |
| DepthSnapshot | 93 | 101 |
| DepthIncremental | 94 | 102 |
| | | |
| | | |
| ProductStateChange | 97 | 108 |
| MassInstrumentStateChange | 99 | 104 |
| InstrumentStateChange | 98 | 103 |

**Table 8:** Template identifiers for BSE EMDI/MDI

**Note:** The template id for the packet header will increase in future releases and can be used to identify the software release.

**Example:** The TID=116 indicates the packet header for BSE MDI in the current release. In the next release the TID for the packet header would increase to the next available integer, i.e. TID=117.

### 5.2.4 Dictionaries

A dictionary is a cache in which previous values are stored. FAST operators (-> 5.2.6) make use of the previous values.

### 5.2.5 Stop bit encoding

Most FAST fields are stop bit encoded, each byte consists of seven data bits for data transfer and a stop bit to indicate the end of a field value. An exception from this rule are Byte Vectors as they are used in the packet header of BSE EMDI/MDI.

### 5.2.6 FAST operators

Field operators are used to remove redundancies in the data values. Message templates are the meta-data for the message and are provided earlier. When the messages arrive, the receiving application has complete knowledge of the message layout via the template definition; it is able to determine the field

values of the incoming message.
The following FAST operators are used in BSE EMDI/MDI:

- delta.
- copy.
- constant.
- default.
- increment.

For more information on the new FAST 1.2 features please refer to:
www.fixprotocol.org > Technical Specifications > FAST Protocol > FAST Protocol Specifications > FAST Extension Version 1.2.

## 5.3 Decoding the FAST-message

The FAST messages need to be decoded by means of the FAST templates. The FAST templates provide all necessary information to decode a message such as data types (e.g. uInt32), field names (e.g. MsgType), FIX tags (e.g. 35) and FAST operators (e.g. increment). The FAST templates also contain information about repeating groups (sequences).

A typical example for a XML FAST template with a repeating group is shown in figure 23 of section 14.1, Example for a XML FAST template.

## 5.4 Transfer decoding

Transfer decoding describes the process of how the fields are decoded from the FAST format. For further information, please refer to section 10 of the FAST Specification Version 1.1. Transfer encoding describes the opposite process.

## 5.5 Composing the Actual FIX-Message

A typical FAST decoder would not deliver FIX messages after the decoding process. In order to compose FIX messages, applications need to apply additional rules.

The sequence of FIX-fields after composing the FIX-message on participants' side is not governed by the FIX-layout of the messages, i.e. the fields names of the FIX-message do not need to be in the same sequence. The FIX message, however, needs to fulfill the minimum requirement:

- BeginString(8) in the Standard Header must be the first tag in the message.
- BodyLength(9) in the Standard Header must be the second tag in the message.
- MsgType(35) in the Standard Header must be the third tag in the message.
- CheckSum(10) Standard Trailer must be the last tag in the message.

## 5.6 New features in FAST version 1.2

The following new features from the FAST 1.2 protocol are used:

- **New Type Definition Syntax**: This allows the separation of the "type definitions" from the "type usage" within template definitions.
- **Enumeration**: This feature can be used when there is a fixed set of valid values for a single field.
- **Set (multi-value field)**: This feature can be used when there is a fixed set of valid values which could be sent together as a bit combination instead of using a repeating group. An example for a set would be the field TradeCondition (277) in the Depth incremental message. Sets are used to define the valid values for fields.
- **Timestamp Data Type**: The use of this feature allows native support of time stamp fields which becomes increasingly important for the BSE market data interface. A time stamp is an integer that represents a number of time units since an epoch.

## 5.7 Data types

The BSE implementation of FAST utilizes the following FAST data types:

- Decimal
- Length
- String
- uInt32/uInt64
- Byte vector
- Set
- Enum
- Timestamp

## 5.8 FAST version 1.1 compatible templates

Participants who choose not to upgrade their FAST 1.2 decoders can use FAST 1.1 compatible files offered by BSE. The following needs to be considered:

- **Enumerations**: As described in the previous chapter enumerations have a list of codes. Participants receive an integer but not the description (meaning) of the integer. Since FAST 1.1 does not support enumerations this description of codes needs to be taken from the valid values provided in the FIX tables, chapter 11, Detailed data feed description and layout.
- **Sets**: Similar to enumerations, however, participants receive a bitmap and multiple items from the list. The items need to be taken from the valid values provided in the FIX tables, chapter 11, Detailed data feed description and layout.

The FAST version 1.2 Extension Proposal available at www.fixprotocol.org > fastspec describes how the encoded field (wire format) value looks.

**Example for enumeration**: TradingSessionID (336) can have one of the following values as defined in the FAST 1.2 XML files:

```
<define name="TradingSessionID">
        <enum>
          <element name="1" id="Day"/>
          <element name="3" id="Morning"/>
          <element name="5" id="Evening"/>
          <element name="7" id="Holiday"/>
        <copy/>
        </enum>
</define>
```

The wire format of the values 1, 3, 5, 7 is 0, 1, 2, 3, i.e. each value is represented by an index. Enumerations are not defined in the FAST 1.1 XML files. When the decoder receives a 3 he needs to know that it means "Holiday".

**Example for set**: TradeCondition (277) can have one or more values as defined in the FAST 1.2 XML files:

```
<define name="TradeConditionSet">
        <set>
          <element name="U" id="ExchangeLast"/>
          <element name="R" id="OpeningPrice"/>
          <element name="AX" id="HighPrice"/>
          <element name="AY" id="LowPrice"/>
          <element name="AJ" id="OfficialClosingPrice"/>
          <element name="AW" id="LastAuctionPrice"/>
          <element name="k" id="OutOfSequenceETH"/>
        </set>
```

The wire format of the values U, R, AX, AY, AJ, AW, k is 1, 2, 4, 8, 16, 32, 64, i.e. each value is represented by a different bit. The values can be added together to form combinations of the values. If U, AX are sent then 1 + 4 = 5 are the encoded field values.

Sets are not defined in the FAST 1.1 XML files. When the decoder receives a 5 he needs to know that it is a combination of 1 and 4 which is "ExchangeLast" and "HighPrice".

Before processing any market data, receiving applications need to retrieve technical and functional information. Reference data can be received in file format (Reference Data from File).

At start-up, reference data must be processed to create the initial order book baseline.

## 6.4 Build the initial order book

Participants first have to build the initial order book. The order book has to be maintained per instrument.

> **Note:** Sequence numbers contained in the market data messages are incremented per product.

### 6.4.1 Build the initial order book with the BSE EMDI

For each instrument within the desired products do the following:


**Figure 6:** BSE EMDI initial order book

### 6.4.2 Build the initial order book with the BSE MDI

The following sequence is recommended for the **BSE MDI:**


**Figure 7:** BSE MDI initial order book

The field LastMsgSeqNumProcessed (369) in the BSE MDI snapshots can be ignored because snapshots and incrementals are sent in-band and don't need to be synchronized with each other.

> **Note:** BSE MDI applications must process depth snapshots beside the depth incrementals because the snapshots might contain new information. If the RefreshIndicator (1187) is set the depth snapshot contains order book information that has not been sent in a depth incremental.

## 6.5 Update the order book

Every update in the form of a depth incremental or depth snapshot message contains the price level and the actual price to which the instruction needs to be applied. The receiver application can update information at a particular level with the new information.

Once participants have built the current order book it needs to be continuously updated:

### 6.5.1 Update the order book with the BSE EMDI

As long as the MsgSegNum values for the depth incremental message are contiguous per product do the following[11]:

- Keep applying all depth incremental messages to the current order book.

> **Note**: Depth snapshot messages are sent on a different channel as the depth incremental messages. Changes to the order book are also sent using the depth snapshot messages but the information is also provided with the incremental messages. Snapshot messages don't need to be processed unless the order book needs to be recreated.

[11] The reason is that the unreliable nature of UDP multicast can cause packets to arrive delayed, in incorrect sequence or may be missing.

### 6.5.2 Update the order book with the BSE MDI

As longs as the MsgSegNum values for the depth incremental message are contiguous per product do the following[11]:

- Keep applying all depth incremental as well as depth snapshot[12] messages to the current order book.

Each incremental message can carry different update instructions with the "update action" (New, Change, Delete, Delete From, Delete Thru, Overlay).

**Note**: The depth snapshot messages for the BSE MDI are sent on the same channel as the depth incremental messages. If the RefreshIndicator (1187) is set, changes to the order book are processed into the depth snapshot messages and not provided as separate depth incremental messages.

[12] only if the RefreshIndicator (1187) = Y

# 7 Recovery

Due to the unreliable nature of UDP multicast it is possible that some packets may either be delayed, arrive in the incorrect order or may be missing. Furthermore the UDP packets may be duplicated at the network level. Receiving applications need to be capable of handling these issues. This chapter describes the scenarios which might occur and provides a guideline on how a receiving application needs to react to those scenarios.

Recovery actions are possible on a packet level by using the respective other service (A or B). In case a packet is lost on both services (A and B) clients can create a new current order book by using snapshot information.

## 7.1 Detecting duplicates and gaps by means of the packet header

The packet header allows receiving applications to identify **identical** packets between Service A and Service B. This is achieved by a simple memory comparison on the first 9 bytes for BSE EMDI or 8 Bytes for BSE MDI of a datagram containing SenderCompId and PacketSeqNum as shown in figure 21, Structure of the packet header for BSE EMDI and figure 22, Structure of the packet header for BSE MDI . Another important function of the packet header is to identify **gaps** by means of the PacketSeqNum which can be retrieved just by decoding the packet header.

> **Note:** Packets with the same SenderCompID (field length: 1 Byte) have contiguous sequence numbers per multicast address / port combination.

This means that field PacketSeqNum can be used not only to detect duplicates but also to detect missing packets. PacketSeqNum is a Byte vector and therefore not stop bit encoded as per the FAST specification.

The packet header itself does not contain any product information. In order to find out which product is missing, the product level sequence number must be used in addition to the packet level sequence number; the packet needs to be decoded further down to the message level. This leaves participants with **two recovery options** when a gap in the PacketSeqNum's of the packet header is detected.

**Example**:
A single multicast address carries products FDAX and FGBL, but the participant is only interested in FGBL.

**I. Pessimistic approach:** The receiving application assumes that FGBL is part of the missing packet: It immediately starts recovery actions [13] just by decoding the packet header.

- **Advantage:** Recovery is triggered immediately when observing a missing PacketSeqNum without decoding the entire message.
- **Disadvantage:** The recovery might not be necessary, if FGBL is not part of the message which is inside the lost packet.

**II. Optimistic approach:** The receiving application assumes that FGBL is not part of the missing packet: It waits for the next message on the same service and decodes the packet up to the message level to find out if a packet for FGBL has been lost before triggering recovery actions.

- **Advantage:** This approach allows the participant to recover only products of interest.
- **Disadvantage:** The receiving application needs to wait for the next message. However, the next packet may not contain a message for the product in question.

[13] by means of the other service (live-live concept) or by listening to the depth snapshot

## 7.2 How to recover data via the respective other service (A or B)

Feeds are replicated onto two services, "Service A" or "Service B", and carried on different multicast addresses. This feature provides the possibility to recover missed packets, and participants are advised to join both services.

In each of the following tables, the "Time" column is entirely arbitrary and is intended to show only the sequence of events and in some cases the relative delay between dependent events.

The following table explains the design concept for Service A and B. The table contains the field MsgSeq-Num from the message itself. However, it could also contain the field PacketSeqNum from the Packet Header.

| | Service A: | | | Service B: | |
|---|---|---|---|---|---|
| Time | MsgSeqNum | Message | Time | MsgSeqNum | Message |
| 10:30:00 | 206 | New 151@4 | 10:30:01 | 206 | New 151@4 |
| 10:30:05 | 207 | Delete 151@5 | 10:30:07 | 207 | Delete 151@5 |
| | lost | | 10:30:12 | 208 | New 151@5 |
| 10:30:10 | 209 | New 152@4 | 10:30:13 | 209 | New 152@4 |

**Table 9:** Recovery via Service B (live-live concept)

As the above example shows, the same information is delivered on Service A and B. While MsgSeqNum = 208 is missing on Service A, it is provided on Service B.

