IT Development Division

Trading Systems Development Department

ATHEX
Athens Stock Exchange

MARKET
DATA FEED

OASIS MDFS Specification

Version: 0.16

Revision History

Version	Date	Description
0.12	2024/03/11	UAT release.
0.13	2024/04/16	Removed sections for unused instructions “Delete Thru”, “Delete From” and “Overlay” from section “Order Book Handling”.
0.14	2024/05/24	1. Updated section “2.1. Multicast Groups & Services” and table “Figure 4 - Instrument Type Groupings”. 2. Updated formatting of section “5. Order Book Handling” for consistency with updated FIX message content.
0.15	2024/07/24	1. Updated section ”3.6. TCP/IP Retransmission Service”. 2. Updated section “4. FAST Message Encoding”. 3. Updated section “5. Order Book Handling”. 4. Updated section “6.1 Comparison With Legacy IDS Service (IOCP)
0.16	2025/01/20	1. Added section “6. Instrument Prices Handling”. 2. Updated section “5.1. Market/Stop/ATO/ATC orders”. 3. Added section “3.1. Handling Incremental & Snapshot Traffic”. 4. Updated section “3.7. TCP/IP Retransmission Service”. 5. Updated section “3.7.1. TCP/IP Retransmission Procedure” to reflect the updated “UEFD = EncapsulatedFASTData” messages.

Table of Contents

Table of Figures

1. Introduction

The ATHEX Market Data Feed Service (**MDFS**) provides real time, trading data feed information for all instruments traded in the OASIS platform.

The MDFS provides data using the Financial Information eXchange (FIX) Protocol which is a technical specification that is owned, maintained, and developed through the collaborative efforts of FIX Trading Community. More specifically the data format follows the FIX 5.0 SP2 specification and the data is encoded according to the FAST 1.2 specification. Some messages, tags and tag values from FIX Extension Packs the FIX 5.0 SP2 specification are utilized in MDFS messages.

The FIX protocol is an industry standard used by institutions, market participants and vendors worldwide. It facilitates the streamlined, open, and adaptable exchange of information between counterparties and is used in multiple aspects of trading, including the dissemination of market data (such as that served by the MDFS).

The FAST encoding method is a binary encoding method for message oriented data streams that aims to be space and processing efficient. It reduces the size of a data stream by removing redundant data and serializing of the remaining data through binary encoding, self-describing field lengths and bit maps indicating the presence or absence of fields. The FAST encoding method is widely used by institutions serving market data to reduce the data stream size and remove any unnecessary overhead, allowing for reduced latency and bandwidth consumption.

The MDFS delivers market data by implementing an incremental / snapshot message approach that is outlined by the FIX Trading Community, using UDP multicast as the network transport protocol. This approach enables a rich and performant market data feed with minimal latency.

A brief comparison with the legacy IDS Service (IOCP) can be found in section 5.1 of the Appendix.

2. Architecture Overview

2.1. Multicast Groups & Services

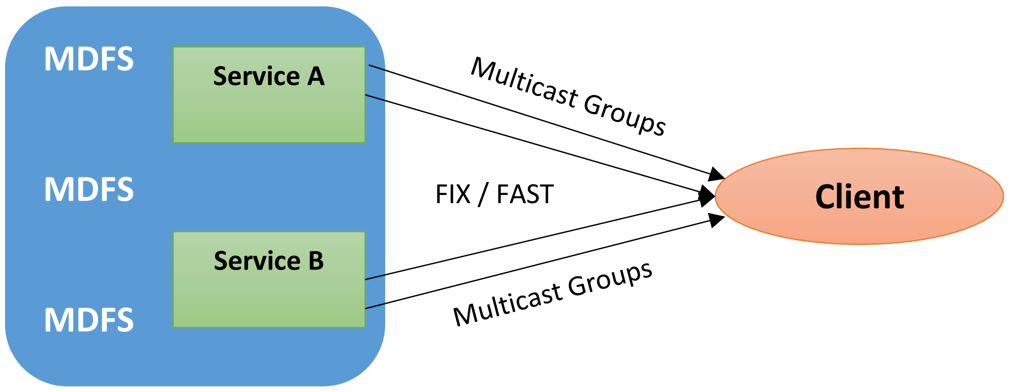

Figure 1 - Multicast Overview

The MDFS utilizes UDP multicast transport for the dissemination of all data. The data is split into different feeds, with each feed receiving messages pertaining to specific Venues, Product Categories, and message types. The following tables are an example of how the multicast groups for each feed are organized:

Venue	Instrument Type	Group	Venue	Instrument Type	Group
Venue 1	Cash & Index	General	Venue 2	Cash & Index	General
		Order Depth			Order Depth
		Top of Book			Top of Book
		Price Depth 5			Price Depth 5
		Price Depth 10			Price Depth 10
		Trades			Trades
	Bonds	General		Bonds	General
		Order Depth			Order Depth
		Top of Book			Top of Book
		Price Depth 5			Price Depth 5
		Price Depth 10			Price Depth 10
		Trades			Trades
	Derivatives	General		Derivatives	General
		Order Depth			Order Depth
		Top of Book			Top of Book
		Price Depth 5			Price Depth 5
		Price Depth 10			Price Depth 10
		Trades			Trades

Figure 2 - Incremental Multicast Groups

Venue	Instrument Type	Group	Venue	Instrument Type	Group
Venue 1	Cash & Index	General Snapshots	Venue 2	Cash & Index	General Snapshots
		Order Depth Snapshots			Order Depth Snapshots
		Top of Book Snapshots			Top of Book Snapshots
		Price Depth 5 Snapshots			Price Depth 5 Snapshots
		Price Depth 10 Snapshots			Price Depth 10 Snapshots
		Trades Snapshots			Trades Snapshots
	Bonds	General Snapshots		Bonds	General Snapshots
		Order Depth Snapshots			Order Depth Snapshots
		Top of Book Snapshots			Top of Book Snapshots
		Price Depth 5 Snapshots			Price Depth 5 Snapshots
		Price Depth 10 Snapshots			Price Depth 10 Snapshots
		Trades Snapshots			Trades Snapshots
	Derivatives	General Snapshots		Derivatives	General Snapshots
		Order Depth Snapshots			Order Depth Snapshots
		Top of Book Snapshots			Top of Book Snapshots
		Price Depth 5 Snapshots			Price Depth 5 Snapshots
		Price Depth 10 Snapshots			Price Depth 10 Snapshots
		Trades Snapshots			Trades Snapshots

Figure 3 - Snapshot Multicast Groups

Each multicast group is served to a specific IP Address & UDP Port combination. All **Incremental** multicast groups will be transmitted via the **UDP port 10000**, and all **Snapshot** multicast Groups will be transmitted via the **UDP port 20000**. Each client connects to multiple feeds that disseminate information relevant to them.

