Circular no.: MCX/CTCL/005/2026 January 05, 2026

**Trading Market Data Interfaces (APIs) - MCX Market Data Interface (MDI) And MCX Enhanced Market Data Interface (EMDI) – v1.6**

In terms of provisions of the Rules, Bye-Laws and Business Rules of the Exchange and in continuation to Exchange Circular no. MCX/CTCL/088/2023 dated February 09, 2023, and Circular no.: MCX/CTCL/199/2023 dated March 28, 2023.

Trading Members and Empanelled vendors are requested to note that, the Exchange has released MCX Market Data Interface (MDI) and MCX Enhanced Market Data Interface (EMDI) Market Data Interface API - Version 1.6. The details of changes are mentioned in revision history i.e. Addition of values 7 (Upper Circuit Limit) and 8 (Lower Circuit Limit) in field MDEntryType in section 6.3 Depth Incremental Message. Member are requested to make necessary amendments at your end for the aforesaid changes.

The above said changes are available in test environment, Trading Members and Empanelled vendors can use the same to test changes in their application.

Exchange will publish separate circular to communicated go live date.

In case of any queries or clarification on new interfaces document, trading members/vendors are requested to get in touch on following contact details:

- Email – ctcl@mcxindia.com
- Phone: +91 22 – 6649 4040 / 6731 8888

Trading Members and Vendors are requested to take note of the same.

For and on behalf of
Multi Commodity Exchange of India Ltd.

Abhay Angarkar
VP - Technology
Encl.: As above

**Multi Commodity Exchange of India Limited**

# Trading Market Data Interface

# MCX Market Data Interface (MDI) and

# MCX Enhanced Market Data Interface (EMDI)

**Version 1.6**
**January 05, 2026**

## Copyright

All trademarks that appear in the document have been used for identification purposes only and belong to their respective companies.

## Document details

| Name | Version no. | Description |
|---|---|---|
| MCX_FastFIX_MDI_EMDI_API | V 1.6 | API documentation for Trading Market Data interface |

## Document Revision List

| Revision No. | Revision Date | Revision Description |
|---|---|---|
| 1.1 | 28-Apr-2022 | Creation of Version 1.1 |
| 1.2 | 09-Jun-2022 | **Index Stats message –** following fields added<br>LifeHigh, LifeLow, 52WeekHigh, 52WeekLow, closeIndexFlag<br>**Depth snapshot message –** MDSshGrp Sequence modified<br>**Exchange message** removed<br>OpenInterestLastUpdtime field removed in **Depth Snapshot & Incremental message** |
| 1.3 | 17-Aug-2022 | **Depth Incremental message -** Total Buy Quantity and Total sell Quantity fields are added in <MDIncGrp><br>**Depth Snapshot message -** Total Buy Quantity and Total sell Quantity fields are modified with 134-Bid Size and 135- Offer Size |
| 1.4 | 06-Feb-2023 | **Overlay** – Added new MDUpdateAction<br>**Depth Incremental & Snapshot message -** Modified description of TotalTradedValue Field |
| 1.5 | 24-Mar-2023 | **Index Stats message -** Tag No. 40010 changed to 400010 |
| 1.6 | 05-Jan-2026 | **Depth Incremental Message -** section 6.3**.**<br>Addition of values **7 (Upper Circuit Limit)** and **8 (Lower Circuit Limit)** in field MDEntryType. |

## MCX Contact Details

| The Exchange's Member & Vendor may contact Technology Division to seek clarification at: | |
|---|---|
| Multi Commodity Exchange of India Limited<br>Exchange Square, Suren Road, Chakala, Andheri (East), Mumbai 400 093.<br>www.mcxindia.com | Tel: +91 – 22 – 66494000 / 67318888<br>Fax: +91 – 22 – 66494151<br>Email – ctcl@mcxindia.com |

## Restriction on Use and Disclaimer of Information and Data

All the information contained in this document constitutes a trade secret and/or information that are commercial or financial and confidential or privileged. It is furnished in confidence with the understanding that it will not, without the prior written permission of MCX, be used or disclosed for other than allowed purposes.

The copyright in this work may be vested with MCX and / or its suppliers. No part of this document may be copied, reproduced, stored in a retrieval system, or transmitted, in any form or by any means whether, electronic, mechanical, or otherwise without the prior written permission of MCX.

The recipient acknowledges that MCX and its suppliers may have copyright in the work. The recipient further agrees that the work is confidential information and contains proprietary MCX information belonging to MCX and / or its suppliers. The recipient manifests, by its receipt of the work, its acknowledgment of MCX and / or its suppliers copyright in the work, its acceptance that the work is confidential information, and its compliance with the terms contained in this notice.

Although MCX has made every effort to provide accurate information at the date of publication, it does not give any representations or warranties as to the accuracy, reliability or completeness of the information in this document. Accordingly, MCX, its subsidiaries and their employees, officers and contractors and its suppliers shall not, to the extent permitted by law, be liable for any direct or indirect loss arising in any way (including by way of negligence) from or in connection with anything provided in or omitted from this document or from any action taken, or inaction, in reliance on this document.

MCX reserves the right to amend details in this document at any time and without notice. Strictly for private circulation only. This document must not be circulated to other users without prior permission of MCX

# Contents

1 INTRODUCTION ..... 8
1.1 MAIN AUDIENCE ..... 8
1.2 DATA FEEDS ..... 8
1.2.1 MARKET DATA INTERFACES ..... 8
1.3 FURTHER READING MATTER FOR THIS TOPIC ..... 8
1.4 DIFFERENCES BETWEEN THE INTERFACES ..... 9
1.5 CHOOSING BETWEEN THE T7 EMDI AND THE T7 MDI ..... 10
1.6 TRADING STATES ..... 11
1.6.1 PRODUCT STATE CHANGES ..... 11
1.6.2 INSTRUMENT STATE CHANGES ..... 12
1.7 OVERVIEW OF THE VARIOUS MESSAGE TYPES ..... 12
1.7.1 T7 EMDI / MDI ..... 12
1.8 FIX OVER FAST ..... 13
2 FIX/FAST-IMPLEMENTATION ..... 13
2.1 STRUCTURE OF MESSAGES ..... 13
2.2 FAST TERMINOLOGY ..... 14
2.2.1 FAST RESET MESSAGE ..... 14
2.2.2 PRESENCE MAP (PMAP) ..... 14
2.2.3 TEMPLATE ID (TID) ..... 14
2.2.4 DICTIONARIES ..... 15
2.2.5 STOP BIT ENCODING ..... 15
2.2.6 FAST OPERATORS ..... 15
2.3 DECODING THE FAST-MESSAGE ..... 16
2.4 TRANSFER DECODING ..... 16
2.5 COMPOSING THE ACTUAL FIX-MESSAGE ..... 16
2.6 NEW FEATURES IN FAST VERSION 1.2 ..... 16
2.7 DATA TYPES ..... 17
2.8 FAST VERSION 1.1 COMPATIBLE TEMPLATES ..... 17
3 DESCRIPTION OF A TYPICAL TRADING DAY ..... 18
3.1 START OF DAY OPERATION ..... 19
3.2 BUILD THE INITIAL ORDER BOOK ..... 19
3.2.1 BUILD THE INITIAL ORDER BOOK WITH THE T7 EMDI ..... 19
3.2.2 BUILD THE INITIAL ORDER BOOK WITH THE T7 MDI ..... 20
3.3 UPDATE THE ORDER BOOK ..... 20
3.3.1 UPDATE THE ORDER BOOK WITH THE T7 EMDI ..... 20
3.3.2 UPDATE THE ORDER BOOK WITH THE T7 MDI ..... 20
4 GENERAL ORDER BOOK RULES AND MECHANICS ..... 22
4.1 GENERAL ORDER BOOK RULES AND MECHANICS ..... 22
4.1.1 NEW PRICE LEVEL ..... 23
4.1.2 CHANGE OF A PRICE LEVEL ..... 24
4.1.3 OVERLAY ..... 25
4.1.4 DELETION OF A PRICE LEVEL ..... 26