Ideally a receiving application processes packets from both Service A and B simultaneously and would take into account the message that arrives first and discardes the second (identical) message.

In the unlikely event that the message has neither been received via Service A nor Service B, the receiver is required to initiate a loss of data scenario:

- The order book needs to be recreated by using the **depth snapshot** messages in conjunction with the **depth incremental** messages. This procedure is similar to the Start Up procedure. Please see section 6.4, Build the initial order book.

## 7.3 Delayed packets

The following example indicates a simple case:

| Time | MsgSeqNum | Message |
|---|---|---|
| 10:30:00 | 132 | New 151@4 |
| 10:30:04 | 133 | Delete 151@5 |
| 10:30:39 | 134 | New 152@4 |

**Table 10:** Packets arriving in correct sequence

In this example, messages arrive in the correct order. The message was not delayed between BSE and the receiving application. There is no special requirement on the application; the message can be processed in the same order as they arrive.

Multicast does not guarantee that the order in which packets are received is the same as the order in which they are sent. For instance, BSE Market Data Interface sends incremental messages in ascending MsgSeqNum order, but they might arrive in an incorrect order at the receiving application.

Consider the following example:

| Time | MsgSeqNum | Message |
|---|---|---|
| 10:30:00 | 206 | New 151@4 |
| 10:30:04 | 208 | Delete 151@5 |
| 10:30:10 | 207 | New 152@4 |

**Table 11:** Delayed Packet 207

In this example, message 207 is delayed within the network, allowing message 208 to arrive first.

A correct communications layer responds as follows:

1. Release message 206 to the application immediately on arrival.
2. On arrival of 208, recognises that 207 is missing.
3. Start an appropriate timed operation to trigger the recovery actions if the out-of-sequence message 207 fails to arrive in a reasonable time.
4. Assuming that 207 arrives within that reasonable time, release 207 and then 208 to the application in that order and cancel the timed recovery action.

## 7.4 Missing packets

All lost packets start life as "delayed" packets, as illustrated in the preceding case. The communications layer of the receiving application is responsible for deciding when to declare a network packet as lost. In the following example it is assumed that MsgSeqNum = 207 from the example above does not arrive within the allowed time. Therefore it is considered as lost:

| Time | MsgSeqNum | Message |
|---|---|---|
| 10:30:00 | 206 | New 151@4 |
| | lost | |
| 10:30:04 | 208 | Delete 151@5 |
| 10:30:10 | 209 | New 152@4 |

**Table 12:** Missing seqNum 207

The correct behaviour in this instance is:

1. Release message 206 immediately on arrival.
2. Hold on to 208 because it is out-of-sequence, and initiate timer-based recovery actions.
3. Hold on to 209 for the same reason. Timer-based recovery actions are already pending for this product, so do not reset the timer.

(a) Even though message 209 is a "New" operation, it may be unsafe to apply 208 and 209 because we do not know what 207 contains.

4. If the missing message (207) fails to arrive within the allowed time:

   (a) Initiate recovery from the respective other service (A or B) for message (207). If this works then release (207) and then all messages with higher MsgSeqNum's.

   (b) In case the recovery from the respective other service (A or B) fails: initiate recovery via snapshots.

### 7.4.1 Recovery (BSE EMDI)

Depth snapshot and depth incremental messages are distributed via separate channels for the EMDI. For instance, depth incremental messages could be sent on multicast address $A_2{}^I$, port x and the snapshot message on multicast address $A_2{}^S$ with port y (see Figure 2, Overview of the three interfaces).

Incrementals are sent whenever there is a change of the order book (event-driven); snapshots are sent periodically in intervals regardless of whether the order book has changed since the last snapshot (time-driven).

Each message sequence number (field: MsgSeqNum) on the market data incremental feed is unique and contiguous by product across messages. Therefore the sequence number can be used to detect losses. If any gap of the arriving sequence numbers is detected and this gap cannot be filled by using the respective other service (A or B) the receiving application should initiate a snapshot recovery.

The following example shows missing depth incremental messages (MsgSeqNum's 208-209) and depth snapshots (with LastMsgSeqNumProcessed) which relate to the missing message. MsgSeqNum's for the depth snapshot do not exist, which is indicated with "N/A" in the table.

| MsgSeqNum | Product | LastMsgSeqNumProcessed | Message Type | Channel |
|---|---|---|---|---|
| 205 | A | | | $A_1{}^I$ |
| 206 | A | | depth incremental | $A_1{}^I$ |
| 207 | A | | depth incremental | $A_1{}^I$ |
| lost | A | | depth incremental | $A_1{}^I$ |
| lost | A | | depth incremental | $A_1{}^I$ |
| 210 | A | | depth incremental | $A_1{}^I$ |
| 1000 | B | | depth incremental | $A_2{}^I$ |
| N/A | A | 209 | depth snapshot | $A_1{}^S$ |
| 211 | A | | depth incremental | $A_1{}^I$ |
| N/A | B | 1000 | depth snapshot | $A_2{}^S$ |
| 1001 | B | | depth incremental | $A_2{}^I$ |

**Table 13:** Snapshots and incrementals within the BSE EMDI

The appropriate recovery action for missing depth incrementals is the same as the logic described in section 6.4.1, Build the initial order book with the BSE EMDI.

There are some additional points to be aware of when performing recovery:

- Depth snapshot messages are not sequenced, but they are still theoretically subject to out-of-order packet delivery. Applications must consider this in determining that their snapshot cycle is complete. The packet sequence number in the packet header can be used to detect out-of-order delivery.
- The LastMsgSeqNumProcessed (369) is not necessarily the same for all instruments belonging to a product on the market data snapshot feed.

**Note:** The market data snapshot feed does not contain any "start" or "end" messages to delineate the cycle.

There are two ways to determine when to leave the snapshot feed during recovery:

**Method 1: Process specific products**

For each SenderCompID (49) contributing to the market data snapshot feed, depth snapshot messages are grouped by product as illustrated below:

$P_1 I_1$ | $P_1 I_2$ | $P_1 I_3$ | $P_1 I_n$ | $P_2 I_1$ | $P_2 I_2$ | $P_2 I_3$ | $P_2 I_n$ | $P_3 I_1$ | $P_3 I_2$ | $P_3 I_3$ | $P_3 I_q$ | [...]

**with:**
$P_n$ : Product n
$I_q$: Simple or complex instrument q for product n

Depth snapshots for instruments in the same product will often all appear in the same packet, but this should not be relied upon as it is not true when the amount of data is simply too great to fit into a single packet, and under certain other technical conditions on the exchange.

A change of product MarketSegmentID (1300) for a given SenderCompID (49) indicates the end of the depth snapshot messages for the respective product. This allows applications to easily determine when they've received a snapshot for every instrument in the products they're interested in and leave the snapshot feed.

**Method 2: Process an entire depth snapshot cycle**

It's also easy for an application to listen to an entire snapshot cycle.

Applications can determine when they've seen an entire snapshot cycle simply by remembering the SecurityID (48) of the first depth snapshot message they saw from each SenderCompID (49).

When they see the same SecurityID (48) again for each SenderCompID (49), they know that a complete depth cycle has been seen and can leave the snapshot feed.

> **Note:** Receiving applications also need to consider depth snapshot messages for newly created complex instruments.
>
> **Note:** If a failover occurs during snapshot processing the SenderCompID (49) for the affected partition changes and the snapshot cycle for that partition starts again.

### 7.4.2 Recovery (BSE MDI)

Snapshot and incremental messages are sent on the same channel and carry a contiguous sequence number (field: MsgSeqNum) per product. The snapshot always carries the latest information and might carry new information, not already sent with an incremental message. The following table shows an example for the distribution of incremental and snapshot messages for two products:

| MsgSeqNum | Product | Message Type | Channel |
|---|---|---|---|
| 5 | A | | $A_1^{S,I}$ |
| 6 | A | depth incremental | $A_1^{S,I}$ |
| lost | A | depth incremental | $A_1^{S,I}$ |
| 25 | B | depth incremental | $A_2^{S,I}$ |
| 8 | A | depth incremental | $A_1^{S,I}$ |
| 9 | A | depth snapshot | $A_1^{S,I}$ |
| 10 | A | depth snapshot | $A_1^{S,I}$ |
| 11 | A | depth incremental | $A_1^{S,I}$ |
| 26 | B | depth snapshot | $A_2^{S,I}$ |
| 27 | B | depth incremental | $A_2^{S,I}$ |

**Table 14:** Snapshots and incrementals within the BSE MDI

If the depth incremental message for product A with MsgSeqNum = 7 is lost, a consistent order book can be rebuilt from the next snapshot message for product A, in this case arriving with MsgSeqNum=9.

All depth incremental messages for product A with a lower sequence number than the next market data snapshot message for product A must be discarded, e.g. MsgSeqNum = 8 (incremental) must be discarded as its effect is included in MsgSeqNum = 9 (snapshot).

Since multicast doesn't guarantee the correct sequence of the incoming message, it is recommended to buffer all incoming incrementals while waiting for the next snapshot message. The buffered incrementals for product A with MsgSeqNum $\geq$ 11 can be applied to the latest snapshot with MsgSeqNum = 10.

> **Note:** LastMsgSeqNumProcessed is not necessary for recovery purposes in the BSE MDI.

# 8 Various time stamps in BSE and how to use them

This section provides a list of each time stamp field in the two BSE Market Data Interfaces, it describes the measurement point and explains what is being measured.

All time stamps are provided in UTC (nanoseconds since "Unix Epoch" (01.01.1970)). While the format is provided in nanoseconds the actual precision of time stamps can be in microseconds. In that case the last three digits of the time stamp field is "000".

## 8.1 Time stamps (BSE EMDI)

The following picture shows all time stamps on the BSE Matching Partition for requests/responses for orders, and execution messages:

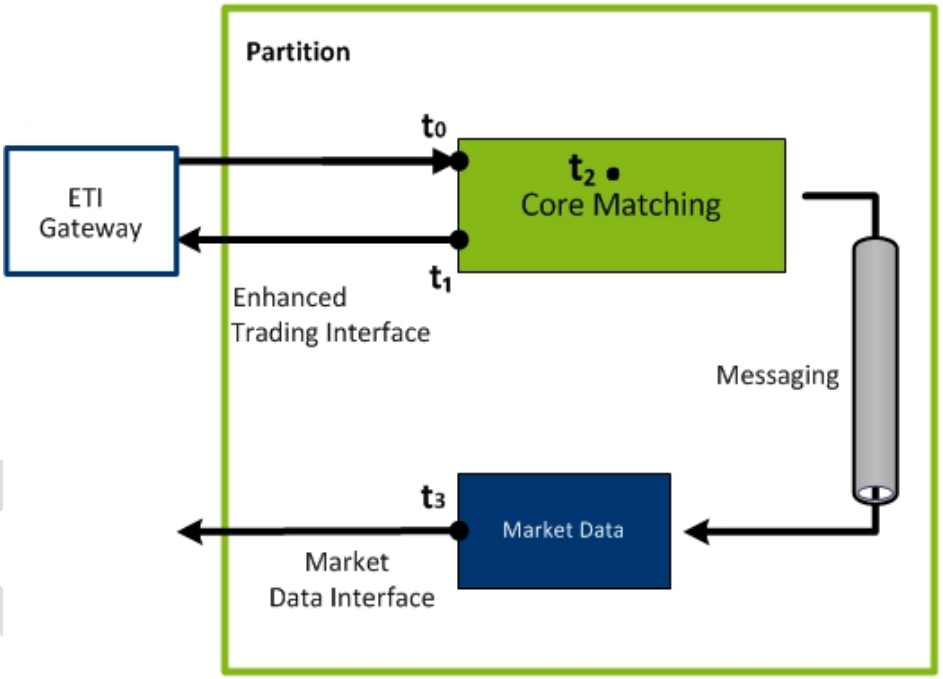


**Figure 8:** Time stamps and their measurement points

| Message | Field Name | Time stamp | Description of time stamp |
|---|---|---|---|
| Packet header | PerformanceIndicator | $t_3 - t_0$ | Time between the arrival of an incoming order/ transaction on the BSE matching engine and send time of the corresponding outgoing market data. The PerformanceIndicator is sent for Incrementals only. |
| | SendingTime | $t_3$ | Time, the BSE Market Data is written onto the socket for the fastest Service (A or B). For more information please see "distribution sequence" on pg. 13. The time stamp is the same for Service A and Service B even though packets on both services are not sent out at the same time. |

| Message | Field Name | Time stamp | Description of time stamp |
|---|---|---|---|
| Depth incremental | MDEntryTime[14] | $t_2$ | Two possibilities:<br>• In case of an order: Time of the last order book update.<br>• In case of a trade: Match time. |
| Depth incremental | AggressorTimeStamp[15] | $t_0$ | Entry time of the incoming order that triggered the trade.<br>This time stamp is only available in case of a trade (MDEntryType=2).<br>•<br>•<br>• |
| Depth snapshot | LastUpdateTime | $t_2$ | Time of the last order book update. |
| | | $t_2$ | |
| Product state change<br>Mass instrument state change<br>Instrument state change | TransactTime | $t_2$ | Time when request was processed by the matcher. |

**Table 15:** Meaning of the time stamps

[14] This field maps to ExecID (17) in BSE ETI order / responses and notifications as well as to TransactTime (60) in the BSE ETI Trade Notification.
[15] This field maps to TrdRegTSTimeIn (21002) in BSE ETI.

## 8.2 Time stamps (BSE MDI)

The field PerformanceIndicator in the packet header message is not available for the BSE MDI. Also the field AggressorTimeStamp (28820) will not have any value. All other fields are the same as for the BSE EMDI.

The field LastUpdateTime in the depth snapshot message contains the time at which the last update was applied to the order book. In the special situation that several orders entered in the middle of a netting interval cancel each other out, this field shows the time stamp of the last order entry even though the net result is that there is no actual change.

An example of when this would happen is the creation and subsequent deletion of an order within the netting interval.

# 9 Important topics with use cases and examples

The following section "Use Cases" describes situations which require special attention. Various examples are provided.

## 9.1 Reference data messages

Reference data provides technical and functional information about all products and instruments available in BSE.

## 9.3 General order book rules and mechanics

The BSE Market Data Interfaces, BSE EMDI and MDI, provide order book updates from level 1 to the maximum level. The order book can be constructed by the depth incremental messages or by the depth snapshot message.
All on-exchange trades and order book updates are reported via the same depth incremental messages. However, trades are always sent out prior to order book updates. The following design principles apply to order book updates:

- Orders are aggregated per price level and are not distributed individually.
- Changes to the book that result from one atomic action in the matching engine are disseminated in one depth incremental message for BSE EMDI.
- Each BSE EMDI packet relates only to a single product. In other words, although each BSE EMDI packet may contain multiple messages, those messages will always relate to the same product. This does not apply to BSE MDI where a single packet may relate to multiple products.
- Price levels are provided explicitly (field: MDPriceLevel (1023)) and do not need to be derived through the price itself.
- During the product states "Start-Of-Day", "" and "End-Of-Day" or when no price levels exist, an empty book (MDEntryType=J) is disseminated for the depth snapshot message (not for incremental). In addition to an empty book, statistical information is sent in "Pre-Trading", "" and in "End-Of-Day".
- An implied price is the only element of the group without a price level (for MDEntryType = 0 = Bid or 1 = Offer). For price levels from 1 to max. price levels, outright prices are distributed. An implied price can either be fully implied or partially implied (for more information please refer to section 9.3.1, Determination of the price sources).
- If two (or more) synthetic prices (with the same price) are created for the Best Market via a different path, the sum of the quantities are reported for the particular price. An example is provided in section 14.2.4, table 49.
- There can be multiple updates in one message. The bid side is updated first followed by the ask side.
- If update instructions "new" or "delete" is sent for an implied price, the order book levels 1-n don't need to be shifted down or up.

[20] A complete snapshot cycle is a combination of start, refdata snapshots, refdata incrementals and end message.

- Order book update instructions are sent for each order book side without a specific order of update actions but ordered by price level instead.
  - from best outright price (price level 1)
  - down to the worst price (max. price level configured per product).
  - if the resulting book depth is larger than the specified maximum product depth only the specified maximum product depth must be saved.
- Intraday expired instrument information is provided by a depth incremental and instrument state change message.
- Only the snapshot and incremental messages of the BSE MDI carry a common and contiguous sequence number per product. The incremental message of BSE EMDI contains a contiguous sequence number per product across all messages, while the snapshot message provides the last sequence number (LastMsgSeqNumProcessed ) sent in the incremental message.
- Only the best implied price is published. The best implied price will be included in market data only in case it is equal to or better than the best direct price in the respective instrument.
- Whenever the quantity or price of the Best Market changes it is disseminated with update action "New" on the incremental feed. Similarly, the Best Market is removed with update action "Delete".

**Note:** The order book is only valid after the entire incremental message has been fully processed.

Figure 16 illustrates a typical order book and terminology used in the following chapters.

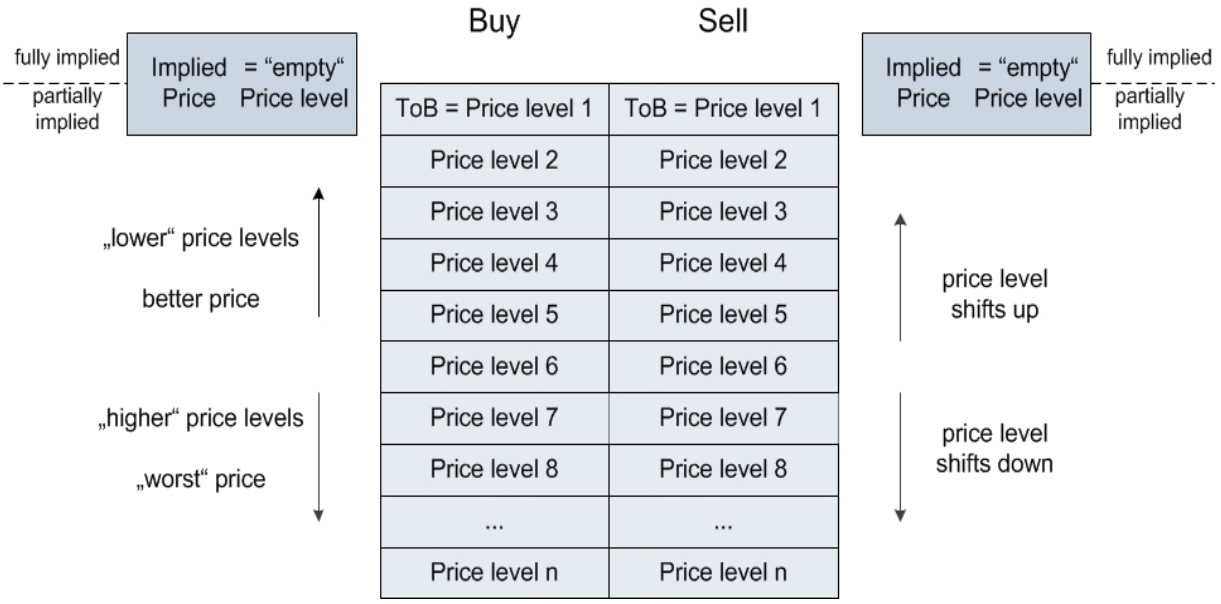


**Figure 16:** Typical order book

An implied price can be either better (fully implied) or the same (partially implied) as price level 1.

### 9.3.1 Determination of the price sources

The new trading platform supports synthetic matching, where the implied prices from complex instruments can create prices equal or better than the best outright price in the instrument. The implied prices are disseminated in the market data in addition to the prices from outright orders. These prices are shown without a price level. The reported quantities for implied prices and level 1 are not aggregated, i.e. quantities on level 1 are fully outright and do not contain any implied components.

The BSE system publishes implied prices in market data only in case it is equal to or better than the best outright price in the respective instrument.

In order to find out which situation applies, a price comparison between the implied price (with empty price level) and level 1 (see figure 16) needs to be done:

1. Implied price is better than the outright price at level one -> Fully Implied.
2. Implied price disseminated is equal to the outright price at level 1 -> Partially Implied.
3. Implied price is deleted or absent -> the Best Market price is fully outright and is the same as on level 1.

Examples for all three cases are provided in section 14.2, Example for determination of the price source.

### 9.3.2 New price level

When a new price level is created in the order book, a depth incremental message is sent with field MDUpdateAction (279) = 0 ("New"). This indicates that:

- The new price level is to be inserted at the specified price level.[21] .
- All existing rows in the order book at the specified and higher levels are to be incremented accordingly.[22] .
- Price levels exceeding the maximum specified depth must not be kept in memory.

**Note:** The field MDPriceLevel (1023) is used to identify which level is being inserted.

**Example:** Buy Limit Order, 10@58.22, enters an empty order book:

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 1068 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 89 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 0 | New |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 8852 | Instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.22** | Price |
| 271 | > MDEntrySize | **10** | Quantity |
| 346 | > NumberOfOrders | 1 | Number of order/ on this level |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | $t_0$ | official time of book entry |

**Table 16:** MDUpdateAction "New"

[21] A MDUpdateAction (279) = 0 ("New") is also disseminated whenever the quantity changes for the implied price (empty price level).

[22] This is not the case if the MDUpdateAction (279) = 0 ("New") is sent for the implied price (with empty price level).

### 9.3.3 Change of a price level

A depth incremental message with MDUpdateAction = 1 ("Change") indicates

- A change at a given price level.
- All fields but the price on the specified side at the price level should be updated.

**Note:** MDUpdateAction="Change" is sent only for depth $\geq 1$ when the price does not change. A MDUpdateAction (279) "Change" contains a price which can be used as a consistency check. However, it never contains a price that is different from the existing one on the current price level.

**Example:** Quantity changed to 8 for limit order above:

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 1069 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 89 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 1 | Change |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 8852 | Instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.22** | Price |
| 271 | > MDEntrySize | **8** | Quantity |
| 346 | > NumberOfOrders | 1 | Number of order/ on this level |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | $t_1$ | official time of book entry |

**Table 17:** MDUpdateAction "Change"

### 9.3.4 Overlay

A depth incremental message with MDUpdateAction (279) = 5 ("Overlay") is used to

- Change the price of a given price level. Other parameters, e.g quantity might also change.

**Note:** MDUpdateAction="Overlay" is sent only for depth $\geq$ 1, i.e. the field MDPriceLevel (1023) must be present. In contrast to the MDUpdateAction="Change" this instruction contains a price change. The overlay function is normally used when there is just one price level disseminated.

**Example:** Buy limit order replaces the best buy limit order during instrument state "Auction":

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 205 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 70 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 5 | |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 63743 | Instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **2.48** | Price |
| 271 | > MDEntrySize | N/A | Quantity remains the same in this example |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | $t_5$ | official time of book entry |

**Table 18:** MDUpdateAction "Overlay"

### 9.3.5 Deletion of a price level

A depth incremental message with MDUpdateAction (279)= 2 ("Delete") is used

- to delete a specified price level.

**Note:** All price levels greater than the deleted one should be decremented. The price of the price level to be deleted is also sent within the message and can be used as a consistency check.

