IT Development Division

Trading Systems Development Department

ATHEX
Athens Stock Exchange

MARKET DATA FEED

OASIS MDFS Specification

Version: 0.12

Revision History

Version	Date	Description
0.12	2024/03/11	UAT release.

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

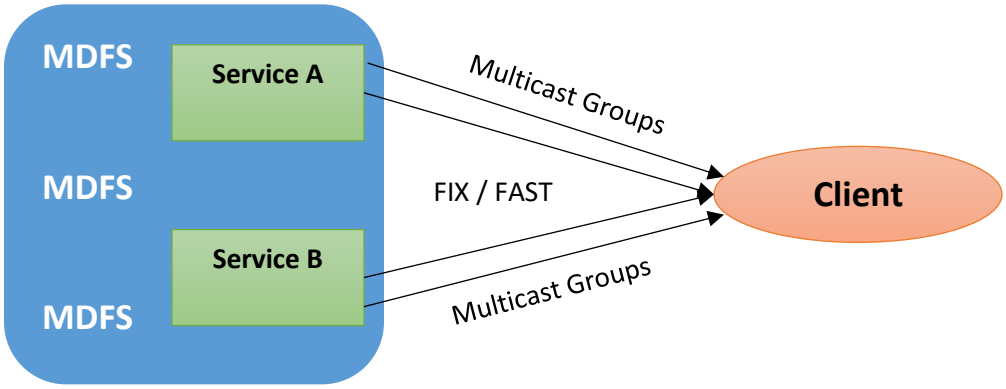

Figure 1 - Multicast Overview

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 "167=SecurityType" can be seen in the following table:

Instrument Type Grouping	Value of FIX Tag "167=SecurityType"
Cash & Index	CS = Common Stock ETF = Exchange Traded Fund WAR = Warrant INDEX = General type for a contract based on an established index INAV = ETF Indicative Net Asset Value
Bonds	BOND = Bond
Derivatives	FUT = Future MLEG = Multileg Instrument OPT = Option REPO = Repurchase

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. 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.

   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.
4. 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.
5. Apply all the remaining buffered incremental messages to the instruments' order books.
6. Keep processing the incoming incremental messages to update the instruments' order books in real time.

3.2. 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.

3.3. 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

3.4. 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.

3.5. 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.

3.6. 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 session may be active per client at any given time, the MDFS will not accept any sessions after the first.
- Each client can send a maximum of 10,000 requests per day.
- Each request can have a maximum range of 1,000 messages.

3.6.1. 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 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, the client sends an “ApplicationMessageRequest” message containing:
    - The multicast group the messages are requested from.
    - The range of sequence numbers of the requested messages.
5. The MDFS replies to the retransmission request, accepting with an “ApplicationMessageRequestAck” message or rejecting it with a “Reject” message.
6. If the retransmission request was accepted, the MDFS will start sending the requested messages encapsulated in “MulticastDataRetransmission” messages. The tag “95 = RawDataLength” contains the length of the encapsulated message, while the tag “96 = RawData” contains the actual 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:

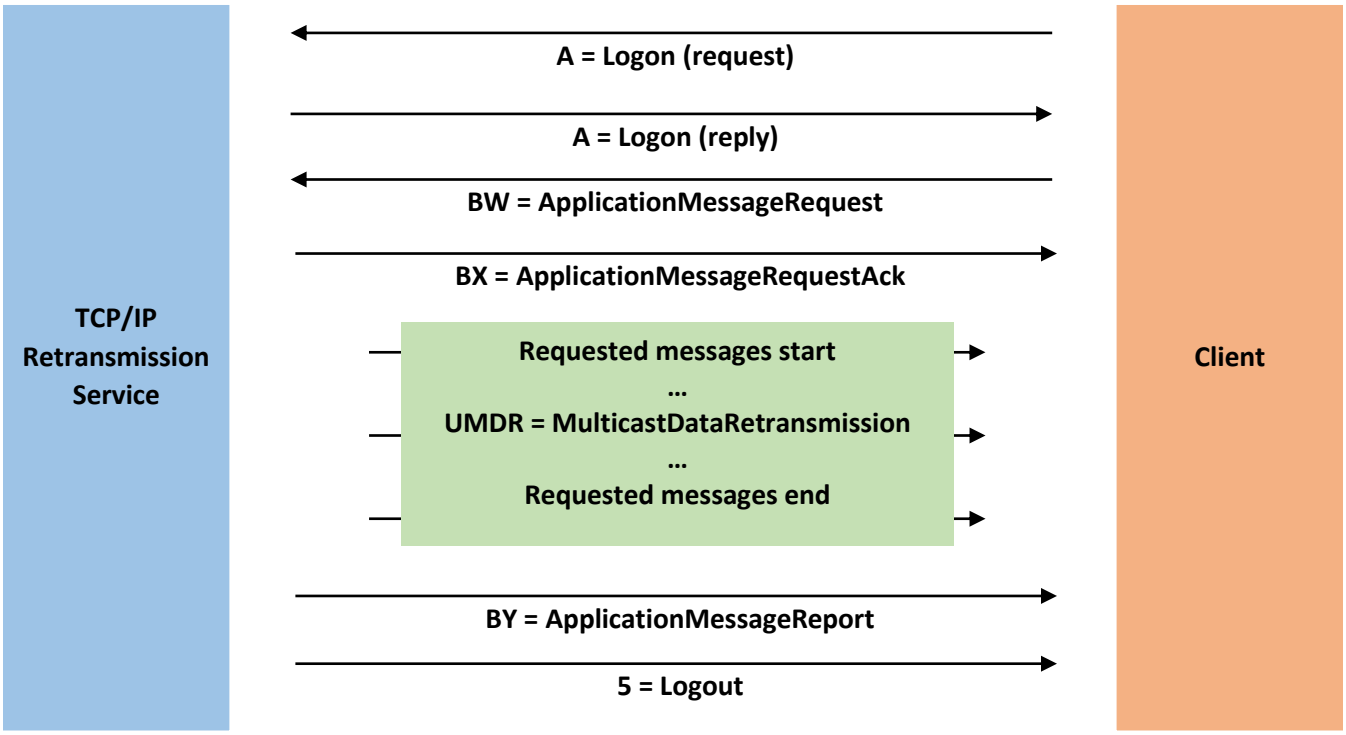

Figure 8 - TCP/IP Retransmission

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

- The requested messages are encapsulated in "MulticastDataRetransmission" messages, thus allowing for them to be retransmitted unaltered to the client, matching exactly those broadcasted via multicast.
- In order to save bandwidth, any "Heartbeat" messages that were originally sent via multicast are not included in the retransmission. As a result, consecutive "MulticastDataRetransmission" messages may contain encapsulated messages that have a gap in sequence numbers (tag "34 = MsgSeqNum") and the client should be prepared to handle them.
- 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.
  - A request has an invalid or duplicate ID in tag "1346 = ApplReqID".
- After the retransmission request has been processed, the MDFS will close the TCP/IP session.
- If the client does not send a "Logon" message within 5 seconds of opening the TCP/IP session, the MDFS will close the session by sending a "Logout" message.

- 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.

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.

The following methods are utilized for data compression:

- Implicit Tagging
- 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. 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

4.2. Packet Structure

The following table is a representation of a FAST Packet:

Header	FAST Encoded FIX Message		
MsgSeqNum	PMAP	Template ID	Message Data
4 Bytes	N Bytes	1 Byte	N Bytes

Figure 9 - FAST Packet Structure

Where:

- **MsgSeqNum:** Contains the same value as tag "34=MsgSeqNum". This allows the client to filter duplicate packets (from Services A & B) without needing to decode the FAST message, decreasing processing overhead.
- **PMAP (Presence Map):** A bit vector that helps the decoder to find if data is present or it is implied (omitted). It occurs at the beginning of each FAST message and at the beginning of every sequence/group.
- **Template ID:** Describes the layout and characteristics of the message to be decoded.
- **Message Data:** The FAST encoded FIX message data.

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 5.1 of the appendix.

4.4. 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.
- Copy: The field will be omitted if it was already used with the same value.
- Increment: If the difference between the current value and the previous value is exactly 1, the field can be omitted.
- Delta: Encode the difference between the previous value and the current value.

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

4.5. 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.

4.6. 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 8^{th} bit to indicate the end of a field value.

4.7. 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.

4.8. 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.

5. Order Book Handling

This section contains in-depth 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 the 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.

Note 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.

5.1. Market & ATO/ATC orders

Market orders and At the Opening/At the close orders (value “Market” in tag “40 = OrdType” and values “2 = At the Opening (OPG)”/”7 = At the Close“ in tag “40 = OrdType”) 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 book for all book types, and are ordered by their release timestamp in the matching engine.