4.1.5 Deletion of multiple price levels from a given price level onwards ..... 26
4.1.6 Deletion of multiple price levels up to a given price level ..... 27
4.2 Trade Volume Reporting (T7 EMDI) ..... 28
4.2.1 Use case 1: Direct match of simple instruments ..... 28
4.2.2 Use case 2: Self-Match prevention (order is totally cancelled) ..... 29
4.2.3 Use case 3: Self-Match prevention (order is partially cancelled) ..... 29
4.3 Trade Volume Reporting (T7 MDI) ..... 29

**5 RECOVERY ..... 29**

5.1 Detecting duplicates and gaps by means of the packet header ..... 29
5.2 Delayed packets ..... 31
5.3 Missing packets ..... 31
5.3.1 Recovery (T7 EMDI) ..... 32
5.3.2 Recovery (T7 MDI) ..... 34

**6 DETAILED DATA FEED DESCRIPTION AND LAYOUT ..... 35**

6.1 Service messages ..... 35
6.1.1 FAST reset message ..... 35
6.1.2 Packet header (T7 EMDI) ..... 35
6.1.3 Packet header (T7 MDI) ..... 36
6.1.4 Functional beacon message ..... 37
6.1.5 Technical heartbeat message ..... 37
6.2 Market data messages ..... 37
6.2.1Depth snapshot message ..... 37
6.3 Depth incremental message ..... 43
6.4 Product State Change ..... 46
6.5 Mass instrument state change message ..... 48
6.6 Index Stats message ..... 51
6.7 Instrument state change message ..... 52

**7 APPENDIX ..... 54**

7.1 Example for a XML FAST template ..... 54

## List of Abbreviations

| Abbreviation | Description |
|---|---|
| EMDI | Enhanced Market Data Interface |
| MDI | Market Data Interface |
| ETI | Enhanced Transaction Interface |
| FAST FIX | FIX Adapted for Streaming (FAST Protocol) (FAST ProtocolSM). 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. |
| Out-Of-Band | Incremental 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 |
| T7 | T7 trading system developed by Deutsche Börse Group |

# 1 Introduction

MCX offers public market data via MDI and EMDI channel.

The **T7 Enhanced Market Data Interface (T7 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; all on-exchange trades are reported individually.

The **T7 Market Data Interface (T7 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. On-exchange trades are not reported individually, however statistical information (daily high/low price, last trade price and quantity) is provided instead.

The T7 MDI provide the following information to the participants:

- Price level aggregated order book depth and trade statistics.
- Product and instrument states.

T7 MDI /EMDI publish market data information following FIX 5.0 SP2 semantics and are FAST 1.2 encoded. The scope of this manual is T7 EMDI, T7 MDI.

## 1.1 Main audience

The target audience of this interface specification is experienced software developer support staff that may be involved in development/support activities for the *T7 Market Data Interfaces*. Prior knowledge of developing for cash or derivative markets 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.

## 1.2 Data feeds

All interfaces deliver public reference and market data in the form of snapshots and incremental. The two public market data interfaces, the **T7 EMDI** for a high bandwidth network and the **T7 MDI** for a low bandwidth network, disseminate information across the T7 network to the receiving application.

### 1.2.1 Market data interfaces

The T7 EMDI and the T7 MDI disseminate public market data information in the form of incremental (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.

## 1.3 Further reading matter for this topic

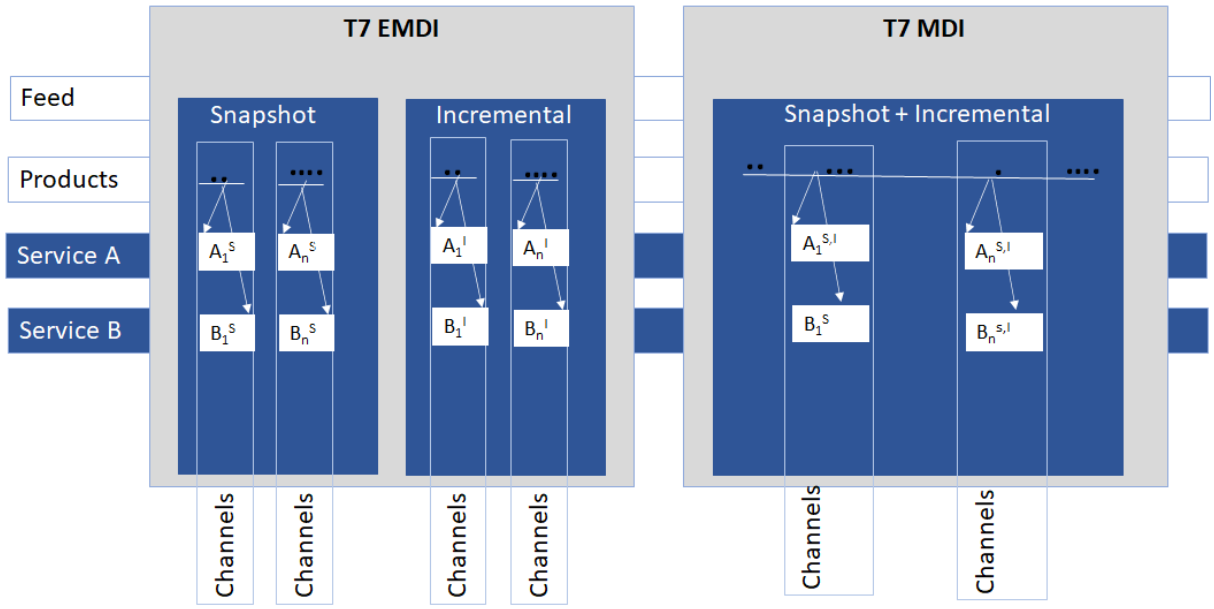
This document is designed as an independent learning and reference manual. However, for background information related to network connectivity, FAST/FIX messages or trading related information (functional), further documents are recommended.
The documents listed below provide useful information.

**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 specification plus the extension Proposal. FIX Adapted for Streaming (FAST)
- **FIX specification documents**: FIX-messages and FIX-tags FIX Standards
- **FIX-Tags**: Specifies all FIX-Tags FIXimate

### 1.4 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 T7 EMDI. The T7 EMDI has multiple channels that have either snapshot ($A_1^S$) to ($A_n^S$) and multiple incremental channels ($A_1^I$) to ($A_n^I$). The T7 MDI has the snapshots and incrementals combined over multiple channels ($A_1^{S,I}$) to ($A_n^{S,I}$).