**Example:** Deletion of limit order modified above:

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 1070 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 89 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 2 | Delete |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 8852 | Instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.22** | Price |
| 271 | > MDEntrySize | N/A | Quantity not populated for "Delete" ! |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | $t_2$ | official time of book entry |

**Table 19:** MDUpdateAction "Delete"

### 9.3.6 Deletion of multiple price levels from a given price level onwards

A depth incremental message with MDUpdateAction (279) = 4 ("Delete From") is used to

- Delete all price levels $\geq$ specified price level.

**Note:** All price levels from the specified one and up to the maximum need to be deleted.

**Example:** Deletion of all orders for SecurityID = 8852, MarketSegmentID = 89 from level 3 and above:

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 1068 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 89 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 4 | Delete From |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 8852 | Identifier assigned to each instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.19** | Price |
| 271 | > MDEntrySize | N/A | Quantity not populated ! |
| 1023 | > MDPriceLevel | **3** | Book level |
| 273 | > MDEntryTime | $t_3$ | official time of book entry |

**Table 20:** MDUpdateAction "Delete From"

### 9.3.7 Deletion of multiple price levels up to a given price level

A depth incremental message with MDUpdateAction (279) = 3 ("Delete Thru") is used to

- Delete all price levels from 1 to the specified price level.

**Note:** All higher than the specified price levels are shifted down to fill the gap of the deleted price levels.
**Example:** Deletion of all price levels from 1 to price level 3.

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 35 | MsgType | X | MarketDataIncrementalRefresh |
| 34 | MsgSeqNum | 1068 | |
| 49 | SenderCompID | 75 | Unique id of a sender |
| 1300 | MarketSegmentID | 89 | Product |
| 268 | NoMDEntries | 1 | |
| 279 | > MDUpdateAction | 3 | Delete Thru |
| 269 | > MDEntryType | 0 | Bid |
| 48 | > SecurityID | 8852 | Unique identifier assigned to each instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.22** | Price on level 3 |
| 271 | > MDEntrySize | 10 | Quantity |
| 346 | > NumberOfOrders | 1 | Number of order/ on this level |
| 1023 | > MDPriceLevel | **3** | Book level |
| 273 | > MDEntryTime | $t_4$ | official time of book entry |

**Table 21:** MDUpdateAction "Delete Thru"

## 9.4

### 9.4.1 Manual Trade Entry (by Market Supervision) (BSE EMDI)

The entry consists of

1. TradeCondition (277) field are always set to "Out Of Sequence" (=k).

2. MDEntryType (269) field is always set to "Trade" (=2).

3. MDEntrySize (271) and MDEntryPx (270) is filled with quantity and price of the trade.

4. The updated quantity of the accumulated trade volume for the instrument.

5. MDEntryID (278) and the MDEntryTime (273).

A manually entered trade will not affect the price statistics. Even when the manually entered trade is higher than the daily high price, it does not change the daily high price. For that reason the field Trade- Condition (277) for a manually entered trade must only contain the "Out Of Sequence" attribute.

### 9.4.2 Trade Reversal (by Market Supervision) (BSE EMDI)

A trade reversal is triggered by the Market Supervision in order to delete a trade partially or completely. A trade can only be reversed with its complete quantity.

Deleting a trade affects the Trade Volume Report, sometimes by just adjusting the accumulated trade volume in the involved instruments, sometimes by additionally adjusting one or more price statistics. An incremental for a trade reversal consists of one entry with MDUpdateAction = 2 ("Delete") and at least one entry with MDUpdateAction = 1 ("Change") per involved instrument.

The incremental entry with the MDUpdateAction = 2 ("Delete") provides information about what was reversed.

The entry consists of

1. MDEntrySize (271) and MDEntryPx (270) of the reversed trade.

2. MDEntryType (269) is set to Trade (= 2).

3. MDEntryID (278) (match event identifier) of the reversed trade.

4. MDEntryTime (273) of when the trade reversal request was processed.

The incremental entry with the MDUpdateAction = 1 ("Change") provides information about what was affected by the reversal. The entry consists of

• MDEntrySize (271) and MDEntryPx (270) if a new last price is set or MDEntryPx only if a other price statistic is affected (High, Low, Opening, Closing). If no price statistic is effected, MDEntrySize and MDEntryPx are empty.

• TradeCondition (277), if MDEntryPx is not empty.

• TradeVolume (1020), the new accumulated volume after the reversal.

• MDEntryType (269) is set to Trade (=2).

• MDEntryTime (273) of the updated last trade if TradeCondition (277) contains "Exchange Last".

## 9.5 Trade Volume Reporting (BSE EMDI)

All on-exchange trades executed on the new BSE trading platform are reported via depth incremental messages. The depth snapshot messages contain statistical information about trades only. Trades can be identified in the incremental messages when MDEntryType is set to 2 (Trade).

The BSE EMDI only disseminates information about on-exchange trades. OTC trade information is not disseminated via the market data incremental and market data snapshot feed of the BSE EMDI.

When an order executes against the book at multiple price levels, this is reflected by a matching event with multiple match steps. Each match step has the trades at one price level and is represented by a unique MDEntryID (278) and published in the market data.

The field MDEntryID (278) is a unique id on product level for each business day. A synthetic match can result in more than one trade volume record with the same MDEntryID (278) as shown in use case 3, section 9.5.3 and use case 4, section 9.5.4.

Every match step occurring in the exchange has an identifier in BSE ETI that is provided in the field FillMatchID (28708) in the Execution Report (8), Execution Report (U8) and TrdMatchID (880) in the Trade Capture Report (AE). This identifier allows participants to link trade capture reports and the corresponding execution report of the BSE ETI with the market data incremental feed of the BSE EMDI.

The following 4 use cases illustrate the MDEntryID (278) and how Trade Volume Reporting works:

### 9.5.1 Use case 1: Direct match of simple instruments

An incoming simple order is matched against two orders of the opposite side of the order book on different price levels.

**Incoming buy order, 150@2885, FESX Sep**

**Existing Order book:**

| Bi | As |
|---|---|
| | 50 |
| | 100 |

**Trade Volume Reporting:** Two trades are reported because two different price levels are involved in the matching process: First 50@2884 gets reported due to a higher matching priority of this price level; afterwards 100@2885.

| In | MD | MD | si | Ty | Ag | #B | #S | Ti |
|---|---|---|---|---|---|---|---|---|
| F | 1 | N | 50 | U | B | 1 | 1 | + |
| F | 2 | N | 100 | U | B | 1 | 1 | + |

with:
U = "Exchange last" R = "Opening price" AX = "High price"
AY = "Low price"

AW = "Last auction price"

### 9.5.2 Use case 2: Direct match of complex instruments

An incoming complex order [23] involving 2 legs is matching against the opposite side of the complex in- strument order book.

**Incoming buy order, 100@8, FESX Sep/Dec**

**Existing Order book:**

| B i | A s |
|---|---|
| | 5 0 |
| | 5 0 |

**Trade Volume Reporting:** Only the entire strategy trade is reported and no trade information is published for the individual legs.

| I | M D | M D | s i | T | A t | # B | # S | T - |
|---|---|---|---|---|---|---|---|---|
| S | 3 | N E | 5 0 | U | B U | 1 | 1 | + 5 |
| S | 4 | N E | 5 0 | U | B U | 1 | 1 | + 1 |

## 9.7 Failure of the market data feed/ matching engine

The following chapters explain fail-over scenarios and how receiving applications need to process them.

### 9.7.1 Normal processing

At start-up, the system assigns a unique sender identifier, the SenderCompID (49) to each market data feed. Afterwards the SenderCompID (49) remains constant for a given product during the entire business day. The SenderCompID (49) as shown in section 7.1 is available in the packet header and in the data message[24], e.g. depth incremental or depth snapshot itself.

For each incremental and snapshot message sent by market and reference data feeds:

- the field content for SenderCompID (49) in the packet header and in each data message is always the same.

For each incremental and snapshot message sent by the market data feeds:

- the PacketSeqNum's in the packet header are contiguous per SenderCompID, multicast address and port combination.
- the MsgSeqNum's in the data message are contiguous per product on the incremental feed of the BSE EMDI.
- the MsgSeqNum's in the data message are contiguous per product on the market data feed of the BSE MDI[25].

Figure 17 provides an example for constant SenderCompID's and increasing sequence numbers:

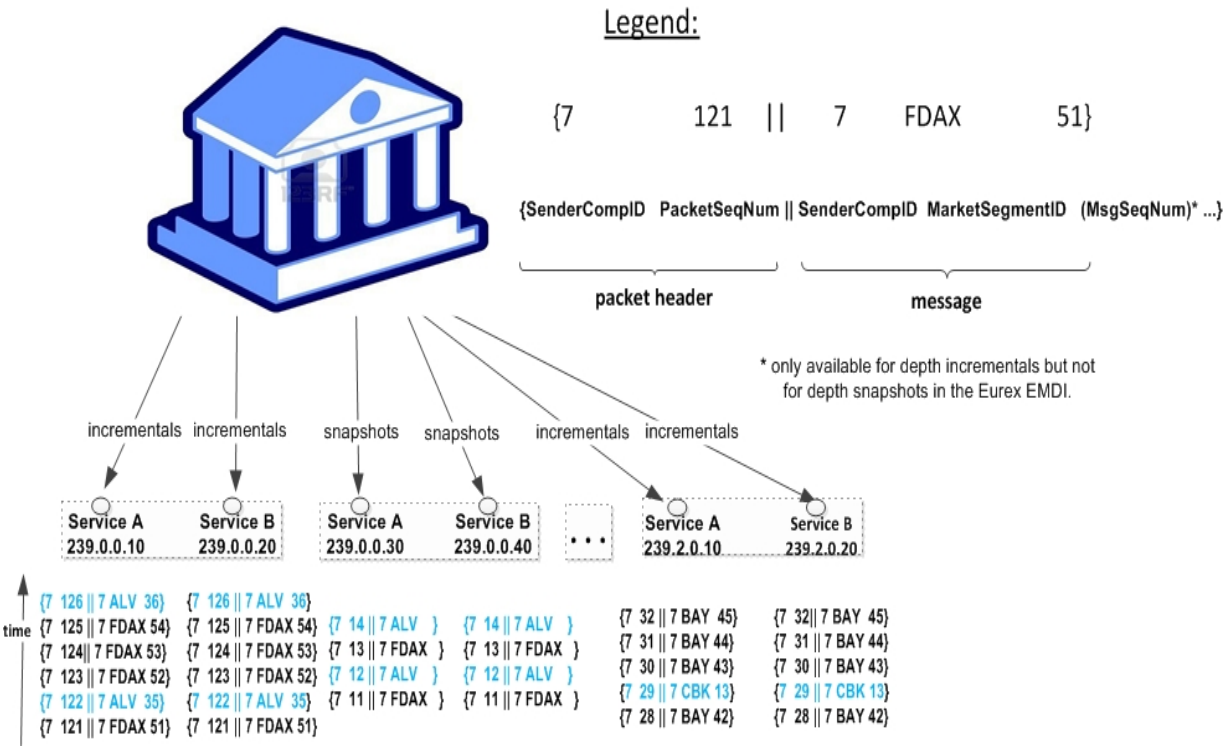


**Figure 17:** Normal processing of sequence numbers for the BSE EMDI

[24] the content is the same.
[25] because the BSE MDI delivers incrementals and snapshots on the same channel.

### 9.7.2 Market data feed fail-over (BSE EMDI)

A new SenderCompID, available in the packet header and in each data message for incrementals and snapshots, indicates a fail-over of the market data feed.

**Incrementals**:

- the PacketSeqNum's in the packet header are reset to 1 and are contiguous per SenderCompID (80), multicast address and port combination.
- the MsgSeqNum's in the data message remain contiguous per product.