The association of the Instrument Type groupings in the tables above with the value of FIX tag “20011= ATHEXSecurityCategory” can be seen in the following table:

Instrument Type Grouping	Value of FIX Tag “20011= ATHEXSecurityCategory”
Cash & Index	0 = Stock / Rights 1 = ETF 2 = Warrant 3 = Stock Index 4 = ETF Indicative Net Asset Value (INAV)
Bonds	5 = Bond
Derivatives	6 = Option 7 = Future 8 = Repo 9 = Standard Combination

Figure 4 - Instrument Type Groupings

An overview of the messages sent via each Group type can be seen on the following table:

Group Type	Messages
General General Snapshots	Security Status Trading Session Status News Index Value Closing Price Start of Day Price High/Low Limit Modification Instrument Summary Auction Price
Order Depth Order Depth Snapshots	Empty Book Order Depth Update
Top of Book Top of Book Snapshots	Empty Book Top of Book Update
Price Depth 5 Price Depth 5 Snapshots	Empty Book Price Depth Update (Up to 5 levels)
Price Depth 10 Price Depth 10 Snapshots	Empty Book Price Depth Update (Up to 10 levels)
Trades Trades Snapshots	Trade

Figure 5 - Messages per Group type

The details for all message types are available in the "*OASIS MDFS - Message Reference*" document.

There may also exist some multicast groups which do not follow the general structure described in the tables above, the details of which will be made available through other means.

The MDFS replicates all feeds on two identical Services (A & B). This is done to combat the inherent unreliability of the UDP protocol, where the delivery of data packets is not guaranteed resulting in cases of lost packets. It is strongly recommended that clients connect to both services in order to handle any such incidents.

2.2. Incremental Feed Approach

The MDFS follows the paradigm of incremental data feed messages, as outlined by the FIX Trading guideline. This approach relies on Snapshot messages to deliver the initial / current state of all instruments included in the data feed and subsequent incremental messages to keep that state up to date throughout the trading session.

By utilizing this paradigm, the MDFS achieves lower bandwidth consumption and uses a minimal number of commands to update the instruments' order books.

3. Connection Procedure & Data Flow

3.1. Handling Incremental & Snapshot Traffic

All messages received via Incremental and Snapshot feeds will contain the tag “1180 = ApplID” this tag will contain the group’s name (e.g. XATH_CASH_GENERAL) and the “_INCR” or “_SNAP” suffix respectively.

The “_INCR” or “_SNAP” suffixes can be used to differentiate Incremental and Snapshot traffic.

To associate an Incremental feed with the corresponding Snapshot feed, the last five characters of tag “1180 = ApplID” should be removed, thus removing “_INCR” or “_SNAP” suffixes.

3.2. Initial Connection Procedure

A client connection to the MDFS should follow these steps to connect to the data feed and receive real-time information (repeat for each Venue & Product Type of interest):

1. Download reference data using the RDS service.
2. Start listening to the Incremental feed and buffer all messages.
3. Start listening to the Snapshot Feed. Discard all messages until you reach the message indicating the start of a snapshot cycle. Keep listening until you receive the message indicating the end of the snapshot cycle.
4. Disconnect from the Snapshot Feed. Once you have received a full snapshot cycle you will have all the information needed to build the order book baseline for each instrument disseminated in the accompanying incremental stream.
5. Discard all buffered incremental messages with a value less than or equal to the lowest value of tag “369 = LastMsgSeqNumProcessed” among all snapshot messages received in this snapshot cycle.
6. Apply all the remaining buffered incremental messages to the instruments’ order books.
7. Keep processing the incoming incremental messages to update the instruments’ order books in real time.

3.3. Updating the Order Book

As long as the values of tag “34 = MsgSeqNum” in the messages received from the incremental feed are contiguous, the client should keep processing them and applying them to the corresponding order book.

3.4. Handling Data Feeds on Services A&B

As mentioned before the MDFS replicates all feeds on two identical Services (A & B). This is done to combat the inherent unreliability of the UDP protocol, where the delivery of data packets is not guaranteed and there may be cases of lost packets. It is strongly recommended that clients connect to both services in order to handle any such incidents non-disruptively (without resorting to recovery via snapshot).

In a typical scenario the client (assuming they are connected to both Service A & B) should, for each message with a distinct value in tag “34 = MsgSeqNum”, keep the message received first from either service and discard the subsequent copy they receive from the other service.

The following table is an example of the typical data flow on Services A & B, where the shaded cells represent the messages the client should keep, discarding the rest:

Order	**Tag 34 = MsgSeqNum**	
	Service A	**Service B**
1	100	
2		100
3	101	
4		101
5		102
6	102	
7	103	
8		103

Figure 6 - Services A & B Example

3.5. Handling Gaps in Message Sequence Numbers

The client should always check the tag “34 = MsgSeqNum” for gaps in the message sequence of any feed they are connected to. In the case of a gap in the sequence numbers in either of the two services the client should receive the message through the other service (assuming they are connected to both Service A & B).

The following table is an example of a scenario in which a sequence number gap occurs in one of the services, where the shaded cells represent the messages the client should keep:

Order	**Tag 34 = MsgSeqNum**	
	Service A	**Service B**
1	100	
2		100
3	101	
4		101
5		102
6	103	
7		103

Figure 7 - Handling Message Sequence Gaps

In the example above the message with tag “MsgSeqNum = 102” was not received through Service A, but was received through Service B. In this case the client should have no interruption of data flow as they can utilize the message received from Service B.

3.6. Recovery via Snapshot

In the unlikely occasion where a message is not available through either Service A or B then the client should follow the same procedure that is described in section 3.1 Initial Connection Procedure, to perform recovery via snapshot.