Products ... ..

IP multicast + port-combination for a particular product n
On the snapshot or incremental feed: S,I

$A_n^{S,I}$

The snapshot and incremental messages for the **T7 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. 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 T7 EMDI, the snapshot and incremental messages for the **T7 MDI** are sent on one feed only (in-band), therefore there is no need to synchronize both messages. The feed is also divided 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 T7 backend side. Service A and Service B cannot be published at exactly the same time.

### 1.5 Choosing between the T7 EMDI and the T7 MDI

Both types of interfaces, 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 **T7 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 **T7 Market Data Interface (netted)** has a *lower bandwidth* requirement compared to the unnetted 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. The order book depth may be lower than for EMDI.

This interface aggregates the order book changes over a specified time interval, which is published in the *Product Snapshot* message via field *MarketDepthTimeInterval (2563)*. The intervals of the incremental messages are often higher than MarketDepthTimeInterval. This is because a smart bandwidth management logic considers the actual overall bandwidth consumption. This interface has less price levels than the T7 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 T7 EMDI and the T7 MDI:

| Area | T7 EMDI | T7 MDI |
|---|---|---|
| In-band/Out-ofband 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. | Messages on the combined market data incrementals + snapshot feed have one sequence number range per product. |

| Area | T7 EMDI | T7 MDI |
|---|---|---|
| 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. |
| 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 1:** *Main differences between the T7 EMDI and the T7 MDI*

Both interfaces, un-netted and netted, provide different recovery time intervals to offer the participants the opportunity to implement a suitable public market data recovery mechanism. Overview of the T7 Public Interfaces

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

### 1.6 Trading states

State changes are disseminated over both the T7 EMDI and the T7 MDI market data feeds.
The T7 EMDI and the T7 MDI market data feeds follow the FIX protocol for the publication of trading state information. The T7 product and instrument states are displayed by these interfaces as shown in the following tables.
Below sections shows, Trading states for a sample business day for derivates illustrates state messages for a typical business day.

#### 1.6.1 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*** | | | |
|---|---|---|---|
| **T7 Product State** | **FIX TradingSessionID (336)** | **FIX TradingSessionSubID (625)** | **FIX TradeSesStatus (340)** |
| Start of Day | 3 = Morning | 7 = Quiescent | 3 = Closed |
| Pre-Trading (special Preopen) | 3 = Morning | 1 = Pre-Trading | 2 = Restricted |
| Trading | 1 = Day | 3 = Continuous | 2 = Open |
| Closing (Not used) | 1 = Day | 4 = Closing | 2 = Open |
| Post-Trading | 5 = Evening | 5 = Post-Trading | 2 = Closed |
| End of Day | 5 = Evening | 7 = Quiescent | 3 = Closed |
| Halt | 1 = Day | 7 = Quiescent | 1 = Halted |

**Table 2:** *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 (2447)* which can take the values 0 = No or 1 = Yes.

### 1.6.2 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[1] 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.
- The status of the instrument (as opposed to the instrument state) distinguishes active and published instruments and is contained in the field *SecurityStatus* (965).

| ***(mass-) instrument state change message*** | |
|---|---|
| **T7 Instrument State** | **FIX *SecurityTradingStatus (326)* /**<br>**FIX *SecurityMassTradingStatus (1679)***<br>2 = Trading Halt. |
| Trading Halt | 2 = Trading Halt |
| Closed | 200 = Closed |
| Restricted | 201 = Restricted |
| Continuous | 203 = Continuous |

Table 3: Instrument *states*

## 1.7 Overview of the various message types

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

### 1.7.1 T7 EMDI / MDI

**Service messages:**

- *Technical heartbeat message* is sent out on all multicast addresses of the T7 EMDI/MDI.
- *Functional beacon message* (T7 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 T7 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 T7 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.

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

### 1.8 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.

After the decoding process, the actual FIX message can be built by applying the FIX structure to the decoded message.
Participants need a standard FAST template based decoder in order to be able to use the T7 EMDI and T7 MDI. Alternatively, participants can use their own FAST decoder implementation.

## 2 FIX/FAST-Implementation

This chapter describes the message structure for the three interfaces. It also provides the basic FASTrules 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), FAST version 1.2 Extension Proposal FIX Adapted for Streaming (FAST)

### 2.1 Structure of Messages

The two 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:

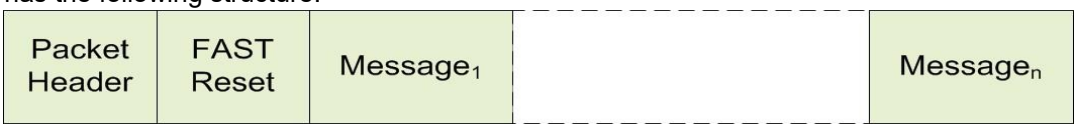



**Figure 1:** *Structure of a UDP datagram*

- The UDP datagram starts with the packet header message.
- 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:

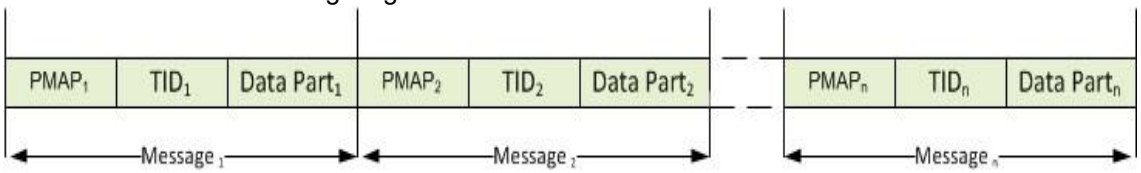



**Figure 2:** *Structure of consecutive messages within one datagram*

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.

## 2.2 FAST terminology

### 2.2.1 FAST reset message

The T7 Market Data Interfaces use **global dictionary** scope for FAST operators. 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.

### 2.2.2 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.

### 2.2.3 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.

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 T7 interfaces:

| Message | TID T7 EMDI | TID T7 MDI |
|---|---|---|
| Functional Beacon (aka Functional Heartbeat) | 109 | - |
| Packet header for T7 EMDI / MDI | 60 | 65 |
| FAST Reset Message | 120 | 120 |
| DepthSnapshot | 93 | 101 |
| DepthIncremental | 94 | 102 |
| ProductStateChange | 97 | 108 |
| MassInstrumentStateChange | 99 | 104 |
| InstrumentStateChange | 98 | 103 |
| Index Stats | 50 | 51 |

**Table 4:** *Template identifiers for T7 EMDI/MDI*

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

**Example:** The TID=65 indicates the *packet header* for T7 MDI in the current release. In the next release the TID for the *packet header* will change to another value.

### 2.2.4 Dictionaries

A dictionary is a cache in which previous values are stored. FAST operators make use of the previous values.

### 2.2.5 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 T7 EMDI/MDI.

### 2.2.6 FAST operators

Field operators are used to remove redundancies in the data values. Message templates are the metadata 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 T7 EMDI/MDI

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

For more information on the new FAST 1.2 features please refer to: FAST Extension Version 1.2.

### 2.3 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).

### 2.4 Transfer decoding

Transfer decoding describes the process of how the fields are decoded from the FAST format. Transfer encoding describes the opposite process.