5.2. Empty Book

Instructs the client to empty an order book of a specific instrument, in order to avoid order book corruption, in the rare occasion of a malfunction in the matching engine.

Example Message:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
RG 268	**NoMDEntries**	1
279	MDUpdateAction	0 = New
269	MDEntryType	J = Empty book
55	Symbol	Example Instrument
207	SecurityExchange	Example Venue
20001	ATHEXMarketID	M
RG End		

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

5.3. Top of the Book

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

Incremental Refresh messages relevant to the Top of the 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 the Book order book:

Example Instrument					
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
RG 268	**NoMDEntries**	1
279	MDUpdateAction	0 = New
269	MDEntryType	0 = Bid
270	MDEntryPx	50
271	MDEntrySize	10
346	NumberOfOrders	2
55	Symbol	Example Instrument
207	SecurityExchange	Example Venue
20001	ATHEXMarketID	M
RG End		

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

Example Instrument					
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 the Book order book:

Example Instrument					
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
RG 268	**NoMDEntries**	1
279	MDUpdateAction	1 = Change
269	MDEntryType	0 = Bid
270	MDEntryPx	50
271	MDEntrySize	4
346	NumberOfOrders	1
55	Symbol	Example Instrument
207	SecurityExchange	Example Venue
20001	ATHEXMarketID	M
RG End		

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 the Book order book looking as follows:

Example Instrument					
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 the Book order book:

Example Instrument					
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
RG 268	**NoMDEntries**	1
279	MDUpdateAction	2 = Delete
269	MDEntryType	1 = Offer
270	MDEntryPx	60
271	MDEntrySize	6
346	NumberOfOrders	1
55	Symbol	Example Instrument
207	SecurityExchange	Example Venue
20001	ATHEXMarketID	M
RG End		

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

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

5.3.4. Overlay - The best price changes

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

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

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	1 = Top of Book
RG 268	**NoMDEntries**	1
279	MDUpdateAction	5 = Overlay
269	MDEntryType	1 = Offer
270	MDEntryPx	60
271	MDEntrySize	6
346	NumberOfOrders	1
55	Symbol	Example Instrument
207	SecurityExchange	Example Venue
20001	ATHEXMarketID	M
RG End		

The message above indicates a new best offer price, possibly along with changes in Volume and No. of Orders. This results in the client's Top of the Book order book looking as follows:

Example Instrument					
Bid			**Offer**		
Price	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
50	4	1	*60*	*6*	*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 omit any tags not relevant to the handling of the order book from the messages and assume a max Price Depth of 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:

	Example Instrument					
	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
269	MDEntryType	0 = Bid
1023	MDPriceLevel	3
270	MDEntryPx	30
271	MDEntrySize	4
346	NumberOfOrders	1
55	Symbol	Example Instrument

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:

	Example Instrument					
	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:

	Example Instrument					
	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
269	MDEntryType	1 = Offer
1023	MDPriceLevel	2
270	MDEntryPx	85
271	MDEntrySize	2
346	NumberOfOrders	1
55	Symbol	Example Instrument

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:

	Example Instrument					
	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:

	Example Instrument					
	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
269	MDEntryType	0 = Bid
1023	MDPriceLevel	3
270	MDEntryPx	35
271	MDEntrySize	3
346	NumberOfOrders	1
55	Symbol	Example Instrument

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:

		Example Instrument					
		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*			

↓

	Example Instrument					
	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:

	Example Instrument					
	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
269	MDEntryType	0 = Bid
1023	MDPriceLevel	2
270	MDEntryPx	40
271	MDEntrySize	7
346	NumberOfOrders	2
55	Symbol	Example Instrument

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:

	Example Instrument					
	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:

	Example Instrument					
	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
269	MDEntryType	1 = Offer
1023	MDPriceLevel	3
270	MDEntryPx	100
271	MDEntrySize	5
346	NumberOfOrders	2
55	Symbol	Example Instrument

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:

	Example Instrument					
	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:

	Example Instrument					
	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
269	MDEntryType	0 = Bid
1023	MDPriceLevel	1
270	MDEntryPx	60
271	MDEntrySize	5
346	NumberOfOrders	2
55	Symbol	Example Instrument

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:

	Example Instrument					
	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

↓

	Example Instrument					
	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

5.4.7. Delete Thru – Deletion of multiple levels, causing a shift

The value "3 = Delete Thru" for tag "279 = MDUpdateAction" is used instead of sending multiple deletion instructions when an order matches multiple levels. It signals the deletion of all levels starting from 1 up to (and including) the level specified in tag "1023 = MDPriceLevel".