3.7. TCP/IP Retransmission Service

In addition to the previously detailed methods of recovery, the MDFS provides a TCP/IP Retransmission Service.

This service provides a way for clients to request a limited number of FIX/FAST messages that have previously been disseminated via the UDP multicast groups. Usage of this service may be desirable if:

- A gap in FIX message sequence numbers is detected in both Service A & B and recovery via Snapshot messages is inadequate.
- The client needs to receive a number of incremental messages that were sent earlier during the day.

The service is only intended as a recovery mechanism, and not as a means to receive the data feed during the day. Some details and limitations of the service are:

- A single retransmission may be ongoing per session at any given time, the MDFS will not accept any retransmission requests while a previous retransmission is ongoing on the same session.
- Each client can send a maximum of 10,000 requests per day.
- Each request can have a maximum range of 1,000 messages.

3.7.1. TCP/IP Retransmission Procedure

The TCP/IP Retransmission Service allows the user to request a specific number of messages that have been sent via a specific multicast group (that the user has access to) and receive them in the same TCP/IP FIX session. The retransmission procedure is as follows:

1. The client establishes a TCP/IP connection with the MDFS.
2. Once the session is established, the client sends a “Logon” message.
3. The MDFS replies to the client’s “Logon” message, accepting it with “Logon” acknowledgment message or rejecting it with a “Logout” message.
4. If the logon attempt was successful, both client and MDFS reset their MsgSeqNum to 1, after which the client sends an “ApplicationMessageRequest” message containing:

- The multicast group the messages are requested from (e.g. “XATH_CASH_GENERAL”). This value is present in tag “1180 = ApplID” of all market data messages, suffixed by “_INCR” or “_SNAP” which should be removed before sending a request.
- The range of sequence numbers of the requested messages.

5. The MDFS replies to the retransmission request, with an “ApplicationMessageRequestAck” message. A successful request will receive the tag “1348 = ApplResponseType” with a value of “0 = Request successfully processed”, or a corresponding error value otherwise. A description for the error value will be available in tag “58 = Text”.
6. If the retransmission request was accepted, the MDFS will start sending the requested messages encapsulated in “UEFD = EncapsulatedFASTData” messages. The tag “95 = RawDataLength” contains the length of the encapsulated message, while the tag “96 = RawData” contains the actual FAST message, as it was sent via multicast.
7. After all requested messages have been sent, the MDFS will send an “ApplicationMessageReport” message to the client, informing him that the retransmission is complete, containing the number of messages sent and the range of sequence numbers sent.
8. After the retransmission is completed, the MDFS will close the TCP/IP session by sending a “Logout” message.

The following diagram illustrates a typical TCP/IP retransmission session:

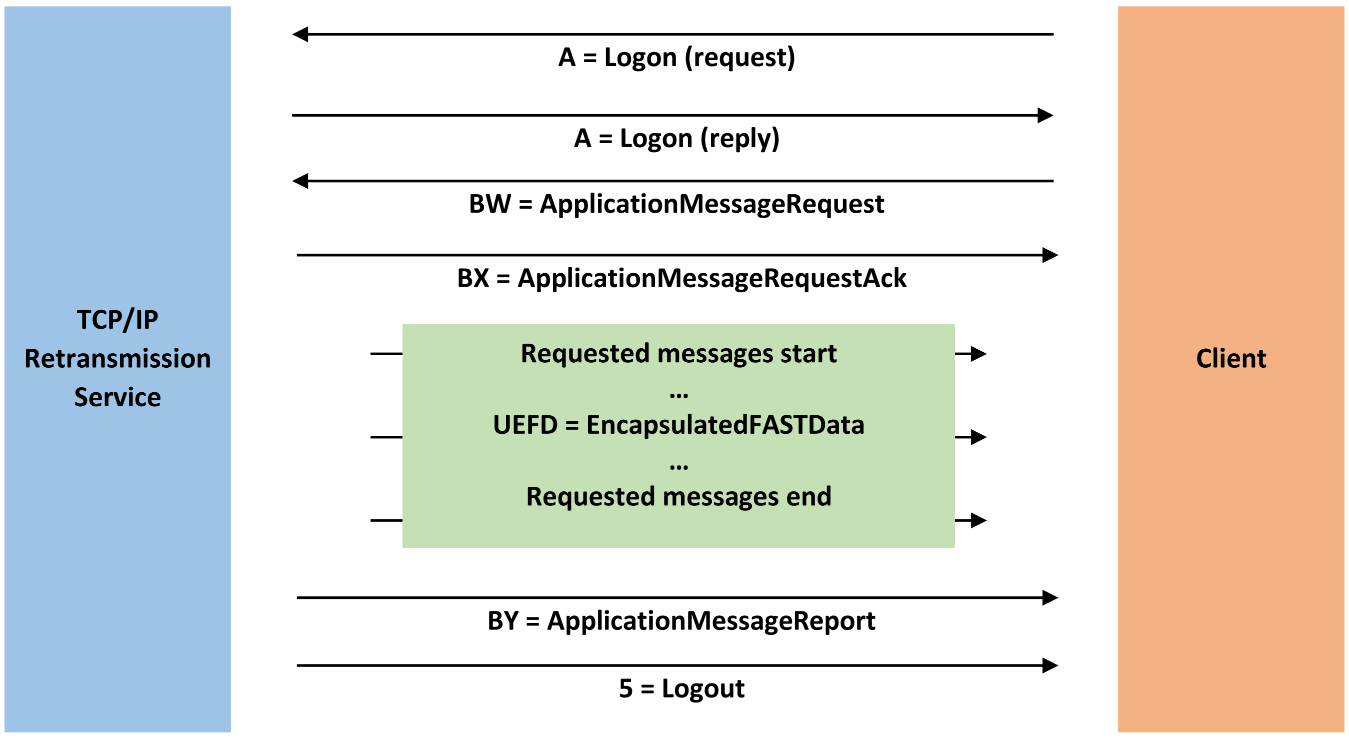
TCP/IP Retransmission Service
A = Logon (request)
A = Logon (reply)
BW = ApplicationMessageRequest
BX = ApplicationMessageRequestAck
Requested messages start
…
UEFD = EncapsulatedFASTData
…
Requested messages end
BY = ApplicationMessageReport
5 = Logout
Client