### 2.5 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.

### 2.6 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 T7 market data interface. A time stamp is an integer that represents a number of time units since an epoch.

### 2.7 Data types

The T7 implementation of FAST utilizes the following FAST data types:

- Decimal
- Length
- String
- uInt32/uInt64/int64
- Byte vector
- Set
- Enum
- Timestamp

### 2.8 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 T7 trading architecture. 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 with T7 Market Data Interfaces - XML FAST Templates.
- **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 with T7 Market Data Interfaces - XML FAST Templates.

.The FAST version 1.2 Extension Proposal 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="6" id="AfterHours"/>
    <element name="7" id="Holiday"/>
    <copy/>
  </enum>
</define>
```

The wire format of the values 1, 3, 5, 6, 7 is 0, 1, 2, 3, 4, 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 4 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"/>
    <element name="BD" id="PreviousClosingPrice"/>
    <element name="a" id="VolumeOnly"/>
    <element name="BB" id="MidpointPrice"/>
    <element name="BC" id="TradingOnTermsOfIssue"/>
    <element name="SA" id="SpecialAuction"/>
    <element name="TC" id="TradeAtClose"/>
  </set>
</define>
```

The wire format of the values U, R, AX, AY, AJ, AW, k, BD, a is 1, 2, 4, 8, 16, 32, 64, 128, 256, 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”

## 3 Description of a typical trading day

This chapter describes a typical trading day, from the start until the end of trading; the following steps need to be taken to prepare for and to receive market data:

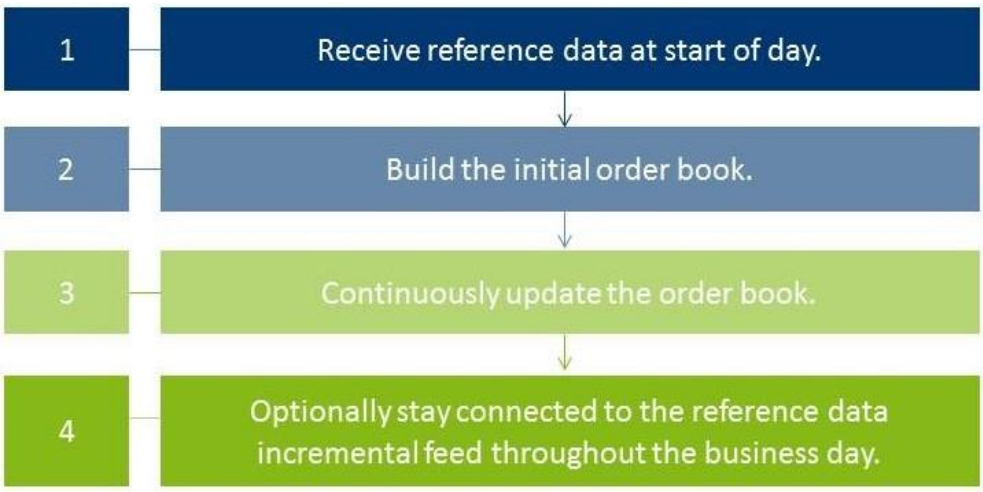



**Figure 3:** *Typical trading day*

### 3.1 Start of day operation

Before processing any market data, receiving applications need to retrieve the reference data. Members are advised to verify that the received reference data refers to the correct business day to ensure that the reference data processed on their end is actually the reference data for the business day in question and not reference data from e.g. the previous business day.

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

### 3.2 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.

#### 3.2.1 Build the initial order book with the T7 EMDI

For each instrument within the desired products do the following:




**Figure 4:** *T7 EMDI initial order book*

### 3.2.2 Build the initial order book with the T7 MDI

The following sequence is recommended for the **T7 MDI:**




**Figure 5:** *T7 MDI initial order book*

The field *LastMsgSeqNumProcessed (369)* in the T7 MDI snapshots can be ignored because snapshots and incrementals are sent in-band and don't need to be synchronized with each other.

**MDI** does not send all snapshots of all instruments of a product contiguously. T7 MDI incremental messages might contain incremental entries (MDIncGrp) for all instruments of a product. A joining application which is in the middle of building the initial order books *must* discard entries for instruments for which they have not received a snapshot yet.

## 3.3 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:

### 3.3.1 Update the order book with the T7 EMDI

As long as the *MsgSegNum* values for the *depth incremental* message are contiguous per product do the following [2]:

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

**Note**: *Depth snapshot* messages are sent on a different channel to 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.

### 3.3.2 Update the order book with the T7 MDI

As long 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* messages (with *RefreshIndicator (1187)* = *Y*) to the current order book.

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

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 T7 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.

# 4 General Order Book rules and mechanics

## 4.1 General order book rules and mechanics

The T7 Market Data Interfaces, T7 EMDI and MDI, provide order book updates from level 1 to the maximum level. The maximum level is provided for each product in the product snapshot records in the reference data, field *MarketDepth (264)*. 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 T7 EMDI.
- Each T7 EMDI packet relates only to a single product. In other words, although each T7 EMDI packet may contain multiple messages, those messages will always relate to the same product. This does not apply to T7 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“or when no price levels exist, an empty book (MDEntryType=J) is disseminated for the depth snapshot message (not for incremental). In “Pre-Trading “, statistical information is sent in addition to order book if there are any GTC/GTD orders.
- After Post-Trading, further market data updates are not disseminated.
- 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, after each applied individual orderbook update instruction, 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 T7 MDI carry a common and contiguous sequence number per product. The incremental message of T7 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.

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

Figure 6 illustrates a typical order book and terminology used in the following chapters.




**Figure 6:** *Typical order book*

### 4.1.1 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. [3].
- All existing rows in the order book at the specified and higher levels are to be incremented accordingly. [4].
- 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 |

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

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

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 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/quotes on this level |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | t0 | official time of book entry |

**Table 5:** *MDUpdateAction "New"*

### 4.1.2 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 ≥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 |

| Tag number | Tag name | Value | Description |
|---|---|---|---|
| 270 | > MDEntryPx | **58.22** | Price |
| 271 | > MDEntrySize | **8** | Quantity |
| 346 | > NumberOfOrders | 1 | Number of order/quotes on this level |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | t1 | official time of book entry |

**Table 6:** *MDUpdateAction "Change"*

### 4.1.3 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 ≥ 1, i.e. the field MDPriceLevel (1023) must be present. In contrast to the MDUpdateAction="Change" this instruction contains a price change.

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 | t5 | official time of book entry |

**Table 7:** *MDUpdateAction "Overlay"*

### 4.1.4 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. Price and quantity 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 | **8** | Quantity |
| 1023 | > MDPriceLevel | 1 | Book level |
| 273 | > MDEntryTime | t2 | official time of book entry |

**Table 8:** *MDUpdateAction "Delete"*

### 4.1.5 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 ≥ 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 | **13** | Quantity |
| 1023 | > MDPriceLevel | **3** | Book level |
| 273 | > MDEntryTime | t3 | official time of book entry |

**Table 9:** *MDUpdateAction "Delete From"*

### 4.1.6 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<br>48 | > MDEntryType<br>> SecurityID | 0<br>8852 | Bid<br>Unique identifier assigned to each instrument |
| 22 | > SecurityIDSource | M | Marketplace-assigned identifier |
| 270 | > MDEntryPx | **58.22** | Price on level 3 |
| 271<br>346 | > MDEntrySize<br>> NumberOfOrders | 10<br>1 | Quantity<br>Number of order/quotes on this level |
| 1023<br>273 | > MDPriceLevel<br>> MDEntryTime | **3**<br>t4 | Book level<br>official time of book entry |

**Table 10:** *MDUpdateAction "Delete Thru"*

### 4.2 Trade Volume Reporting (T7 EMDI)

All on-exchange trades executed on T7 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 T7 EMDI disseminates information about on book trades and set MDOriginType(1024) is set to Book.
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 and origin type for each business day.