**Snapshots**:

- the PacketSeqNum's in the packet header are reset to 1 and are contiguous per SenderCompID (80), multicast address and port combination.

Figure 18 illustrates the different behaviour for incremental and snapshot messages:

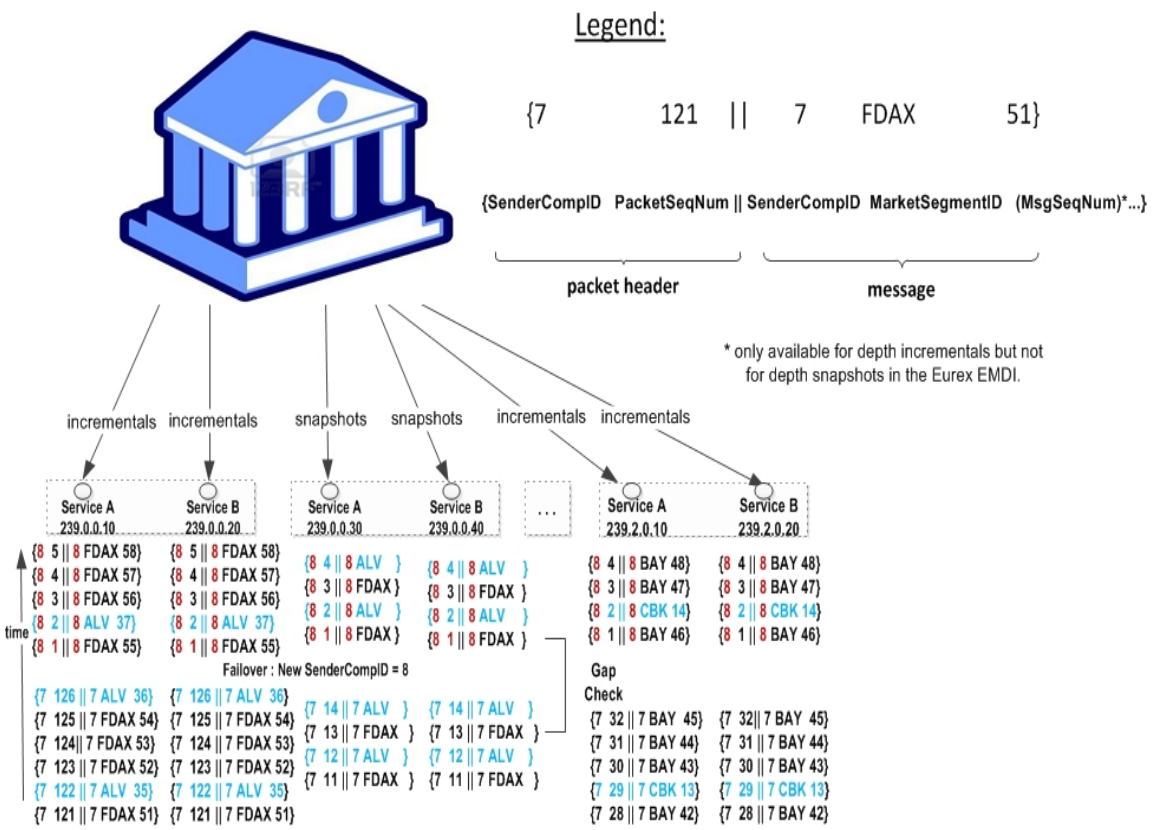


**Figure 18:** Data feed fail-over for BSE EMDI

### 9.7.3 Market data feed fail-over (BSE MDI)

Receiving applications are able to identify a failure as follows:

- the PacketSeqNum's in the packet header are reset to 1 and are contiguous per SenderCompID (80), multicast address and port combination.
- by a change of the SenderCompID (80) in the packet header and in all subsequent messages.
- by a reset of the MsgSegNum's for all products to 1.

The snapshots are sent for all instruments before the incrementals are generated.

Figure 19 illustrates the different behaviour for incremental and snapshot messages:

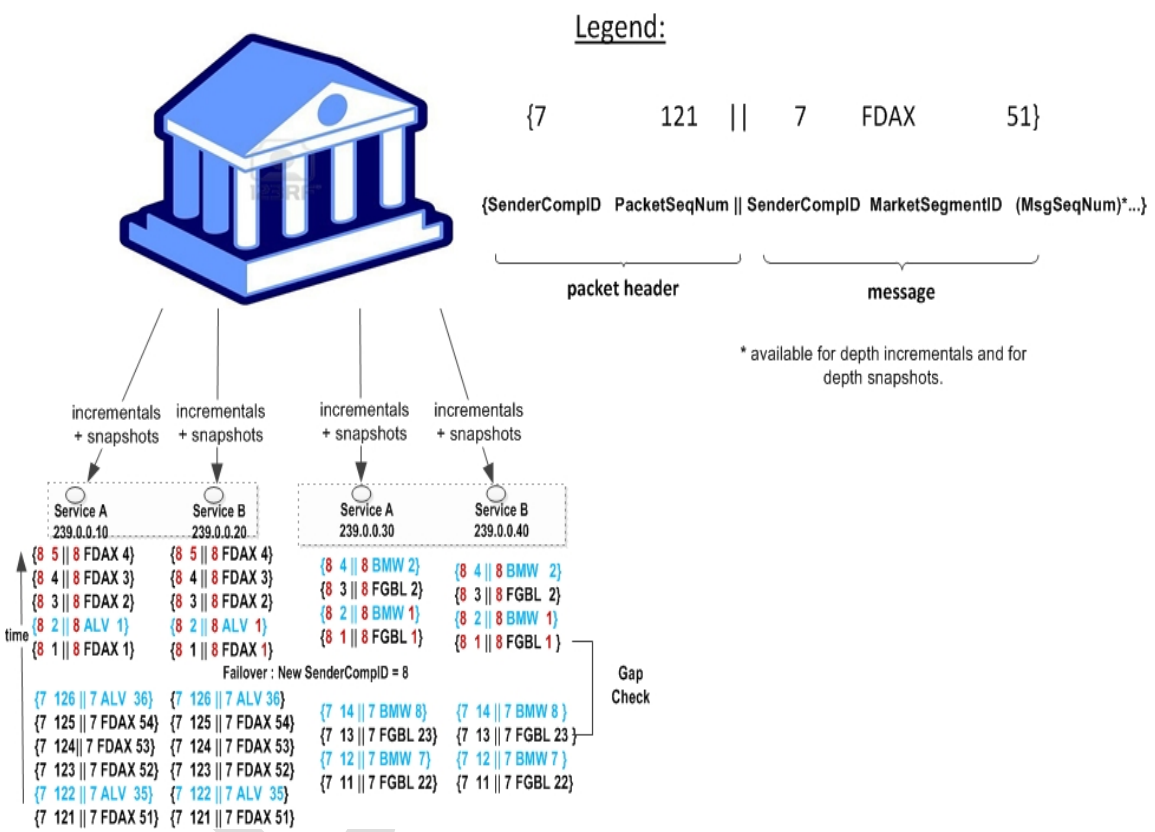


**Figure 19:** Data feed fail-over for BSE MDI

Participants can identify this failover scenario by decoding the packet header of UDP datagram and comparing the SenderCompID value with the previous value.

### 9.7.4 Market data feed restart (BSE EMDI)

A new SenderCompID, available in the packet header and in each data message for incrementals and snapshots, indicates a failure.

**Incrementals**:

- the PacketSeqNum's in the packet header are reset to 1 and are contiguous per SenderCompID, multicast address and port combination.
- the MsgSeqNum's in the data message is reset to 1 and are contiguous per product for incrementals.

**Snapshots**:

- the PacketSeqNum's in the packet header are reset to 1 and are contiguous per SenderCompID, multicast address and port combination.

Once this condition is observed it is safe to assume that a fail-over scenario took place and the only correct action is to rebuild the order book. The receiving application needs to invalidate its view of the order book until an explicit message has been received containing new information. This can either be as a result of a recovery from depth snapshots or from depth incremental messages, as described in section 6.4.1, Build the initial order book with the BSE EMDI.

### 9.7.5 Market data feed restart (BSE MDI)

Receiving applications are able to identify a failure as follows:

- by a change of the SenderCompID (80) in the packet header and in all subsequent messages.
- by a reset of the MsgSegNum's for all products to 1.

The snapshots are sent for all instruments before the incrementals are generated.

Once this condition is observed it is safe to assume that a fail-over scenario took place and the only correct action is to rebuild the order book. The receiving application needs to invalidate its view of the order book until an explicit message has been received containing new information. This can either be as a result of a recovery from depth snapshots or from depth incremental messages, as described in section 6.4.2, Build the initial order book with the BSE MDI.

### 9.7.6 Failure of the matching engine

All non-persistent orders and quotes are deleted. Participants can see a product state change as a result of the market reset. No special processing is necessary for market data applications.

In addition, participants receive a market reset event from their ETI-interface. The service availability message indicates the unavailability of the matcher by the ETI-field MatchingEngineStatus (25005).

## 9.8 Trading states for a sample business day

Section 4.2, Trading states introduced the trading state information. This section describes a typical trading day with the new BSE Exchange Trading Architecture. The example refers to the FDAX future on a typical expiry day. The times for each trading phase are valid for FDAX.

Participants should not rely on any specific order or sequence of messages as described in the following chapters. For instance, the system could send an instrument state change message instead of a mass instrument state change message resulting in the same trading state at the participants' side.

### 9.8.1 Start-Of-Day

The system startup occurs in the morning. Note that with the BSE New Trading Architecture, business days are not technically linked to the local calendar. Under normal circumstances a business date is equal to the local calendar date. Nevertheless it is possible that the system startup and with it the new business day starts before midnight on the previous calendar day.

At startup, the FDAX product goes into the product state "Start-of-Day", while all its instruments are in the state Closed. Traders have no access to the order book.

The system sends a product state change message (FIX TradingSessionStatus (MsgType = h)) with the field TradingSessionID (336) set to 3 = Morning and the field TradingSessionSubID (625) set to 7 = Quiescent. This indicates the product state "Start-of-Day".

The system furthermore sends mass instrument state change message (FIX SecurityMassStatus (MsgType = CO)) with the field SecurityMassTradingStatus (1679) containing 200 = Closed, which indicates that all instruments are in the state Closed. This message is sent once for the futures contracts (specified in the field InstrumentScopeProductComplex (1544) containing 1 = Simple Instrument) and once for futures spreads (specified in the field InstrumentScopeProductComplex (1544) containing 5 = Futures Spread) which is the only complex instrument type supported for futures.

The reference data feed begins with the system startup. Instruments that are scheduled to expire during the day are included in the reference data, but instruments that have already expired on a previous business day are no longer included in the reference data.

### 9.8.4 Continuous Trading

At **8:00 CET**, the opening auction period of the FDAX product ends and continuous trading starts. There is no product state change involved, but all the instruments transition to the instrument state Continuous. The change of the instrument state implies an auction trade if the order book was crossed. This applies also to the complex instruments (futures spreads), even though they had no formal auction call phase before.

In the instrument state Continuous, traders can maintain their orders and quotes. Incoming orders and quotes are continuously matched. The system publishes order book and trade data.

The system sends two mass instrument state change messages with the field SecurityMassTradingStatus (1679) containing 203 = Continuous, which means that all instruments are in the state Continuous. This message is sent once for simple instruments and once for futures spreads.

### 9.8.5 Intraday Expiry

At **13:00 CET**, the front month contract of the FDAX future expires on an expiration day. The affected simple instrument goes to the instrument state Restricted. The same happens to all complex instruments

(futures spreads) that have the affected simple instrument as a leg. For these instruments, all quotes are deleted automatically. Traders may delete their orders but not modify them or add new orders.