Figure 8 - TCP/IP Retransmission

Some details regarding the TCP/IP Retransmission procedure above are:

- TCP FIX sessions use **SSL** encryption. To establish an **SSL** connection with MDFS service **TLS v1.3** should be used.
- The requested messages are encapsulated in “UEFD = EncapsulatedFASTData” messages, thus allowing for them to be retransmitted unaltered to the client. The exact same FAST message that was broadcasted in the Multicast Group will be contained in tag “96 = RawData”.
- Heartbeat messages sent in the Multicast Groups always have tag “34 = MsgSeqNum” with a value of “0”, and they are not included in the messages sent via the Retransmission Service, thus the client will only receive Market Data messages in the Retransmission.
- For the FIX session in the Retransmission Service the clients will need to use the appropriate TargetCompID and SenderCompID field values:
  - TargetCompID = “ATHEX”
  - SenderCompID = The Username that will be provided for you to use in the Retransmission Service
- Logon requests may be rejected for the following reasons:
  - The provided credentials are incorrect.
  - The client has another active TCP/IP session for the Retransmission Service.
  - The maximum number of retransmission requests per day has been exceeded.
- Retransmission requests may be rejected for the following reasons:
  - The requested multicast group is invalid.
  - The client is not authorized to access the requested multicast group.
  - The requested messages are not available (i.e. invalid range).
  - The requested range exceeds the maximum retransmission range.
  - The FIX message is ill-formed or does not match the Protocol specifications.
- After the retransmission request has been processed, the MDFS will close the TCP/IP session.
- If a TCP session is opened and the Logon message is not sent within 30 seconds, the MDFS will terminate the connection.
- The MDFS has a timeout of 5 seconds between “Logon” and “ApplicationMessageRequest” messages. If the client does not send a “ApplicationMessageRequest” within 5 seconds of receiving the “Logon” reply, the MDFS will close the session by sending a “Logout” message.
- The MDFS will provide two IP addresses for the TCP/IP Retransmission Service (A & B) that can be used interchangeably.
- The requested retransmission range is specified via the tags “1182 = ApplBegSeqNum” and “1183 = ApplEndSeqNum”. If the “1183 = ApplEndSeqNum” tag’s value is 0, then the server will retransmit messages with sequence numbers starting from “1182 = ApplBegSeqNum” until either the limit per request is reached, or until there are no more messages left to send for the requested multicast group.
- It is the client’s responsibility to send unique ApplReqID identifiers during the day, which are used by the exchange to distinctly identify Retransmission Requests and troubleshoot potential issues.

4. FAST Message Encoding

The FAST Protocol (SM) is developed, maintained and supported by the FIX Trading Community's Market Data Optimization Working Group. The protocol is intended to enable efficient use of bandwidth in high volume messaging without incurring material processing overhead or latency. The MDFS' implementation is based on the FAST 1.2 specification. Please refer to the documentation available at the provided link for more details regarding encoding and decoding FAST FIX messages.

The following methods are utilized for data compression:

- Implicit Tagging
- Optional Fields
- Field Operators
- Presence Maps
- Stop-bit Encoding
- Binary Encoding

These methods are further explained in subsequent sections of this document.

The FAST format encoding rules for MDFS are distributed as XML Templates.

4.1. Packet Structure

The following table is a representation of a FAST Packet:

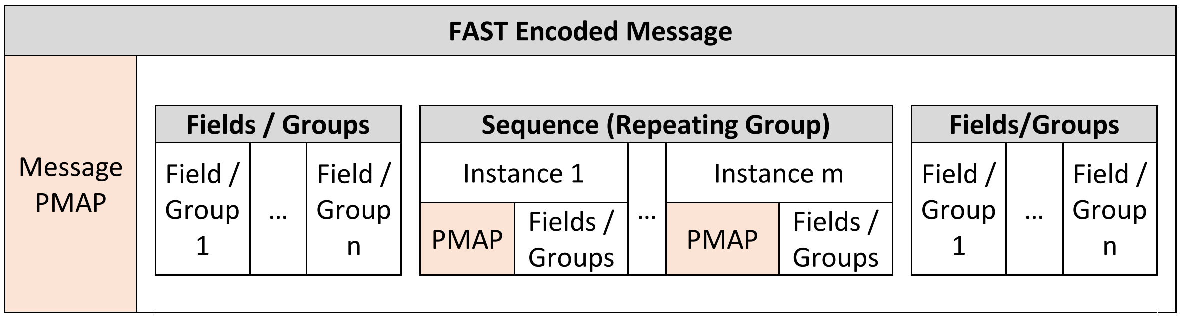

Figure 9 - FAST Packet Structure

Where:

- **Field:** A FAST-encoded FIX tag.
- **Group:** A group of FAST-encoded FIX tags, that usually appear together. Appears as a <group> element in FAST .xml templates.
- **Sequence (Repeating Group):** A FIX repeating group. Appears as a <sequence> element in FAST .xml templates.
- **Instance:** An instance of a FIX repeating group.

4.2. Data Types

The following data types used in FAST templates:

- Signed and unsigned 32/64-bit integer
- Decimal number
- Length
- String - ASCII (7-bit) strings (no special characters allowed)
- Byte vector

4.3. Templates & Implicit Tagging

Every FAST message has a template ID as the first integer field that will be used by the decoder to choose what template will be used to decode it. The template describes what fields from the original FIX message are included, their types and transfer encodings.

By having a fixed field order, FAST templates reduce redundancies within a message, as the field meaning is deferred by its position in the message and there is no need to transfer the field tag to describe the field value. If the original FIX message contains fields that are not specified in the template, they are simply ignored when encoding, and as such do not need to be decoded as well.

There can be several templates for the same FIX message (“MsgType = X’, for instance), but referring to different versions of the message layout.

The templates are distributed in a single XML file. An example of the format can be found in section 6.2 of the appendix.

4.4. Mandatory and Optional Fields

The optional presence attribute indicates whether the field is mandatory or optional. If the attribute is not specified, the field is mandatory.

4.5. Field Operators

Field operators are used to remove redundancies in the data values. The message templates (which are provided beforehand) serve as the metadata for the message. Upon receiving a message, the recipient has complete knowledge of the message layout via the template definition and is able to determine the field values of the incoming message.