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

	Example Instrument					
	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	70	4	1
2	40	7	2	80	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	3 = Delete Thru
269	MDEntryType	0 = Bid
1023	MDPriceLevel	2
270	MDEntryPx	40
271	MDEntrySize	7
346	NumberOfOrders	2
55	Symbol	Example Instrument

The message above indicates the deletion of all levels starting from 1 up to 2. This shifts price level 3 to the top level. This results in the client's Price Depth order book looking as follows:

	Example Instrument					
	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	-	-	-	70	4	1
2	-	-	-	80	2	1
3	30	4	1	90	6	3

↓

	Example Instrument					
	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	30	4	1	70	4	1
2	-	-	-	80	2	1
3	-	-	-	90	6	3

5.4.8. Delete From – Deletion of multiple levels

The value "4 = Delete From" for tag "279 = MDUpdateAction" is used instead of sending multiple deletion instructions. It signals the deletion of all levels starting from the level specified in tag "1023 = MDPriceLevel" up to (and including) the max order book level.

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

	Example Instrument					
	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	70	4	1
2	40	7	2	80	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	4 = Delete From
269	MDEntryType	1 = Offer
1023	MDPriceLevel	2
270	MDEntryPx	80
271	MDEntrySize	2
346	NumberOfOrders	1
55	Symbol	Example Instrument

The message above indicates the deletion of all levels starting from 2 to the maximum level, in this case 3 (since we are assuming a max order book depth of 3). This results in the client's Price Depth order book looking as follows:

	Example Instrument					
	Bid			Offer		
Level	**Price**	**Volume**	**No. of Orders**	**Price**	**Volume**	**No. of Orders**
1	50	5	2	70	4	1
2	40	7	2	-	-	-
3	30	4	1	-	-	-

5.4.9. Overlay – Change of a level's price

The value "5 = Overlay" is used for tag "279 = MDUpdateAction" instead of sending two messages with values "2 = Delete" and " 1 = New" respectively to achieve the same result. An overlay instruction may contain changes to volume and/or no. of orders, in addition to price.

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

	Example Instrument					
	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	-	--	

The following message is sent:

Tag		Value
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	2 = Price Depth
279	MDUpdateAction	5 = Overlay
269	MDEntryType	0 = Bid
1023	MDPriceLevel	3
270	MDEntryPx	35
271	MDEntrySize	4
346	NumberOfOrders	1
55	Symbol	Example Instrument

The message above indicates a change of price for level 3. This results in the client's Price Depth order book looking as follows:

	Example Instrument					
	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	*35*	*4*	*1*	-	-	-

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. The scenarios below omit any tags not relevant to the handling of the order book from the messages, for simplicity's sake.

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:

Example Instrument							
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
269	MDEntryType	1 = Offer
290	MDEntryPositionNo	6
270	MDEntryPx	90
271	MDEntrySize	3
37	OrderID	121
55	Symbol	Example Instrument

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:

Example Instrument							
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:

Example Instrument							
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
269	MDEntryType	0 = Bid
290	MDEntryPositionNo	5
270	MDEntryPx	40
271	MDEntrySize	3
37	OrderID	122
55	Symbol	Example Instrument

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:

Example Instrument							
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" is used to signal a change to an order's volume. It is important to 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:

Example Instrument							
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
269	MDEntryType	1 = Offer
290	MDEntryPositionNo	3
270	MDEntryPx	80
271	MDEntrySize	2
37	OrderID	109
55	Symbol	Example Instrument

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:

Example Instrument							
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:

Example Instrument							
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
269	MDEntryType	0 = Bid
290	MDEntryPositionNo	7
270	MDEntryPx	30
271	MDEntrySize	7
37	OrderID	104
55	Symbol	Example Instrument

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:

Example Instrument							
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	-	-	-				

↓

Example Instrument							
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:

Example Instrument							
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
269	MDEntryType	1 = Offer
290	MDEntryPositionNo	4
270	MDEntryPx	90
271	MDEntrySize	4
37	OrderID	103
55	Symbol	Example Instrument

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:

Example Instrument							
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

↓

Example Instrument							
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				

5.5.6. Delete Thru – Deletion of multiple entries

The value "3 = Delete Thru" for tag "279 = MDUpdateAction" is used instead of sending multiple deletion instructions when an order matches multiple levels. It signals the deletion of all entries starting from position 1 up to (and including) the position specified in tag "290 = MDEntryPositionNo"