#### 4.2.1 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, 10@85,**
**BMW Existing Order book:**

| Bid | Ask |
|---|---|
| | 5@84.9 |
| | 5@85 |

**Trade Volume Reporting:** Two trades are reported because two different price levels are involved in the matching process: First 5@84.9 gets reported due to a higher matching priority of this price level; afterwards 5@85.

| Instr. | MDEntryID | MDUpdateAction | size@prc | TradeCond. | AggrSide | #Buy | #Sell |
|---|---|---|---|---|---|---|---|
| Inst1 | 1 | NEW | 5@84.9 | U,R,AX,AY | BUY | 1 | 1 |
| Inst1 | 2 | NEW | 5@85 | U,AX | BUY | 1 | 1 |

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

### 4.2.2 Use case 2: Self-Match prevention (order is totally cancelled)

An incoming order is cancelled due to Self-Match prevention.
Incoming **buy order, 150@84, Inst1 Mar, PANID1=, Member A Existing Order book:**

| Bid | Ask |
|---|---|
| | 50@84 (PANID1=, Member A) |

**Trade Volume Reporting:** A trade is reported: 0@84, AggressorSide BUY. MDEntryID, TradeCondition, number of Buy and number of Sell orders are not filled. The resting cancelled quantity is 50. The incoming cancelled quantity (150) is not reported.

| Instr. | MDEntryID | MDUpdateAction | size@prc | TradeCond. | AggrSide | #Buy | #Sell | #RestingCxlQty |
|---|---|---|---|---|---|---|---|---|
| Inst1 | | NEW | 0@84 | | BUY | | | 50 |

### 4.2.3 Use case 3: Self-Match prevention (order is partially cancelled)

An incoming order is partially cancelled due to Self-Match prevention. **Incoming buy order, 150@84, Inst1, PANID1, Member A Existing Order book:**

| Bid | Ask |
|---|---|
| | 20@84 (MatchInstCrossID=1, Member A) |
| | 30@84 (MatchInstCrossID=0) |

**Trade Volume Reporting:** A trade is reported: 30@84, AggressorSide BUY. The resting cancelled quantity is 20.The incoming cancelled quantity (120) is not reported.

| Instr. | MDEntryID | MDUpdateAction | size@prc | TradeCond. | AggrSide | #Buy | #Sell | #RestingCxlQty |
|---|---|---|---|---|---|---|---|---|
| Inst1 | 1 | NEW | 30@84 | U | BUY | 1 | 1 | 20 |

### 4.3 Trade Volume Reporting (T7 MDI)

The T7 MDI only provides statistical data (daily high/low price as well as total trade volume) for trades as well as the last traded price and quantity. For each simple instrument participating in a trade, T7 MDI reports the total traded volume even when there are no simple instrument orders involved in the trade.

## 5 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 even 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.

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

The 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 FGOLD and FSILVER, but the participant is only interested in FGOLD.

I. **Pessimistic approach:** The receiving application assumes that FGOLD is part of the missing packet: It immediately starts recovery actions 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 FGOLD is not part of the message which is inside the lost packet.

II. **Optimistic approach:** The receiving application assumes that FGOLD 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 FGOLD 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.

### 5.2 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 11:** *Packets arriving in correct sequence*

In this example, messages arrive in the correct order. The message was not delayed between T7 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, T7 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 12:** *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.

### 5.3 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 | |

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

**Table 11:** *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: then initiate recovery via snapshots.

### 5.3.1 Recovery (T7 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.

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 (timedriven).

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 | LastMsgSeq-NumProcessed | Message Type | Channel |
|---|---|---|---|---|
| 205 | A | | quote request | A1I |
| 206 | A | | depth incremental | A1I |
| 207 | A | | depth incremental | A1I |
| lost | A | | depth incremental | A1I |
| lost | A | | depth incremental | A1I |
| 210 | A | | depth incremental | A1I |

| MsgSeqNum | Product | LastMsgSeq-NumProcessed | Message Type | Channel |
|---|---|---|---|---|
| 1000 | B | | depth incremental | A2I |
| N/A | A | 209 | depth snapshot | $A_1S$ |
| 211 | A | | depth incremental | A1I |
| N/A | B | 1000 | depth snapshot | $A_2S$ |
| 1001 | B | | depth incremental | A2I |

**Table 12:** *Snapshots and incrementals within the T7 EMDI*

The appropriate recovery action for missing *depth incrementals* is the same as the logic described in section Build the initial order book with the T7 EMDI.
There are some additional points to be aware of when performing recovery:

- During recovery, applications should be prepared to receive *depth incremental* messages for instruments they didn't know existed. This can occur if a strategy creation event (via a *complex instrument update* on the market data feed) is missed due to packet loss. In this case, applications must consult the reference data snapshot feed to obtain the strategy description.
- *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:

#### *5.3.1.1 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:

P1I1 | P1I2 | P1I3 | P1In | P2I1 | P2I2 | P2I3 | P2In | P3I1 | P3I2 | P3I3 |
P3Iq | [...] **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.

#### *5.3.1.2 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.

### 5.3.2 Recovery (T7 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 | quote request | $A_1$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 13:** *Snapshots and incrementals within the T7 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* ≥11 can be applied to the latest snapshot with *MsgSeqNum* = 10.

**Note:** *LastMsgSeqNumProcessed* is not necessary for recovery purposes in the T7 MDI.

# 6 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.
Please consider, that the following tables will only list valid values for enum and set data types, which are used within that specific context.

## 6.1 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.

### 6.1.1 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)

**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.

### 6.1.2 Packet header (T7 EMDI)

#### *6.1.2.1 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.

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. |

| Field Name | Data Type | Description |
|---|---|---|
| PacketSeqNumber | byte vector | Datagram sequence number. |
| SendingTime | byte vector | Time when market data feed handler writes packet on the wire. |
| PerformanceIndicator | byte vector | Not used |

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 7:** *Structure of the packet header for T7 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 T7 EMDI incremental feed. The *PartitionID* is available in messages on both incremental and snapshot feed of the T7 EMDI.

### 6.1.3 Packet header (T7 MDI)

***Delivered in every UDP-datagram***

The *packet header* of T7 MDI contain following fields.

| 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:

| 1 Byte | 1 Byte | 1 Byte | 1 Byte | 4 Bytes | 1 Byte | 8 bytes |
|---|---|---|---|---|---|---|
| PMAP | TID | Sender Comp ID | Length | PacketSeqNum | Length | SendingTime |
| 1 | 2 | 3 | 4 | 8 | 9 | 17 |

**Figure 8:** *Structure of the packet header for T7 MDII*

### 6.1.4 Functional beacon message

#### *6.1.4.1 Delivered on: T7 EMDI incremental*

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.

| 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, for EMDI or Market Identifier |
| 369 | LastMsgSeqNum-Processed | Y | uInt32 | Last sequence number on the incremental feed for this SenderSubID. |

### 6.1.5 Technical heartbeat message

Delivered on: every channel for T7 EMDI, T7 MDI

The technical heartbeat (also called technical beacon) 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.