The operators used by the MDFS are:

- (None): The field will be encoded as is.
- Constant: The field will always have a predetermined value.
- Default: The field is omitted from the message if it is equal to the default value. Used in MDFS templates to force the usage of a PMAP bit for the field.

More details about these operators can be found in the FAST Specification documents.

4.6. Presence Map (PMAP)

The presence map is a bit map indicating the presence or absence of a field in the message body. One bit is used in the PMAP for each field that requires it. The allocation of a bit for a field in the presence map is governed by the FAST field encoding rules.

4.7. Stop Bit Encoding

All FAST fields are stop bit encoded with the exception of byte vectors. Instead of using a length indicator or the standard FIX-separator (<SOH> byte), each byte consists of 7 bits for data transfer and the 8th bit to indicate the end of a field value.

4.8. Binary Encoding

Binary encoding is used on numbers, rendering them into binary across the 7 data bits in each byte. Thus, a number less than 2^7-1, (127) will only occupy one byte, a number between 2^7 and 2^7*2 – 1 (16,383), will occupy two bytes etc.

4.9. Decoding Overview

The following is a brief overview of the steps required to decode a FAST message to the underlying FIX format:

1. The client receives a FAST encoded FIX message.
2. Template Identification.
3. Extraction of binary encoded bits.
4. Mapping the received bits to template fields.
5. Field decoding using operators to determine values according to the template.
6. Generation and processing of the FIX message.

4.10. Decoding Example

The following table provides a detailed example on how to decode a FAST-encoded message. The template used in this example can be found in section 6.2 of the appendix.

#	Field	Attributes / Operators	Type	Presence	PMAP Bit Required	PMAP bit	Encoded Value	Stop Bit Decoded Value	Value
Message Data **Hex: 0XF8** 0xA2 0x82 0x54 0x45 0X53 0xD4 0x82 **0XB0** 0xFF 0x04 0x9E 0x81 0x02 0xAC **Binary:** 1**1111**000 10100010 10000010 01010100 01000101 01010011 11010100 10000010 1**011**0000 1111111 00000100 10011110 10000001 00000010 10101100 **Message PMAP:** 1**1111**000									
	Template ID	None	uInt32	Mandatory	true	1	10100010	_0100010	34
1	35 =MsgType	Constant	string	Mandatory	false				“W”
2	1021 = MDBookType	Default	uInt32	Optional	true	1	10000010	_0000010*	1
3	55 = Symbol	Default	string	Optional	true	1	01010100 01000101 01010011 11010100	01010100 01000101 01010011 01010100	“TEST”
Sequence (Repeating Group) Data **Hex:** 0x82 **0xB0** 0xFF 0x04 0x9E 0x81 0x02 0xAC **Binary:** 10000010 1**011**0000 1111111 00000100 10011110 10000001 00000010 10101100									
4	268 = NoMDEntries	Default	length	Mandatory	true	1	10000010	_0000010*	1
Repeating Group Instance Data **Hex: 0xB0** 0xFF 0x04 0x9E 0x81 0x02 0xAC **Binary:** 1**011**0000 1111111 00000100 10011110 10000001 00000010 10101100 **PMAP:** 1**011**0000									
5	1023 = MDPriceLevel	Default	uInt32	Optional	true	0			
6	270 = MDEntryPx	Default	decimal	Optional	true	1	Exponent: 1**1**11111 Mantissa: 00000100 10011110	Exponent: _**1**111111 → 10^-1 Mantissa: _0000100 _0011110 →542	54.2
7	271 = MDEntrySize	Default	decimal	Optional	true	1	Exponent: 1**0**000001 Mantissa: 00000010 10101100	Exponent: _**0**000001* → 10^0 Mantissa: _0000010 _0101100 → 300	300

Figure 10 - FAST Decoding Example

* To decode **Positive** arithmetic fields that are nullable (according to the FAST protocol standard) we need to take the positive value of the result (without the stop bits) and subtract 1 from it. That is why i.e NoMDEntries which results to 00000010 without the stop bit is translated to 1, or why the exponent for MDEntrySize which results to 0000001 without stop bit is translated to 0.

Note:

Utilized PMAP bits are in **bold.**

Stop bits are underlined.

5. Order Book Handling

This section contains instructions on how to maintain the different types of order books for an instrument.

The three types of order books supported by the MDFS are:

- Top of Book
- Price Depth
- Order Depth

For each instrument, the client can keep these order books up to date by following the instructions contained in this section when processing the Incremental messages received through the MDFS. Keep in mind that:

- In accordance with FIX guidelines all order book handling instructions are handled by messages that contain repeating groups with tag "269=MDEntryType" having value "0 = Bid", "1 = Offer" or “J = Empty book”. As such any other repeating group types, such as those with tag "269=MDEntryType" having value "2 = Trade" should not be used to alter the order book, as the appropriate Order Depth Update messages for each side of the trade will be sent containing the appropriate order book maintenance actions.
- There is no parity in the values of tag “60 = TransactTime” between the Oder Depth & Top of Book/Price Depth books as these are handled independently, i.e. the value of tag “60 = TransactTime” of an Orde Depth Update message will be different from that of the Top of Book/Price Depth Update message that is triggered by the same order altering both books.

Note: Some tags that do not affect the handling of the orders books will be omitted from the example messages included in this section to improve readability. The actual messages transmitted will include additional tags.

5.1. Market/Stop/ATO/ATC orders

This section details the handling of Market/Stop/ATO/ATC orders in the order & price depth books.

5.1.1. Order Depth Book

Market/Stop (value “1=Market”/” = Stop” in tag “40 = OrdType”) and ATO/ATC orders (value “2 = At the Opening (OPG)”/“7 = At the Close” in tag “59 = TimeInForce”) do not have a set price (thus do not contain the tag “270 = MDEntryPx”). Those orders are always placed at the top of the order depth book with the value of “b = Market Bid”/”c = Market Offer” in tag “269 = MDEntryType” and are ordered by their release timestamp in the matching engine.

5.1.2. Top of Book/Price Depth Book

The volume of Market+ATO or ATC orders is disseminated via the Top of Book / Price Depth books. A repeating group with the value of “b = Market Bid”/”c = Market Offer” in tag “269 = MDEntryType” will be sent to update the volume and number of orders placed for the opening auction (Market+ATO), closing auction (ATC) and any other intraday auction. These repeating group entries do not contain the tag “270 = MDEntryPx”.