For the expired simple instrument, the system sends a instrument state change message (FIX SecurityStatus (MsgType = f)) with the field SecurityTradingStatus (326) containing 201 = Restricted, which says that this particular instrument is in the state Restricted. Furthermore, the field SecurityStatus (965) contains the value 4 = Expired.

For each impacted complex instrument, the system sends a instrument state change message with the field SecurityTradingStatus (326) containing 201 = Restricted. However, the field SecurityStatus (965) still contains the value 1 = Active.

### 9.8.8 End-Of-Day

After **22:30 CET**, the FDAX product goes into the product state End-Of-Day. All its instruments change into the instrument state Closed. Traders can no longer access the order book. The exchange system will start the end-of-day processing.

The system sends a product state change message with the field TradingSessionID (336) set to 5 = Evening and the field TradingSessionSubID (625) set to 7 = Quiescent. This indicates the product state End Of Day

The system also sends two mass instrument state change messages with the field SecurityMassTradingStatus (1679) containing 200 = Closed, which means that all instruments are in the state Closed. This message is sent once for simple instruments and once for future spreads.

# 10 Fine tuning client applications

This chapter covers some aspects of application tuning which should be considered during the design process of receiving applications.

## 10.1 Buffer size

Messages need to be buffered and sorted in order to deal with datagrams arriving in reversed order. A bigger buffer size usually slows down the processing of messages and should therefore be avoided. Conversely, receiving applications might falsely declare a message as lost if the buffer size is too small. As you can see from this example, a bigger buffer size works contrary to the speed of an application but reduces the chances of "lost" messages.

Another factor which effects the ideal buffer size is the ratio of joined multicast streams to available bandwidth of an BSE Market Data connection. A connection which operates at high network utilization levels might more often cause multicast drops or packets arriving in an incorrect sequence.

Last but not least, the location of the receiving application also matters. For instance, an application running in co-location has very few out-of-order multicast packets (none in most cases) while an application which is located at a far distance from the BSE host receives a few packets out-of-order.

Therefore a general recommendation concerning the buffer size cannot be made. Developers need to determine the ideal buffer size during internal testing. Please take into account that the message rate for the public broadcast is usually much lower in the simulation environment than it is in production.

## 10.2 Packet and message processing

It is important that messages are removed from the network in a timely fashion to prevent them from being dropped by the client machine due to "receive buffer" overflow in the IP stack. In addition to the removal of messages from the network stack (as might be performed in response to a select() operation, for example), this design requires a time-based component to determine when a missing packet is declared lost (as opposed to simply delayed).

The mechanism behind this is an implementation detail, and is platform-specific, but in its simplest form a timed select() and polling of an internal list of overdue packets would suffice.

The actual time out value applied is very implementation-specific, and may be either determined dynamically (with a knowledge of when the first overdue packet is declared lost) or a simple static value.

**Note:** Depth incremental messages must not be applied to the order book unless they are in sequence.

For each network packet received, decode it into the constituent FIX message and then for each message:

The market data feeds may contain information about multiple products. If it is not for a product that the clients application is interested in:

- Throw it away.

If the message is already in the cache:

- The clients application already received this message from the mirror channel, or it has been duplicated in the network.
- Throw it away.

Otherwise:

- Add it to the cache.

## 10.3 Application level

Various approaches can lead to faster processing on application level. The approaches depend primarily on the purpose and algorithm of the application.

### 10.3.1 Discarding duplicate packets witin the live-live environment

It is expected that receiving applications process packets from Service A and B simultaneously. The concept of the packet header allows receiving applications to detect duplicates based on the PacketSeqNum. It is recommended to discard a packet after decoding the packet header once it has been identified as duplicate. The actual message following the packet header no longer has to be decoded, allowing a faster processing speed.

### 10.3.2 Order book processing

Depth incremental messages deliver changes of the order book from ToB to worse price levels. Trading algorithms which are based on fast matching without the knowledge of the order book could process ToB only before making a decision and process the order book afterwards.

Conversely, trading algorithms with a matching logic based on the knowledge of the order book need to process the order book before sending orders/quotes.

### 10.3.3 Optimal processing of desired products (BSE EMDI)

Receiving applications interested in certain products need to join a multicast address which contains the desired products according to the mapping table provided in the reference data. Packets may arrive from different partitions on the same multicast address as shown in figure 20. The PartitionID in the packet header for the BSE EMDI can be used to identify packets arriving from partitions which carry the desired products. All other packets can be easily discarded without decoding the entire message.

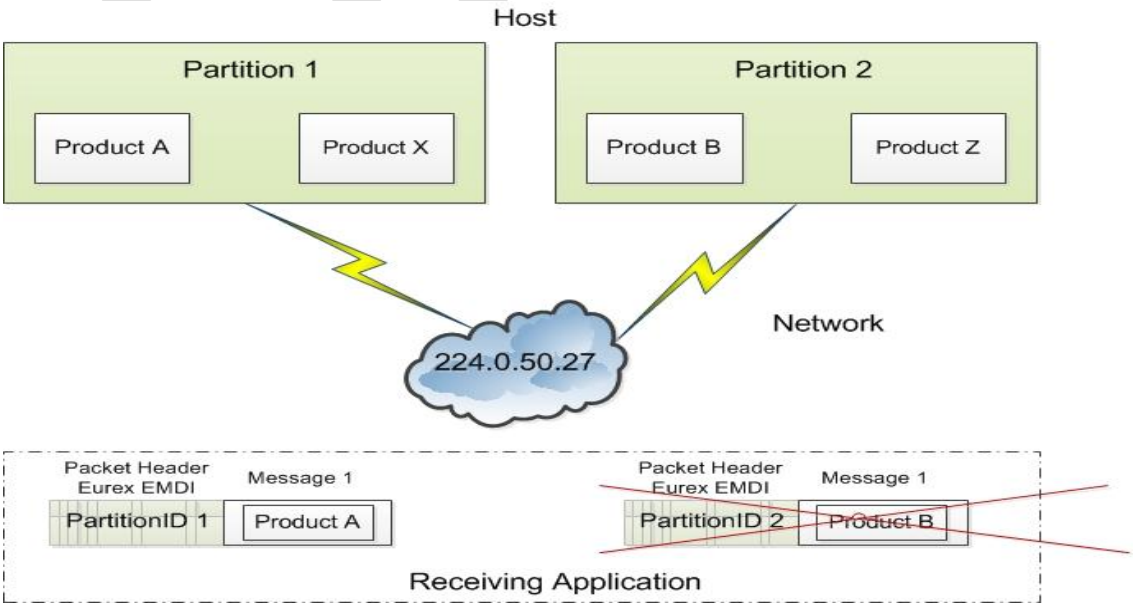


**Figure 20:** Discarding packets with unwanted products based on the PartitionID of the BSE EMDI packet header

The example provided in figure 20 shows two products arriving on multicast address 224.0.50.27. The participant is only interested in product A. Packets containing product A or product X can be identified by the field PartitionID in the packet header. As product X is not one of the desired products it can be discarded after decoding the message.

# Part III

# Reference

## 11 Detailed data feed description and layout

This chapter provides message layouts and field information. It is structured by service messages, data messages and data files.

### 11.1 Service messages

Service messages do not carry any market information. These messages are sent for the purpose of synchronization or to indicate the status of the service.

#### 11.1.1 FAST reset message

The template with ID = 120 is not included in the "FAST Message Templates" file. This TID is reserved in the main FAST specification and allocated by the FAST Session Control Protocol specification (SCP 1.1[26] )

> **Note:** A conforming decoder must be able to deal with the FAST reset message even though it is not mentioned in the template file. Once the FAST reset message is sent out, the dictionary needs to be initialized.

#### 11.1.2 Packet header (BSE EMDI)

**Delivered in: Every UDP-datagram**

The packet header is a technical header used for identification of datagrams and is sent on a channel basis. Every partition stamps outgoing datagrams with a sequence number (field: PacketSeqNum).

One method to identify duplicates between Service A and B is by the use of the field PacketSeqNum which is unique per senderCompID; a faster way is to perform a memory comparison on the first 9 bytes of the packet header.

This method eliminates the need to even decode the header in order to determine, if it has already been processed. This is especially useful to applications using both Service A and Service B feeds, allowing them to determine that a packet has already been processed without incurring any decoding overhead at all.

[26] www.fixprotocol.org/fast > FAST Session Control Protocol.

As the packet header message is not defined in the FIX standard, the FIX Tags for PacketSeqNumber, SendingTime and PerformanceIndicator are not shown in the table below. The following layout is available after FAST decoding of the packet header:

| Field Name | Data Type | Description |
|---|---|---|
| PartitionID | uInt32 | Sending partition. |
| SenderCompID | uInt32 | Unique id for a sender. |
| PacketSeqNumber | byte vector | Datagram sequence number. |
| SendingTime | byte vector | Time when market data feed handler writes packet on the wire. |
| PerformanceIndicator | byte vector | Current load of system. Time difference between the incoming ETI-order/quote and the time the market data is written to the socket. This information is provided for the incremental feed of BSE EMDI only and is not provided for the BSE MDI. |

The following picture shows the structure of the packet header before FAST-decoding :

| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 1 Byte | 4 Bytes | 1 Byte | 8 bytes | 1 Byte | 4 Bytes |
|---|---|---|---|---|---|---|---|---|---|
| PMAP | TID | Parti-tionID | Sender Comp ID | Length | PacketSeqNum | Length | SendingTime | Length | PerformanceIndicator (only for un-netted feed) |
| 1 | 2 | 3 | 4 | 5 | 9 | 10 | 18 | 19 | 23 |

**Figure 21:** Structure of the packet header for BSE EMDI

The last three fields are byte vectors with fixed length. Byte vectors are not stop bit encoded according to the FAST standard. Each of them are preceded by a FAST encoded 1 Byte length field as per the FAST specification for byte vector fields.

**Note:** The field PerformanceIndicator including the length field is only available in messages on the BSE EMDI incremental feed. The PartitionID is available in messages on both incremental and snapshot feed of the BSE EMDI.

### 11.1.3 Packet header (BSE MDI )

**Delivered in: every UDP-datagram**

The packet header of BSE MDI doesn't contain the PerformanceIndicator with length field and the PartitionID. The rest of the packet header is identical to BSE EMDI. Duplicates between Service A and Service B can be detected by a memory comparison on the first 8 bytes of the packet header.

| Field Name | Data Type | Description |
|---|---|---|
| SenderCompID | uInt32 | Unique id for a sender |
| PacketSeqNumber | byte vector | Datagram sequence number |
| SendingTime | byte vector | Time when market data feed handler writes packet on the wire. |

Wire representation:

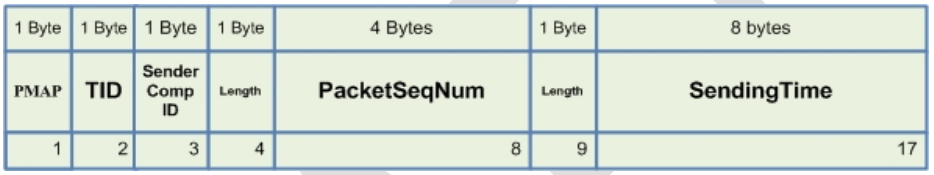


**Figure 22:** Structure of the packet header for BSE MDI

### 11.1.4 Functional beacon message

**Delivered on: BSE EMDI incremental only**

The functional beacon message is sent as a "line active" indicator whenever there are no messages generated on the EMDI incremental feed for the respective product within the last 10 seconds in production.

Functional beacons are sent once the market data service becomes available. If no messages have been sent on the incremental feed for a product then LastMsgSeqNumProcessed (369) is set to zero.