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

Example Instrument							
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	3 = Delete Thru
269	MDEntryType	1 = Offer
290	MDEntryPositionNo	2
270	MDEntryPx	80
271	MDEntrySize	2
37	OrderID	102
55	Symbol	Example Instrument

The message above indicates the deletion of levels 1 and 2. This shifts all entries below position 2, in this case the entries in positions 3 through 6, up by one level. This results in the client's Order Depth order book looking as follows:

Example Instrument							
Bid				Offer			
Position	Price	Volume	Order ID	Position	Price	Volume	Order ID
1	50	5	105	*1*	-	-	-
2	50	3	112	*2*	-	-	-
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

↓

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

5.5.7. Delete From – Deletion of multiple entries

The value "4 = Delete From" for tag "279 = MDUpdateAction" is used instead of sending multiple deletion instructions. It signals the deletion of all entries starting from the position specified in tag "290 = MDEntryPositionNo" up to (and including) last book entry.

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

Example Instrument							
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	4 = Delete From
269	MDEntryType	0 = Bid
290	MDEntryPositionNo	4
270	MDEntryPx	40
271	MDEntrySize	4
37	OrderID	101
55	Symbol	Example Instrument

The message above indicates the deletion of all entries starting from position 4 up to the last position in the book. This results in the client's Order Depth order book looking as follows:

Example Instrument							
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	-	-	-	4	90	4	103
5	-	-	-	5	90	5	120
6	-	-	-	6	90	3	121

5.5.8. Overlay – Change of an order's price

The value "5 = Overlay" is used for tag "279 = MDUpdateAction" instead of sending two messages with values "2 = Delete" and " 1 = New" respectively to achieve the same result. An overlay instruction may contain changes to volume and/or no. of orders, in addition to price.

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

Example Instrument							
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	-	-	-	4	90	4	103
5	-	-	-	5	90	5	120
6	-	-	-	6	90	3	121

The following message is sent:

Tag		**Value**
35	MsgType	X = MarketDataIncrementalRefresh
1021	MDBookType	3 = Order Depth
279	MDUpdateAction	5 = Overlay
269	MDEntryType	1 = Offer
290	MDEntryPositionNo	1
270	MDEntryPx	60
271	MDEntrySize	4
37	OrderID	110
55	Symbol	Example Instrument

The message above indicates a change of price for the entry in position 1, with an Order ID=101. This results in the client's Price Depth order book looking as follows:

Example Instrument							
Bid				**Offer**			
Position	**Price**	**Volume**	**Order ID**	**Position**	**Price**	**Volume**	**Order ID**
1	50	5	105	*1*	*60*	*4*	*110*
2	50	3	112	2	80	2	102
3	50	2	117	3	80	6	109
4	-	-	-	4	90	4	103
5	-	-	-	5	90	5	120
6	-	-	-	6	90	3	121

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. Appendix

6.1. 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.

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.

6.2. 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.

```
<!--Trading Session Status Message-->
<template id="3" name="TradingSessionStatus">
  <!--Component Start - StandardHeader-->
  <string name="BeginString" id="8">
    <constant value="FIXT.1.1"/>
  </string>
  <int32 name="BodyLength" id="9"/>
  <string name="MsgType" id="35">
    <constant value="h"/>
  </string>
  <string name="SenderCompID" id="49"/>
  <string name="TargetCompID" id="56"/>
  <uInt32 name="MsgSeqNum" id="34"/>
  <string name="SendingTime" id="52"/>
  <uInt32 name="LastMsgSeqNumProcessed" id="369" presence="optional"/>
  <uInt32 name="ATHEXSnapshotIndicator" id="20009" presence="optional"/>
  <uInt32 name="ATHEXFragmentIndicator" id="20010" presence="optional"/>
  <!--Component End - StandardHeader-->
  <string name="SecurityExchange" id="207"/>
  <string name="ATHEXMarketID" id="20001"/>
  <string name="ATHEXBoardID" id="20002"/>
  <string name="TradingSessionID" id="336">
    <constant value="1"/>
  </string>
  <string name="TradingSessionSubID" id="625"/>
  <uInt32 name="TradSesStatus" id="340"/>
  <string name="TransactTime" id="60"/>
  <!--Component Start - StandardTrailer-->
  <string name="Checksum" id="10"/>
  <!--Component End - StandardTrailer-->
</template>
```