## 6.2 Market data messages

The market data feeds disperse public market data via the T7 EMDI and the T7 MDI.
Public market data for all instruments are 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 are different for the T7 EMDI and the T7 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 T7 EMDI and T7 MDI is the same.

### 6.2.1Depth snapshot message

Delivered on: T7 EMDI snapshot feed, T7 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 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 - General order book rules and mechanics

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | <table><tr><th>Value</th><th>Description</th></tr><tr><td>W</td><td>Market Data Snapshot Full Refresh</td></tr></table> |
| 34 | MsgSeqNum | N | uInt32 | Not used by unnetted feed (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.<table><tr><th>Value</th><th>Description</th></tr><tr><td>Y</td><td>Mandatory Refresh</td></tr><tr><td>N</td><td>Optional Refresh</td></tr></table> |
| 1300 | MarketSegmentID | Y | uInt32 | Product identifier, e.g. "89". |
| 48 | SecurityID | Y | int64 | Instrument identifier, e.g. "8852". |
| 22 | SecurityIDSource | Y | String | Source Identification.<table><tr><th>Value</th><th>Description</th></tr><tr><td>M</td><td>Marketplace-assigned Identifier</td></tr></table> |
| 1227 | ProductComplex | Y | ProductComplex (enum) | Type of Instrument<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Simple Instrument</td></tr><tr><td>5</td><td>Futures Spread</td></tr></table> |
| 965 | SecurityStatus | Y | Security-Status (enum) | Status of the instrument.<br>**2 = Inactive** will be set for pending deletions of complex instruments.<br>**4 = Expired** will be set for instruments that have expired intraday.<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Active</td></tr></table> |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| | | | | 2 Inactive<br>4 Expired<br>9 Suspended |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 25045 | TESSecurityStatus | N | Security-Status (enum) | Not used. |
| 779 | LastUpdateTime | Y | timestamp | 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. |
| 134 | TotalBuyQuantity | N | decimal | Total Buy Quantity |
| 135 | TotalSellQuantity | N | decimal | Total Sell Quantity |
| <MDSshGrp> sequence starts | | | | |
| 1024 | > MDOriginType | Y | MDOrigin-Type (enum) | **= Book** is for on-exchange trading. |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>269</td><td>> MDEntryType</td><td>Y</td><td>MDEntry-Type (enum)</td><td>J = Empty Book is sent during product states “Start-Of-Day” or when no price levels exist. During “PostTrading“ and “End-Of-Day“ ToB prices are distributed.<br>B = Trade Volume The total traded volume of units traded during the day can be found in the MDEntrySize field. Please note that the total traded volume may include coherent volume (from direct matching of complex instruments) as well.<br>S = Spot price, MDEntryPX and MDEntryTime are filled
<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>7</td><td>Upper Circuit Limit</td></tr>
<tr><td>8</td><td>Lower Circuit Limit</td></tr>
<tr><td>J</td><td>Empty Book</td></tr>
<tr><td>Q</td><td>Auction Clearing Price (NotUsed)</td></tr>
<tr><td>B</td><td>Trade Volume</td></tr>
<tr><td>A</td><td>Imbalance (Not used)</td></tr>
<tr><td>C</td><td>Open Interest</td></tr>
<tr><td>S</td><td>Spot Price</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>1021</td><td>> MDBookType</td><td>N</td><td>MDBook-Type (enum)</td><td>Price depth information
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>2</td><td>Price Depth</td></tr>
</table></td></tr>
<tr><td>1173</td><td>> MDSubBookType</td><td>N</td><td>MDSubBook-Type (enum)</td><td>Not used</td></tr>
<tr><td>828</td><td>> TrdType</td><td>N</td><td>TrdType [5](enum)</td><td>Not used</td></tr>
<tr><td>336</td><td>> TradingSessionID</td><td>N</td><td>TradingSessionID (enum)</td><td>Always attached to the first MDEntry.
<table>
<tr><th>Value</th><th>Description</th></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></td><td></td><td></td><td></td><td><table><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>6</td><td>After-Hours</td></tr><tr><td>7</td><td>Holiday</td></tr></table></td></tr>
<tr><td>625</td><td>> TradingSessionSubID</td><td>N</td><td>TradingSession-SubID (enum)</td><td>See description for TradingSessionID (336).<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>Pre-Trading</td></tr><tr><td>3</td><td>Continuous</td></tr><tr><td>4</td><td>Closing</td></tr><tr><td>5</td><td>Post-Trading</td></tr><tr><td>7</td><td>Quiescent</td></tr></table></td></tr>
<tr><td>25044</td><td>> TESTradSesStatus</td><td>N</td><td>TradSes-Status (enum)</td><td>Not used</td></tr>
<tr><td>326</td><td>> SecurityTradingStatus</td><td>N</td><td>Security-Trading-Status (enum)</td><td>See description for TradingSessionID (336).<br>Trading status of an instrument.<br>2 = Trading Halt,<table><tr><th>Value</th><th>Description</th></tr><tr><td>2</td><td>Trading Halt</td></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></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>2705</td><td>> MarketCondition</td><td>N</td><td>MarketCondition (enum)</td><td>Indicator for stressed market conditions.<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Normal</td></tr></table></td></tr>
<tr><td>2447</td><td>> FastMarketIndicator</td><td>N</td><td>FastMarketIndicator (enum)</td><td>Not used</td></tr>
</table>

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 1174 | > SecurityTradingEvent | N | Security-Trading-Event (enum) | Not used |
| 28872 | > PotentialSecurity-TradingEvent | N | Security-Trading-Event(enum) | Not used |
| 25155 | > SoldOutIndicator | N | Sold-Out-Indicator (enum) | Not used |
| 277 | > TradeCondition | N | Trade-Condition (set) | **Value** / **Description**<br>U – Exchange Last<br>R – Opening Price<br>AX – High Price<br>AY – Low Price<br>AJ – Closing Price<br>BD – Previous Closing Price<br>AU – Life High<br>AV – Life Low |
| 442 | >MultiLegReportingType | N | MultiLeg-Reporting-Type(enum) | Not used |
| 28750 | >MultiLegPriceModel | N | MultiLeg-PriceModel (enum) | Not used |
| 276 | > QuoteCondition | N | QuoteCondition (enum) | Not used |
| 270 | > MDEntryPx | N | decimal | Price. |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 271 | > MDEntrySize | N | decimal | |
| 346 | > NumberOfOrders | N | uInt32 | |
| 1023 | > MDPriceLevel | N | uInt32 | Book level. |
| 273 | > MDEntryTime | N | Timestamp | Time of entry in nanoseconds for last trade entry<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. |
| 28873 | >NonDisclosedTradeVolume | N | Decimal | Not used |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 381 | >TotalTradedValue | N | Decimal | Total Traded Value(In Lacs) |
| 426 | AverageTradedPrice | N | Decimal | Average Traded Price |
| 6139 | > TotalNumberOfTrades | N | uInt32 | Total Number of trades during the day. Only present for MDEntryType = B. Applicable for cash market products only. |
| <MDSshGrp> sequence ends | | | | |