5.2. Empty Book

Instructs the client to empty a book of a specific instrument. Typically sent at the start of the trading session.

Example Message:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	J = Empty book
264	MarketDepth	10 = 10 Levels

A similar message may be sent for any order book type.

5.3. Top of Book

This type of order book contains only be top price level for an instrument.

Incremental Refresh messages relevant to the Top of Book of an instrument are sent multiple times during each trading session in order to give the client the information necessary to keep it up to date.

Examples of how to handle the various possible scenarios follow.

5.3.1. New – Addition to an empty side

Consider the following initial state for the client's Top of Book order book:

Bid			Offer		
Price	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
-	-	-	70	20	4

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	1 = Top of Book
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	50
271	MDEntrySize	10
264	MarketDepth	1 = Top of Book
1023	MDPriceLevel	1
346	NumberOfOrders	2

The message above indicates a new Top of Book entry for the previously empty bid side. This results in the client's Top of Book order book looking as follows:

Bid			Offer		
Price	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
50	*10*	*2*	70	20	4

5.3.2. Change – Change of volume / no. of orders

Consider the following initial state for the client's Top of Book order book:

Bid			Offer		
Price	Volume	No. of Orders	Price	Volume	No. of Orders
50	10	2	70	20	4

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	1 = Top of Book
279	MDUpdateAction	1 = Change
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	50
271	MDEntrySize	4
264	MarketDepth	1 = Top of Book
1023	MDPriceLevel	1
346	NumberOfOrders	1

The message above indicates a change in the volume and no. of orders at the bid side. This results in the client's Top of Book order book looking as follows:

Bid			Offer		
Price	Volume	No. of Orders	Price	Volume	No. of Orders
50	*4*	*1*	70	20	4

5.3.3. Delete – A side becomes empty

Consider the following initial state for the client's Top of Book order book:

Bid			Offer		
Price	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
50	4	1	60	6	1

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	1 = Top of Book
279	MDUpdateAction	2 = Delete
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	60
271	MDEntrySize	6
264	MarketDepth	1 = Top of Book
1023	MDPriceLevel	1
346	NumberOfOrders	1

The message above indicates that there are no orders at the offer side for the instrument, resulting in an empty Top of Book. This results in the client's Top of Book order book looking as follows:

Bid			Offer		
Price	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
50	4	1	-	-	-

5.4. Price Depth Book

This type of order book contains the best bids and offers for an instrument, aggregated by price. The maximum number of levels provided for each price order book depends on the multicast group it is disseminated through.

Incremental Refresh messages relevant to the Price Depth order book of an instrument are sent multiple times during each trading session in order to give the client the information necessary to keep it up to date.

Examples of how to handle the various possible scenarios follow. The scenarios below assume a max Price Depth of 3 (tag "264 = MarketDepth" = 3) for simplicity's sake, but the same concepts apply for any depth.

5.4.1. New – Level insertion at the bottom of the book

Consider the following initial state for the client's Price Depth order book:

	Bid			**Offer**		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	40	2	1	90	6	3
3	-	-	-	100	5	2

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	30
271	MDEntrySize	4
264	MarketDepth	3
1023	MDPriceLevel	3
346	NumberOfOrders	1

The message above indicates a new level at the bottom of the bid side. This results in the client's Price Depth order book looking as follows:

	Bid			**Offer**		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	40	2	1	90	6	3
3	*30*	*4*	*1*	100	5	2

5.4.2. New – Level insertion, causing a shift

Consider the following initial state for the client's Price Depth order book:

	Bid			Offer		
Level	Price	Volume	No. of Orders	Price	Volume	No. of Orders
1	60	5	2	80	4	1
2	40	7	2	90	6	3
3	30	4	1	-	-	-

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	85
271	MDEntrySize	2
264	MarketDepth	3
1023	MDPriceLevel	2
346	NumberOfOrders	1

The message above indicates the insertion of a new level at position 2 of the offer side. When processing this message, the client should shift the entry that was previously at this level, as well as all levels below it down by one level. In this example the entry with Price = 90 is shifted, going from level 2 to 3. This results in the client's Price Depth order book looking as follows:

	Bid			Offer		
Level	Price	Volume	No. of Orders	Price	Volume	No. of Orders
1	60	5	2	80	4	1
2	40	7	2	*85*	*2*	*1*
3	30	4	1	90	6	3

5.4.3. New – Level insertion, causing the deletion of the last level

Consider the following initial state for the client's Price Depth order book:

	Bid			**Offer**		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	60	5	2	80	4	1
2	40	7	2	85	2	1
3	30	4	1	90	6	3

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	35
271	MDEntrySize	3
264	MarketDepth	3
1023	MDPriceLevel	3
346	NumberOfOrders	1

The message above indicates the insertion of a new price level at position 3 of the bid side. When processing this message, the client would shift the entry that was previously at this position, as well as all levels below it down by one level. In this example the entry with Price = 30 is shifted down by one level, going from 3 to 4, thus exceeding the max book depth, and as such should be deleted. This results in the client's Price Depth order book looking as follows:

		Bid			**Offer**		
	Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
Max book depth 3	1	60	5	2	80	4	1
	2	40	7	2	85	2	1
	3	*35*	*3*	*1*	90	6	3
Exceeds max depth		*30*	*4*	*1*			

↓

	Bid			**Offer**		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	60	5	2	80	4	1
2	40	7	2	85	2	1
3	*35*	*3*	*1*	90	6	3

5.4.4. Change – Change of a level's volume / no. of orders

Consider the following initial state for the client's Price Depth order book:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	40	2	1	90	6	3
3	30	4	1	-	-	-

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	1 = Change
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	40
271	MDEntrySize	7
264	MarketDepth	3
1023	MDPriceLevel	2
346	NumberOfOrders	2

The message above indicates a change in the volume and no. of orders at level 2 of the bid side. This results in the client's Price Depth order book looking as follows:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	*40*	*7*	*2*	90	6	3
3	30	4	1	-	-	-

5.4.5. Delete – Level deletion from the bottom of the book

Consider the following initial state for the client's Price Depth order book:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	40	2	1	90	6	3
3	30	4	1	100	5	2