US-customers receive a functional beacon for US-tradable products only.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: 0 – Beacon |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 50 | SenderSubID | Y | uInt32 | Product Identifier, e.g. 89. |
| 369 | LastMsgSeqNumProcessed | Y | uInt32 | Last sequence number on the incremental feed for this product. |

### 11.1.5 Technical heartbeat message

**Delivered on: every channel for BSE EMDI, BSE MDI**

The technical heartbeat message is sent out periodically on every multicast address and consists of a FAST reset message (TID=120) only. The sole purpose of the technical heartbeat message is to keep routing trees alive, i.e. this message prevents routers from dropping multicast packages.

## 11.3 Market data messages

The market data feeds disperses public market data via the BSE EMDI and the BSE MDI. The exchange is able to define the contents of the feeds, the multicast addresses, format of the feed and map exchange offered messages to the feeds.

Public market data for all instruments is distributed over preconfigured multicast addresses. It is possible to configure multiple instruments over one multicast address and the depth of information to be disseminated can be configured on a per product basis. The multicast address and port combinations is different for the BSE EMDI and the BSE MDI.

Two different messages are used for order book updates: The depth incremental is sent if the order book changes (driven by an order book event). Conversely, the depth snapshot is sent in certain intervals independent from any change in the order book (time driven).

The message layout for the BSE EMDI and BSE MDI is the same.

### 11.3.1 Depth snapshot message

**Delivered on: BSE EMDI snapshot feed, BSE MDI data feed**

This message provides periodic updates for orders and trades independent from any change of the order book. Updates are available up to the maximum depth defined by the exchange in the field MarketDepth (264). The Snapshot can be synchronized with the incremental message as described in chapter 6.5, Update the order book. One message per instrument with pre- and post trade data is sent. An empty book is disseminated during the product states as indicated in chapter 9.3, General order book rules and mechanics, bullet 5.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: W – MarketDataSnapshotFullRefresh<br>**Value** / **Description**: Y – MandatoryRefresh; N – OptionalRefresh |
| 34 | MsgSeqNum | N | uInt32 | Not used by unnetted feeds (EMDI) where field is never present. The sequence number of the message is incremented per product across all message types. |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 369 | LastMsgSeqNumProcessed | N | uInt32 | Not used by netted feed (MDI) where field is never present. Last message sequence number sent regardless of message type. |
| 1187 | RefreshIndicator | N | Refresh-Indicator (enum) | Used by netted feed (MDI) only. If set then the depth snapshot information has not been sent with the depth incremental before.<br>**Value** / **Description**: M – Marketplace-assigned Identifier |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| 48 | SecurityID | Y | uInt64 | Instrument identifier, e.g. "8852". |

| | | | | |
|---|---|---|---|---|
| 22 | SecurityIDSource | Y | string | Source Identification. |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>965</td><td>SecurityStatus</td><td>Y</td><td>MDStatus (enum)</td><td>Status of the instrument.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Active</td></tr><tr><td>4</td><td>Expired</td></tr><tr><td>9</td><td>Suspended</td></tr></table></td></tr>
<tr><td>779</td><td>LastUpdateTime</td><td>Y</td><td>timestamp</td><td>Time of last change for SecurityID (nanoseconds).<br>This can be any trade, change of the orderbook on any price level, or also a product or instrument state change information conveyed in this message.</td></tr>
<tr><td colspan="5"><MDSshGrp> sequence starts</td></tr>
<tr><td>268</td><td>NoMDEntries</td><td>Y</td><td>length</td><td></td></tr>
<tr><td>269</td><td>> MDEntryType</td><td>Y</td><td>MDEntry-Type (enum)</td><td>Q = "auction clearing price" is sent as indicative information during the auction.<br><br>J = Empty Book is sent during product states "Start-Of-Day", "Pre-Trading", "Post-Trading" and "End-Of-Day" or when no price levels exist.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Bid</td></tr><tr><td>1</td><td>Offer</td></tr><tr><td>2</td><td>Trade</td></tr><tr><td>J</td><td>EmptyBook</td></tr><tr><td>Q</td><td>AuctionClearingPrice</td></tr></table></td></tr>
<tr><td>828</td><td>> TrdType</td><td>N</td><td>TrdType[29] (enum)</td><td>Defines when the trade happens.<br><br>Only present for MDEntryType=2 and TradeCondition=AW.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>BlockTrade[29]</td></tr><tr><td>2</td><td>EFP[29]</td></tr><tr><td>12</td><td>ExchangeForSwap[29]</td></tr><tr><td>55</td><td>ExchangeBasisFacility [29]</td></tr><tr><td>1000</td><td>VolaTrade[29]</td></tr><tr><td>1001</td><td>EFPFinTrade[29]</td></tr><tr><td>1002</td><td>EFPIndexFuturesTrade[29]</td></tr><tr><td>1100</td><td>OpeningAuctionTrade</td></tr><tr><td>1101</td><td>IntradayAuctionTrade</td></tr><tr><td>1102</td><td>VolatilityAuctionTrade</td></tr><tr><td>1103</td><td>ClosingAuctionTrade</td></tr><tr><td>1104</td><td>CrossAuctionTrade</td></tr></table></td></tr>
<tr><td>336</td><td>> TradingSessionID</td><td>N</td><td>Trading-SessionID (enum)</td><td>Attached to MDEntryType 2=Trade and TradeCondition U=Exchange Last unless there has been no trade so far. In this case it is attached to book information which can simply be Q=Auction Clearing Price or all bids and offers for the currently visible depth (uncrossed book).</td></tr>
</table>

[29] The values 1, 2, 12, 55, 1000, 1001, 1002 are OTC related and are therefore never sent in BSE EMDI/MDI.

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td></td><td></td><td></td><td></td><td>If there are no trades and no book information then it is attached to J=Empty Book.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Day</td></tr><tr><td>3</td><td>Morning</td></tr><tr><td>5</td><td>Evening</td></tr><tr><td>7</td><td>Holiday</td></tr></table></td></tr>
<tr><td>625</td><td>> TradingSessionSubID</td><td>N</td><td>Trading-Session-SubID (enum)</td><td>See description for TradingSessionID.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>PreTrading</td></tr><tr><td>3</td><td>Trading</td></tr><tr><td>4</td><td>Closing</td></tr><tr><td>5</td><td>PostTrading</td></tr><tr><td>7</td><td>Quiescent</td></tr></table></td></tr>
<tr><td>28828</td><td>> FastMarketIndicator</td><td>N</td><td>Fast-Market-Indicator (enum)</td><td>See description for TradingSessionID.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>No</td></tr><tr><td>1</td><td>Yes</td></tr></table></td></tr>
<tr><td>326</td><td>> SecurityTradingStatus</td><td>N</td><td>Security-Trading-Status (enum)</td><td>See description for TradingSessionID.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>200</td><td>Closed</td></tr><tr><td>201</td><td>Restricted</td></tr><tr><td>202</td><td>Book</td></tr><tr><td>203</td><td>Continuous</td></tr><tr><td>204</td><td>OpeningAuction</td></tr><tr><td>205</td><td>OpeningAuctionFreeze</td></tr><tr><td>206</td><td>IntradayAuction</td></tr><tr><td>207</td><td>IntradayAuctionFreeze</td></tr><tr><td>208</td><td>CircuitBreakerAuction</td></tr><tr><td>209</td><td>CircuitBreakerAuctionFreeze</td></tr><tr><td>210</td><td>ClosingAuction</td></tr><tr><td>211</td><td>ClosingAuctionFreeze</td></tr></table></td></tr>
</table>

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>277</td><td>> TradeCondition</td><td>N</td><td>Trade-Condition (set)</td><td>AW is not be combined with any other value and have its own entry in order to convey the auction type through TrdType. In- strument state already changed to continuous when the auction trade is reported.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>U</td><td>ExchangeLast</td></tr><tr><td>R</td><td>OpeningPrice</td></tr><tr><td>AX</td><td>HighPrice</td></tr><tr><td>AY</td><td>LowPrice</td></tr><tr><td>AJ</td><td>OfficialClosingPrice</td></tr><tr><td>AW</td><td>LastAuctionPrice</td></tr></table></td></tr>
<tr><td>270</td><td>> MDEntryPx</td><td>N</td><td>decimal</td><td>Price.</td></tr>
<tr><td>271</td><td>> MDEntrySize</td><td>N</td><td>uInt32</td><td>Quantity.</td></tr>
<tr><td>346</td><td>> NumberOfOrders</td><td>N</td><td>uInt32</td><td></td></tr>
<tr><td>1023</td><td>> MDPriceLevel</td><td>N</td><td>uInt32</td><td>Book level. Absent for implied bid/offer prices.</td></tr>
<tr><td>273</td><td>> MDEntryTime</td><td>N</td><td>timestamp</td><td>Time of entry (nanoseconds) for last trade entry only (Trade-Condition="U").<br>Statistics do not have an official timestamp in the snapshot, even if they happen to be identical to the last trade and be part of the same entry.</td></tr>
<tr><td>1020</td><td>> TradeVolume</td><td>N</td><td>uInt64</td><td>Cumulative volume of units traded in the day.<br>Only sent for MDEntryType 2=Trade.</td></tr>
<tr><td colspan="5"><MDSshGrp> sequence ends</td></tr>
</table>

### 11.3.2 Depth incremental message

**Delivered on: BSE EMDI incremental feed, BSE MDI data feed**

This message provides order book updates and trades. Order book updates are available during Trading and Fast Trading states.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: X – MarketDataIncremental Refresh |
| 34 | MsgSeqNum | Y | uInt32 | The sequence number is incremented per product across all message types on a particular feed. |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| <MDIncGrp> sequence starts | | | | |
| 268 | NoMDEntries | Y | length | |
| 279 | > MDUpdateAction | Y | MDUpdate-Action (enum) | **Value** / **Description**: 0 – New; 1 – Change; 2 – Delete; 3 – DeleteThru; 4 – DeleteFrom; 5 – Overlay |
| 269 | > MDEntryType | Y | MDEntry-Type (enum) | **Value** / **Description**: 0 – Bid; 1 – Offer; 2 – Trade; J – EmptyBook; Q – AuctionClearingPrice |
| 48 | > SecurityID | Y | uInt64 | Instrument identifier, e.g. "8852". |
| 22 | > SecurityIDSource | Y | string | Source Identification. **Value** / **Description**: M – Marketplace-assigned Identifier |
| 270 | > MDEntryPx | N | decimal | Price of market data (trade or order). |
| 271 | > MDEntrySize | N | uInt32 | Quantity of market data (trade or order). |
| 346 | > NumberOfOrders | N | uInt32 | |
| 1023 | > MDPriceLevel | N | uInt32 | Book level. Absent for implied bid/offer prices. |
| 273 | > MDEntryTime | N | timestamp | For bids and offers the official time of book entry, for trades official time of execution (all in nanoseconds). |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td colspan="5"><TradeEntryGrp> (optional) group starts</td></tr>
<tr><td>828</td><td>> TrdType</td><td>N</td><td>TrdType[30] (enum)</td><td>Defines when the trade happens.<br>Only present for MDEntryType=2 and TradeCondition=AW.<br>For trades outside the auctions, this field is not set.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>BlockTrade[30]</td></tr><tr><td>2</td><td>EFP[30]</td></tr><tr><td>12</td><td>ExchangeForSwap[30]</td></tr><tr><td>55</td><td>ExchangeBasisFacility [30]</td></tr><tr><td>1000</td><td>VolaTrade[30]</td></tr><tr><td>1001</td><td>EFPFinTrade[30]</td></tr><tr><td>1002</td><td>EFPIndexFuturesTrade[30]</td></tr><tr><td>1100</td><td>OpeningAuctionTrade</td></tr><tr><td>1101</td><td>IntradayAuctionTrade</td></tr><tr><td>1102</td><td>VolatilityAuctionTrade</td></tr><tr><td>1103</td><td>ClosingAuctionTrade</td></tr><tr><td>1104</td><td>CrossAuctionTrade</td></tr></table></td></tr>
<tr><td>1020</td><td>> TradeVolume</td><td>N</td><td>uInt64</td><td></td></tr>
<tr><td>277</td><td>> TradeCondition</td><td>N</td><td>Trade-Condition (set)</td><td>Defines the type of price for MDEntryPx.<br>Only present for MDEntryType=2.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>U</td><td>ExchangeLast</td></tr><tr><td>R</td><td>OpeningPrice</td></tr><tr><td>AX</td><td>HighPrice</td></tr><tr><td>AY</td><td>LowPrice</td></tr><tr><td>AJ</td><td>OfficialClosingPrice</td></tr><tr><td>AW</td><td>LastAuctionPrice</td></tr><tr><td>k</td><td>Out of sequence</td></tr></table></td></tr>
<tr><td>28819</td><td>> MDGapIndicator</td><td>N</td><td>uInt32</td><td>Reserved for future use.</td></tr>
<tr><td>28820</td><td>> AggressorTimestamp</td><td>N</td><td>timestamp</td><td>Entry time of the incoming order that triggered the trade.<br>Only present for MDEntryType=2.</td></tr>
<tr><td>28731</td><td>> AggressorSide</td><td>N</td><td>Aggressor-Side (enum)</td><td>Side of the incoming order, which created the trade.<br>Only present for MDEntryType=2.<br><br><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Buy</td></tr><tr><td>2</td><td>Sell</td></tr></table></td></tr>
<tr><td>28821</td><td>> NumberOfBuyOrders</td><td>N</td><td>uInt32</td><td>Number of buy orders involved in this trade.<br>Only present for MDEntryType=2.</td></tr>
<tr><td>28822</td><td>> NumberOfSellOrders</td><td>N</td><td>uInt32</td><td>Number of sell orders involved in this trade.<br>Only present for MDEntryType=2.</td></tr>
<tr><td>278</td><td>> MDEntryID</td><td>N</td><td>uInt32</td><td>Represents the match step ID. This field is unique together with MarketSegmentID.<br>Only present for MDEntryType = 2.</td></tr>
<tr><td colspan="5"><TradeEntryGrp> (optional) group ends</td></tr>
<tr><td colspan="5"><MDIncGrp> sequence ends</td></tr>
</table>