### 6.3 Depth incremental message

#### *6.3.1 Delivered on: T7 EMDI incremental feed, T7 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 – Market Data Incremental 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 | |
| 1024 | > MDOriginType | Y | MDOrigin-Type (enum) | **0 = Book** is for on-exchange trading. |
| 279 | > MDUpdateAction | Y | MDUpdate-Action (enum) | **Value** / **Description**: 0 – New; 1 – Change; 2 – Delete; 3 – Delete Thru; 4 – Delete From; 5 – Overlay |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>269</td><td>> MDEntryType</td><td>Y</td><td>MDEntry-Type (enum)</td><td>See Depth snapshot message.<br>B = Trade Volume Trade volume entry for MDI, to provide new total trade volume from the last netting interval. Also used in EMDI for recovery purposes after a failover on the exchange side. In this case, the total traded volume of units traded during the day can be found in the MDEntrySize field.<br>S = In case of Spot price, MDEntryPx and MDEntry Time
<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>7</td><td>Upper Circuit Limit</td></tr>
<tr><td>8</td><td>Lower Circuit Limit</td></tr>
<tr><td>Q</td><td>Auction Clearing Price (Not used)</td></tr>
<tr><td>B</td><td>Trade Volume</td></tr>
<tr><td>A</td><td>Imbalance (Not used)</td></tr>
<tr><td>C</td><td>Open Interest</td></tr>
<tr><td>S</td><td>Spot Price</td></tr>
</table></td></tr>
<tr><td>48</td><td>> SecurityID</td><td>Y</td><td>int64</td><td>Instrument identifier, e.g. "8852".</td></tr>
<tr><td>22</td><td>> SecurityIDSource</td><td>Y</td><td>string</td><td>Source Identification.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>M</td><td>Marketplace-assigned Identifier</td></tr>
</table></td></tr>
<tr><td>270</td><td>> MDEntryPx</td><td>N</td><td>decimal</td><td>Price of market data (trade or order).</td></tr>
<tr><td>271</td><td>> MDEntrySize</td><td>N</td><td>decimal</td><td>Quantity or trade volume when MDEntryType = 2 or "B".</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.</td></tr>
<tr><td>273</td><td>> MDEntryTime</td><td>N</td><td>timestamp</td><td>For bids and offers the official time of book entry, for trades official time of execution (all in nanoseconds).</td></tr>
</table>

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 28872 | > PotentialSecurity-TradingEvent | N | Security-Trading-Event (enum) | Not used |
| 276 | > QuoteCondition | N | QuoteCondition (enum) | Not used |
| 134 | > TotalBuyQuantity | N | decimal | Total Buy Quantity |
| 135 | > TotalSellQuantity | N | decimal | Total Sell Quantity |
| <TradeEntryGrp> (optional) group starts | | | | |
| 828 | > TrdType | N | TrdType (enum) | Not used |
| 2667 | > AlgorithmicTrade-Indicator | N | Algorithmic-TradeIndicator (enum) | Not used |
| 277 | > TradeCondition | N | Trade-Condition (set) | Defines the type of price for MDEntryPx. Only present for MDEntryType 2 = Trade. |
| 442 | > MultiLegReportingType | N | MultiLeg-Reporting-Type (enum) | Not used |
| 28750 | > MultiLegPriceModel | N | MultiLeg-PriceModel (enum) | Not used |
| 2445 | > AggressorTime | N | timestamp | Not used |
| 5979 | > Reserve | N | timestamp | Not used |

| Value | Description |
|---|---|
| U | Exchange Last |
| R | Opening Price |
| AX | High Price |
| AY | Low Price |
| AJ | Official Closing Price |
| BD | Previous Closing Price |
| AU | Life High |
| AV | Life Low |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 2446 | > AggressorSide | N | Aggressor-Side (enum) | Side of the incoming order that triggered the trade. Only present for MDEntryType=2.<br>**Value** / **Description**<br>1 – Buy<br>2 – Sell |
| 2449 | > NumberOfBuyOrders | N | uInt32 | Number of buy orders involved in the trade. Only present for MDEntryType=2 and Trade Condition other than "a" (Volume Only). |
| 2450 | > NumberOfSellOrders | N | uInt32 | Number of sell orders involved in the trade. Only present for MDEntryType=2 and Trade Condition other than "a" (Volume Only). |
| 6139 | > TotalNumberOfTrades | N | uInt32 | Total Number of trades during the day. Only present for MDEntryType=2. Applicable for cash market products only. |
| 28869 | > RestingCxlQty | N | decimal | Quantity that was cancelled due to SMP. Only present for MDEntryType=2. |
| 278 | > MDEntryID | N | uInt32 | Represents the match step ID. This field is unique together with MarketSegmentID. Only present for MDEntryType = 2. |
| 28873 | >NonDisclosedTradeVolume | N | decimal | Not used |
| 381 | >TotalTradedValue | N | Decimal | Total Traded Value(In Lacs) |
| 426 | >AverageTradedPrice | N | Decimal | Average Traded Price |
| <TradeEntryGrp> (optional) group ends | | | | |
| <MDIncGrp> sequence ends | | | | |

### 6.4 Product State Change

#### *6.4.1 Delivered on: T7 EMDI incremental feed, T7 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** |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| | | | | h — Trading Session Status |
| 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 | TradingSessionID (enum) | **Value** / **Description**: 1 Day; 3 Morning; 5 Evening; 6 After-Hours; 7 Holiday |
| 625 | TradingSessionSubID | Y | TradingSession-SubID (enum) | **Value** / **Description**: 1 Pre-Trading; 3 Continuous; 4 Closing; 5 Post-Trading |
| 340 | TradSesStatus | Y | TradSes-Status (enum) | **Value** / **Description**: 1 Halted; 2 Open; 3 Closed |
| 2705 | MarketCondition | N | MarketCondition (enum) | **Value** / **Description**: 0 Normal |
| 2447 | FastMarketIndicator | Y | FastMarketIndicator (enum) | Not used |
| 60 | TransactTime | Y | timestamp | Time in nano seconds,indicates the time when the message was sent. |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 25044 | TESTradSesStatus | N | TradSes-Status (enum) | Not used |