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	2 = Delete
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	100
271	MDEntrySize	5
264	MarketDepth	3
1023	MDPriceLevel	3
346	NumberOfOrders	2

The message above indicates the deletion of a level at the bottom of the offer side. This results in the client's Price Depth order book looking as follows:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	80	4	1
2	40	2	1	90	6	3
3	30	4	1	-	-	-

5.4.6. Delete – Level deletion, causing a shift

Consider the following initial state for the client's Price Depth order book:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	60	5	2	80	4	1
2	40	7	2	85	2	1
3	30	4	1	90	6	3

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	2 = Delete
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	60
271	MDEntrySize	5
264	MarketDepth	3
1023	MDPriceLevel	1
346	NumberOfOrders	2

The message above indicates the deletion of the first level of the bid side. When processing this message, the client should remove the level and shift all levels below up by one level. In this example levels 2 and 3 are shifted up by one level. This results in the client's Price Depth order book looking as follows:

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	-	-	-	80	4	1
2	40	7	2	85	2	1
3	30	4	1	90	6	3

↓

	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	40	7	2	80	4	1
2	30	4	1	85	2	1
3	-	-	-	90	6	3

5.5. Order Depth Book

This type of order book contains the full order depth for a given instrument. Incremental Refresh messages relevant to the Order Depth order book of an instrument are sent multiple times during each trading session in order to give the client the information necessary to keep it up to date.

Examples of how to handle the various possible scenarios follow.

5.5.1. New – Entry Insertion at the bottom of the book

Consider the following initial state for the client's Order Depth order book:

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	3	109
4	40	4	101	4	90	4	103
5	30	1	100	5	90	5	120
6	30	7	104				

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	90
271	MDEntrySize	3
290	MDEntryPositionNo	6
37	OrderID	121

The message above indicates a new order with price 90 at position 6 of the offer side. This results in the client's Order Depth order book looking as follows:

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	3	109
4	40	4	101	4	90	4	103
5	30	1	100	5	90	5	120
6	30	7	104	*6*	*90*	*3*	*121*

5.5.2. New – Entry insertion, causing a shift

Consider the following initial state for the client's Order Depth order book:

Bid				Offer			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	3	109
4	40	4	101	4	90	4	103
5	30	1	100	5	90	5	120
6	30	7	104	6	90	3	121

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	0 = New
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	40
271	MDEntrySize	3
290	MDEntryPositionNo	5
37	OrderID	122

The message above indicates a new order with price 40 at position 5 of the bid side. When processing this message, the client should shift the entry that was previously at this position, as well as all positions below it by one. This results in the client's Order Depth order book looking as follows:

Bid				Offer			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	3	109
4	40	4	101	4	90	4	103
5	*40*	*3*	*122*	5	90	5	120
6	30	1	100	6	90	3	121
7	30	7	104				

5.5.3. Change – Change of an entry's volume

The value "1 = Change" for tag "279 = MDUpdateAction" signals a change to an order's volume. Note that this is only used when the order's volume is **decreased**, as an increase in volume could potentially change the order's position and as such would be disseminated by a "3 = Delete" instruction, followed by a "0 = New" instruction.

Consider the following initial state for the client's Order Depth order book:

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	3	109
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121
7	30	7	104				

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	1 = Change
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	80
271	MDEntrySize	2
290	MDEntryPositionNo	3
37	OrderID	109

The message above indicates a change in volume at position 3 of the offer side. This results in the client's Order Depth order book looking as follows:

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	*3*	*80*	*2*	*109*
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121
7	30	7	104				

5.5.4. Delete – Entry deletion from the bottom of the book

Consider the following initial state for the client's Order Depth order book:

Bid				Offer			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121
7	30	7	104				

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	2 = Delete
55	Symbol	Example Instrument
269	MDEntryType	0 = Bid
270	MDEntryPx	30
271	MDEntrySize	7
290	MDEntryPositionNo	7
37	OrderID	104

The message above indicates the deletion of the entry at position 7 of the bid side. This results in the client's Order Depth order book looking as follows:

Bid				Offer			
Position	Price	Volume	Order ID	Position	Price	Volume	Order ID
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121
7	-	-	-				

↓

Bid				Offer			
Position	Price	Volume	Order ID	Position	Price	Volume	Order ID
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121

5.5.5. Delete – Entry deletion, causing a shift

Consider the following initial state for the client's Order Depth order book:

Bid				Offer			
Position	Price	Volume	Order ID	Position	Price	Volume	Order ID
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	4	90	4	103
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	2 = Delete
55	Symbol	Example Instrument
269	MDEntryType	1 = Offer
270	MDEntryPx	90
271	MDEntrySize	4
290	MDEntryPositionNo	4
37	OrderID	103

The message above indicates a deletion of the entry at position 4 of the offer side. When processing this message, the client should remove the entry and shift all entries below up by one position. In this example levels 5 and 6 are shifted up by one level. This results in the client's Order Depth order book looking as follows:

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	*4*	*90*	*4*	*103*
5	40	3	122	5	90	5	120
6	30	1	100	6	90	3	121

↓

Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	1	70	4	110
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	40	4	101	4	90	5	120
5	40	3	122	5	90	3	121
6	30	1	100				

5.6. Order Books in Snapshots

The Snapshots received in the various types of multicast groups contain all the required information to construct the order books for each instrument.

The Snapshot messages follow the same format as the Incremental messages described in the previous sections, with the following differences:

- The tag "35 = MsgType" contains the value "W = MarketDataSnapshotFullRefresh".
- The tag "279 = MDUpdateAction" is absent, all messages are treated as if the value was "0 = New".
- An "Empty Book" message is contained in Snapshots for instruments with an empty book of that type.

By applying the same methods described in the previous sections and taking into considerations these differences, a client can construct the instrument's initial order books by utilizing the snapshots and keep them up to date by applying the incremental feeds.

6. Instrument Prices Handling

This section contains useful information related to the handling of various instrument prices (such as auction prices and closing price) via Incremental Refresh messages.