[30] The values 1, 2, 12, 55, 1000, 1001, 1002 are OTC related and part of the Extended Reference & Market Data Service.

### 11.3.3 Product state change message

**Delivered on: BSE EMDI incremental feed, BSE MDI data feed**

The product state change message provides permanent updates on the trading state for a particular product.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: h – TradingSessionStatus |
| 34 | MsgSeqNum | Y | uInt32 | The sequence number is incremented per product across all message types on a particular feed. |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| 336 | TradingSessionID | Y | Trading-SessionID (enum) | **Value** / **Description**: 1 – Day; 3 – Morning; 5 – Evening; 7 – Holiday |
| 625 | TradingSessionSubID | Y | Trading-Session-SubID (enum) | **Value** / **Description**: 1 – PreTrading; 3 – Trading; 4 – Closing; 5 – PostTrading; 7 – Quiescent |
| 340 | TradSesStatus | Y | TradSes-Status (enum) | **Value** / **Description**: 1 – Halted; 2 – Open; 3 – Closed |
| 28828 | FastMarketIndicator | Y | Fast-Market-Indicator (enum) | Indicates if product is in state "Fast Market". **Value** / **Description**: 0 – No; 1 – Yes |
| 60 | TransactTime | Y | timestamp | |

### 11.3.4 Mass instrument state change message

**Delivered on: BSE EMDI incremental feed, BSE MDI data feed**

The mass instrument state change message provides the state information for all instruments of a certain instrument type within a product. Where not all indicated instruments are affected by the new state, the exception list (SecurityTradingStatus (326)) is populated with one entry for each such instrument.

Under Fast Market conditions, this message is sent with the FastMarketIndicator (28828) set but the actual state information may not have changed and is simply a re-statement of the previous information.

A state change affecting a single instrument (such as an intraday expiration) does not trigger a mass instrument state change.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: CO – SecurityMassStatus |
| 34 | MsgSeqNum | Y | uInt32 | The sequence number is incremented per product across all message types on a particular feed. |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| 1544 | InstrumentScopeProduct-Complex | Y | Instrument-Scope-Product-Complex (enum) | Instrument type of affected instruments. **Value** / **Description**: 1 – SimpleInstrument; 2 – StandardOptionStrategy; 3 – NonStandardOptionStrategy; 4 – Volatility Strategy; 5 – Futures Spread |
| 1679 | SecurityMassTradingStatus | Y | Security-Mass-Trading-Status (enum) | **Value** / **Description**: 200 – Closed; 201 – Restricted; 202 – Book; 203 – Continuous; 204 – OpeningAuction; 205 – OpeningAuctionFreeze; 206 – IntradayAuction; 207 – IntradayAuctionFreeze; 208 – CircuitBreakerAuction; 209 – CircuitBreakerAuctionFreeze; 210 – ClosingAuction; 211 – ClosingAuctionFreeze |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>28828</td><td>FastMarketIndicator</td><td>Y</td><td>Fast-Market-Indicator (enum)</td><td>Indicates if product is in state "Fast Market". This indicator refers to a product but is provided on instrument level.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>No</td></tr><tr><td>1</td><td>Yes</td></tr></table></td></tr>
<tr><td>60</td><td>TransactTime</td><td>Y</td><td>timestamp</td><td>Time when request was processed by the matcher (nanosec- onds).</td></tr>
<tr><td colspan="5"><SecMassStatGrp>sequence starts</td></tr>
<tr><td>146</td><td>NoRelatedSym</td><td>N</td><td>length</td><td></td></tr>
<tr><td>48</td><td>> SecurityID</td><td>Y</td><td>uInt64</td><td>Instrument identifier, e.g. "8852".</td></tr>
<tr><td>22</td><td>> SecurityIDSource</td><td>Y</td><td>string</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>M</td><td>Marketplace-assigned Identifier</td></tr></table></td></tr>
<tr><td>965</td><td>> SecurityStatus</td><td>Y</td><td>Security-Status (enum)</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Active</td></tr><tr><td>4</td><td>Expired</td></tr><tr><td>9</td><td>Suspended</td></tr></table></td></tr>
<tr><td>326</td><td>> SecurityTradingStatus</td><td>Y</td><td>Security-Trading-Status (enum)</td><td><table><tr><th>Value</th><th>Description</th></tr><tr><td>200</td><td>Closed</td></tr><tr><td>201</td><td>Restricted</td></tr><tr><td>202</td><td>Book</td></tr><tr><td>203</td><td>Continuous</td></tr><tr><td>204</td><td>OpeningAuction</td></tr><tr><td>205</td><td>OpeningAuctionFreeze</td></tr><tr><td>206</td><td>IntradayAuction</td></tr><tr><td>207</td><td>IntradayAuctionFreeze</td></tr><tr><td>208</td><td>CircuitBreakerAuction</td></tr><tr><td>209</td><td>CircuitBreakerAuctionFreeze</td></tr><tr><td>210</td><td>ClosingAuction</td></tr><tr><td>211</td><td>ClosingAuctionFreeze</td></tr></table></td></tr>
<tr><td colspan="5"><SecMassStatGrp> sequence ends</td></tr>
</table>

### 11.3.5 Instrument state change message

**Delivered on: BSE EMDI incremental feed, BSE MDI data feed**

The instrument state change message provides state information for a single instrument. It also informs participants about intraday expirations of instruments. In that case the field SecurityStatus (965) is set to "Expired".

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: f – SecurityStatus |
| 34 | MsgSeqNum | Y | uInt32 | The sequence number is incremented per product across all message types on a particular feed. |
| 49 | SenderCompID | Y | uInt32 | Unique id of a sender. |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| 48 | SecurityID | Y | uInt64 | Instrument identifier, e.g. "8852". |
| 22 | SecurityIDSource | Y | string | **Value** / **Description**: M – Marketplace assigned identifier |
| 965 | SecurityStatus | Y | Security-Status (enum) | **Value** / **Description**: 1 – Active; 4 – Expired; 9 – Suspended |
| 326 | SecurityTradingStatus | Y | Security-Trading-Status (enum) | Trading status of an instrument. **Value** / **Description**: 200 – Closed; 201 – Restricted; 202 – Book; 203 – Continuous; 204 – OpeningAuction; 205 – OpeningAuctionFreeze; 206 – IntradayAuction; 207 – IntradayAuctionFreeze; 208 – CircuitBreakerAuction; 209 – CircuitBreakerAuctionFreeze; 210 – ClosingAuction; 211 – ClosingAuctionFreeze |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>28828</td><td>FastMarketIndicator</td><td>Y</td><td>Fast-Market-Indicator (enum)</td><td>Indicates if product is in state "Fast Market". This indicator refers to a product but is provided on instrument level.<br><table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>No</td></tr><tr><td>1</td><td>Yes</td></tr></table></td></tr>
<tr><td>60</td><td>TransactTime</td><td>Y</td><td>timestamp</td><td>Time when request was processed by the matcher (nanosec- onds).</td></tr>
</table>

# 13 FAST templates

Two versions for FAST templates are offered:

- FAST templates based on version 1.2
- FAST templates compatible with version 1.1

Participants can either use a decoder which has the new FAST 1.2 feature implemented or use their existing decoder based on FAST 1.1.

The FAST XML files are provided by BSE in separate files:

- EMDIFastTemplates*.xml for all messages on the EMDI incremental feed.
- EMDSFastTemplates*.xml for all messages on the EMDI snapshot feed.
- MDIFastTemplates*.xml for all messages on the MDI feed.
- The FAST templates can be downloaded at the member section of the BSE website

at:

www.BSEchange.com > Technology > System Documentation > New Trading Architecture > Release 1.0 > Market Data Interfaces > FAST 1.2 Templates.

# 14 Appendix

## 14.1 Example for a XML FAST template

This example refers to chapter 5.3, Decoding the FAST-message.

```
<template id="86" name="DepthIncremental">
<string name="MsgType" id="35">
<constant value="X" />
</string>
<uInt32 name="MsgSeqNum" id="34">
<increment />
</uInt32>
<uInt32 name="SenderCompID" id="49">
<copy />
</uInt32>
<uInt32 name="MarketSegmentID" id="1300">
<copy />
</uInt32>
<sequence name="MDEntriesGrp">
<length name="NoMDEntries" id="268" />
<field name="MDUpdateAction" id="279">
<type name="MDUpdateAction" />
</field>
<field name="MDEntryType" id="269">
<type name="MDEntryType" />
</field>
<uInt64 name="SecurityID" id="48" />
<string name="SecurityIDSource" id="22">
<constant value="M" />
</string>
<decimal name="MDEntryPx" id="270" presence="optional">
<delta />
</decimal>
<uInt32 name="MDEntrySize" id="271" presence="optional" />
<uInt32 name="NumberOfOrders" id="346" presence="optional" />
<uInt32 name="MDPriceLevel" id="1023" presence="optional">
<delta />
</uInt32>
<timestamp name="MDEntryTime" unit="nanosecond" id="273">
<copy />
</timestamp>
<group name="TradeEntryGrp" presence="optional">
<uInt32 name="TradeVolume" id="1020" presence="optional" />
<field name="TradeCondition" id="277" presence="optional">
<type name="TradeConditionSet" />
</field>
<uInt32 name="GapIndicator" id="8719" presence="optional" />
<timestamp name="AggressorTimestamp" unit="nanosecond" id="8720" presence="optional" />
<field name="AggressorSide" id="5797" presence="optional">
<type name="Side" />
</field>
<uInt32 name="NumberOfBuyOrders" id="8721" presence="optional" />
<uInt32 name="NumberOfSellOrders" id="8722" presence="optional" />
<uInt32 name="MDEntryID" id="278" />
</group>
</sequence>
</template>
```

**Figure 23:** Example for a FAST template with repeating group