### 6.5 Mass instrument state change message

***Delivered on: T7 EMDI incremental feed, T7 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.

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 – Security Mass Status |
| 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-ScopeProductComplex (enum) | Instrument type of affected instruments.<br>**Value** / **Description**: 1 – Simple Instrument; 5 – Futures Spread |
| 30965 | SecurityMassStatus | Y | Security-Status (enum) | The instrument status of all affected instruments.<br>**Value** / **Description**: 1 – Active; 2 – Inactive; 4 – Expired; 9 – Suspended |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 1679 | SecurityMassTradingStatus | N | Security-Trading-Status(enum) | See SecurityTradingStatus in *Depth snapshot*<br>**Value** / **Description**<br>2 – Trading Halt<br>200 – Closed<br>201 – Restricted (PreOpen)<br>203 – Continuous |
| 28894 | MassMarketCondition | Y | MarketCondition (enum) | See *Depth snapshot* message<br>**Value** / **Description**<br>0 – Normal |
| 2447 | FastMarketIndicator | Y | FastMarketIndicator (enum) | Not used |
| 1680 | Security-MassTradingEvent | N | Security-Trading-Event (enum) | Not used |
| 35155 | MassSoldOutIndicator | N | Sold-Out-Indicator (enum) | Not used |
| 60 | TransactTime | Y | timestamp | Time when request was processed (nanoseconds). |
| 35045 | TESSecurityMassStatus | N | Security-Status (enum) | Not used |
| <SecMassStatGrp> sequence starts | | | | |
| 146 | NoRelatedSym | N | length | |
| 48 | > SecurityID | Y | int64 | Instrument identifier, e.g. "8852". |
| 22 | > SecurityIDSource | Y | string | **Value** / **Description**<br>M – Marketplace-assigned Identifier |
| 965 | > SecurityStatus | Y | Security-Status (enum) | See *Depth snapshot* message<br>**Value** / **Description**<br>1 – Active<br>2 – Inactive<br>4 – Expired<br>9 – Suspended |

<table>
<tr><th>Tag</th><th>Field Name</th><th>Req'd</th><th>Data Type</th><th>Description</th></tr>
<tr><td>326</td><td>> SecurityTradingStatus</td><td>N</td><td>Security-Trading-Status (enum)</td><td>See Depth snapshot message<br>Empty for flexible instruments.<table><tr><th>Value</th><th>Description</th></tr><tr><td>2</td><td>Trading Halt</td></tr><tr><td>200</td><td>Closed</td></tr><tr><td>201</td><td>Restricted</td></tr><tr><td>203</td><td>Continuous</td></tr></table></td></tr>
<tr><td>2705</td><td>> MarketCondition</td><td>Y</td><td>MarketCondition (enum)</td><td>See Depth snapshot message<table><tr><th>Value</th><th>Description</th></tr><tr><td>0</td><td>Normal</td></tr></table></td></tr>
<tr><td>1174</td><td>> SecurityTradingEvent</td><td>N</td><td>Security-Trading-Event(enum)</td><td>Not used</td></tr>
<tr><td>25155</td><td>> SoldOutIndicator</td><td>N</td><td>Sold-Out-Indicator (enum)</td><td>Not used</td></tr>
<tr><td>25045</td><td>> TESSecurityStatus</td><td>N</td><td>Security-Status (enum)</td><td>Not used</td></tr>
<tr><td colspan="5"><SecMassStatGrp> sequence ends</td></tr>
<tr><td>893</td><td>LastFragment</td><td>Y</td><td>LastFragment (enum)</td><td>Indicates whether this message is the last in a sequence of messages that together convey a joint exception list of SecMassStatGrp. All messages up to the last with LastFragment = Y share the same root level content and an application first needs to combine all single exception lists before the Mass State Change message could be applied with the fully joint exception list<table><tr><th>Value</th><th>Description</th></tr><tr><td>N</td><td>Not Last Message</td></tr><tr><td>Y</td><td>Last Message</td></tr></table></td></tr>
</table>

### 6.6 Index Stats message

Delivered on: T7 EMDI incremental feed, T7 MDI data feed

This message provides information for the current and index related statistics. The message is complete each time it is delivered and thus does not require recovery.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | String | **Value** / **Description**: I – Index broadcast |
| 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 | uIDepth nt32 | Index code |
| 40001 | IndexHigh | Y | Decimal | High value |
| 40002 | IndexLow | Y | Decimal | Low value |
| 40003 | IndexOpen | Y | Decimal | Open Value |
| 40004 | IndexClose | Y | Decimal | Close Value |
| 40005 | IndexValue | Y | Decimal | Current Index Value |
| 40006 | LifeHigh | Y | Decimal | The highest price during the lifetime of a particular contract. |
| 40007 | LifeLow | Y | Decimal | The lowest price during the lifetime of a particular contract. |
| 40008 | 52WeekHigh | Y | Decimal | The highest price over the period of past 52 weeks |
| 40009 | 52WeekLow | Y | Decimal | The lowest price over the period of past 52 weeks |
| 400010 | closeIndexFlag | Y | Decimal | If the closeIndexFlag is true then in CloseIndex field we receive index close value and if it is false then we receive PreviousCloseIndex value |
| 60 | TransactTime | Y | Timestamp | Time when index value is last updated in nanosec |

### 6.7 Instrument state change message

***Delivered on: T7 EMDI incremental feed, T7 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 4 = Expired.

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 35 | MsgType | Y | string | **Value** / **Description**: f – Security Status |
| 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 | int64 | Instrument identifier, e.g. "8852". |
| 22 | SecurityIDSource | Y | string | **Value** / **Description**: M – Marketplace-assigned identifier |
| 965 | SecurityStatus | Y | Security-Status (enum) | See *Depth snapshot* message<br>**Value** / **Description**: 1 – Active; 2 – Inactive; 4 – Expired; 9 – Suspended |
| 326 | SecurityTradingStatus | N | Security-Trading-Status (enum) | See *Depth snapshot* message<br>Empty for flexible instruments.<br>**Value** / **Description**: 2 – Trading Halt; 200 – Closed; 201 – Restricted; 202 – Book; 203 – Continuous |
| 2705 | MarketCondition | Y | MarketCondition (enum) | See *Depth snapshot* message<br>**Value** / **Description**: 0 – Normal |

| Tag | Field Name | Req'd | Data Type | Description |
|---|---|---|---|---|
| 2447 | FastMarketIndicator | Y | FastMarketIndicator (enum) | Not used |
| 1174 | SecurityTradingEvent | N | Security-Trading-Event (enum) | Not used |
| 25155 | SoldOutIndicator | N | Sold OutIndicator (enum) | Not used |
| 60 | TransactTime | Y | timestamp | Time when request was processed (nanoseconds). |
| 25045 | TESSecurityStatus | N | Security-Status (enum) | Not used |

# 7 Appendix

## 7.1 Example for a XML FAST template

This example of Decoding the FAST-message.

```
<template id="103" name="InstrumentStateChange">
    <string name="MsgType" id="35">
            <constant value="f"/>
    </string>
    <uInt32 name="MsgSeqNum" id="34"> <increment/>
    </uInt32>
    <uInt32 name="SenderCompID" id="49"> <copy/>
    </uInt32>
    <uInt32 name="MarketSegmentID" id="1300"> <copy/>
    </uInt32>
    <int64 name="SecurityID" id="48"/>
    <string name="SecurityIDSource" id="22">
      <constant value="M"/>
    </string>
    <field name="SecurityStatus" id="965">
      <type name="SecurityStatus">
        <default value="0"/>
      </type>
    </field>
    <field name="SecurityTradingStatus" id="326" presence="optional">
      <type name="SecurityTradingStatus">
        <copy/>
      </type>
    </field>
    <field name="MarketCondition" id="2705">
      <type name="MarketCondition"> <default value="0"/>
      </type>
    </field>
    <field name="SecurityTradingEvent" id="1174" presence="optional">
      <type name="SecurityTradingEvent"> <copy/>
      </type>
    </field>
    <timestamp name="TransactTime" unit="nanosecond" id="60">
      <delta/>
    </timestamp>
  </template>
```

**Figure 15:** *Example for a FAST template with repeating group*