6.1. Auction Prices

The following procedure describes the handling of an instrument's auction prices via Incremental Refresh messages:

1. An instrument enters an auction pre-call phase. A "f = SecurityStatus" message with tag "625=TradingSessionSubID" having value "102 = Pre-Call (Auction)" will be sent.
2. While the pre-call phase is underway "X = MarketDataIncrementalRefresh" messages with one repeating group with tag "269=MDEntryType" having value "v = Projected Auction Price" will be sent that contain a projection of the auction price. These messages have tag "279=MDUpdateAction" having value "0 = New" for the first message and "1 = Change" for all subsequent updates.
3. Once the auction concludes "f = SecurityStatus" message with tag "625=TradingSessionSubID" having value "2 = Opening (Auction Price is calculated)" will be sent.
4. Subsequently a message "X = MarketDataIncrementalRefresh" messages two repeating groups will be sent:
    a. The first repeating group with tag "269=MDEntryType" having value "v = Projected Auction Price" and tag "279=MDUpdateAction" having value "2 = Delete" will be sent to signify that the pre-call phase has ended and therefore the projected price should be discarded.
    b. The second repeating group with tag"269=MDEntryType" having value "w = Auction Price" and tag "279=MDUpdateAction" having value "0 = New" will be sent containing the actual auction price.
5. This process is repeated for all auctions during the trading day, including the opening and closing auctions. If a price for another auction was sent previously, the first projected price message (described in step 2) will contain an additional repeating group at the start with tag "269=MDEntryType" having value "w = Auction Price" and tag "279=MDUpdateAction" having value "2 = Delete" to signify that a new auction pre-call phase is starting, and as such the previous auction's price should be discarded.

6.2. Closing Price

The following procedure describes the handling of an instrument's closing price via Incremental Refresh messages:

1. An instrument enters the closing auction's pre-call phase. A "f = SecurityStatus" message with tag "625=TradingSessionSubID" having value "102 = Pre-Call (Auction)" will be sent.
2. While the pre-call phase is underway "X = MarketDataIncrementalRefresh" messages with one repeating group with tag "269=MDEntryType" having value " u = Projected Closing Price" will be sent that contain a

projection of the closing price. These messages have tag "279=MDUpdateAction" having value "0 = New" for the first message and "1 = Change" for all subsequent updates.

3. Once the closing auction concludes "f = SecurityStatus" message with tag "625=TradingSessionSubID" having value "2 = Opening (Auction Price is calculated)" will be sent.
4. Subsequently a message "X = MarketDataIncrementalRefresh" messages two repeating groups will be sent:
    a. The first repeating group with tag "269=MDEntryType" having value " u = Projected Closing Price" and tag "279=MDUpdateAction" having value "2 = Delete" will be sent to signify that the pre-call phase has ended and therefore the projected closing price should be discarded.
    b. The second repeating group with tag"269=MDEntryType" having value " 5 = Closing price" and tag "279=MDUpdateAction" having value "0 = New" will be sent containing the actual closing price.

Note: An instrument's closing price is not always equal to its closing auction price, thus projections and prices for both the closing auction and the closing price itself are sent.

7. Appendix

7.1. Comparison With Legacy IDS Service (IOCP)

The MDFS is intended to completely replace the legacy IDS Service (IOCP).

The main differences between the two systems are:

1. **The way they approach the dissemination of the market data originating from the trading platform.**

   The IDS Service is at its core a translation of internal messages generated by the trading platform to the proprietary IDS format messages, more tailored to fit the needs of the clients (exchange members & data vendors). The client has the option to request retransmission of previously disseminated data in the exact form it was previously transmitted as.

   In contrast the MDFS is focused on providing fast, up-to-date information on the current state of all the instruments being traded in the trading platform and on keeping the various order books current. The messaging protocol is no longer proprietary, but the industry standard FIX / FAST protocol is used.

2. **The incremental / snapshot paradigm.**

   The IDS service would send redundant and duplicate information on many occasions, as a result of not following an incremental update approach. Messages would contain information that had already been transmitted previously, when only a small subset of fields had changed. Clients would also need to have received the entirety of the market data messages generated during a trading session in order to be up to date with the current state of the session.

   The MDFS by following the incremental update / snapshot approach can minimize the sending of redundant information and improve the efficiency of the data transmission. In addition, by providing snapshots, the MDFS offers clients the option to get the current state of the trading session in a fast and efficient manner, without having to receive and process any past data they may not be interested in.

3. **The networking and architectural paradigms they employ.**

   The IDS Service uses TCP networking for all communication with the client, this necessitates the existence of a session protocol (implemented through the IOCP's Control channel) in addition to the data transmission channels. This provides reliable transmission but comes with considerable overhead, with message retransmissions further impacting performance.

   The MDFS uses UDP multicast to disseminate market data, in accordance with industry standards and best practices. This approach, when combined with FAST message encoding, results in much lower latency and bandwidth usage. Another benefit of this approach is the lack of a need for a session protocol as all authentication / authorization is done at the network level, which simultaneously allows for more granular access to different feed types. It does come with some inherent unreliability due to the nature of the UDP network protocol, but the MDFS' architecture has multiple ways to combat this such as the concurrent A & B Services and the snapshot recovery mechanism.

4. **The timestamps format they use.**

   The MDFS follows the FIX Protocol standard of sending all timestamps in UDP and YYYYMMDD-HH:MM:SS.ssssss format. This is in contrast to the legacy IDS service which sent all timestamps in local time and YYYYMMDDHHMMSSssssss format.

Those differences result in the MDFS being a much more modern and performant service, that improves the way clients access the exchange's market data feed and potentially reduces costs by providing data in an established and widely used format.

7.2. FAST Template XML Example

The following FAST Template is an example of the format that is used by MDFS to encode & decode FIX messages. An XML file with templates for all of MDFS' message types is provided.

```xml
<template id="34" name="ExampleMessage">
  <string name="MsgType" id="35">
    <constant value="W"/>
  </string>
  <uInt32 name="MDBookType" id="1021" presence="optional">
    <default />
  </uInt32>
  <string name="Symbol" id="55" presence="optional">
    <default/>
  </string>
  <sequence name="MDTestGroup" presence="optional">
    <length name="NoMDEntries" id="268">
      <default/>
    </length>
    <uInt32 name="MDPriceLevel" id="1023" presence="optional">
      <default/>
    </uInt32>
    <decimal name="MDEntrySize" id="271" presence="optional">
      <default/>
    </decimal>
    <decimal name="MDEntryPx" id="270" presence="optional">
      <default/>
    </decimal>
  </sequence>
</template>
```