IT Development Division

Trading Systems Development Department

ATHEX
Athens Stock Exchange

OASIS MDFS Specification

Version: 2.0

Revision History

Version	Date	Description
1.0	2025/02/24	Multicast release.
2.0	2025/06/16	TCP & APA release. 1. Updated section “4.10. Decoding Example”, field “268 = NoMDEntries” changed from mandatory to optional. 2. Updated sections “1. Introduction” & “2. Architecture Overview” to reflect the new TCP/IP functionality. 3. Added comparison of UDP Multicast and TCP/IP services in section 2. Architecture Overview”. 4. Removed section “3. Connection Procedure & Data Flow”, moved part 5. Added section “3. General Guidelines” which includes parts of the now removed “3. Connection Procedure & Data Flow” section. 6. Added Section “4. TCP/IP Service”. 7. Added Section “5. UDP Multicast Service” which contains parts of the now removed “3. Connection Procedure & Data Flow” section. 8. Added note in section “6. FAST Message Encoding”. 9. Updated section “9.1 Comparison With Legacy IDS Service (IOCP)”. 10. Added section “6.1 Template Versioning”. 11. Added section “6.12 Partial Decoding”. 12. Renamed section “8. Instrument Prices Handling” to “8. Market Data Guidelines”. 13. Added sections “8.3. Bond Volumes” and “8.4. APA OTC Trade Reports”. 14. Updated section “7.1.2. Top of Book/Price Depth Book” for Market/ATO/OTC prices handling. 15. Updated language throughout the document for clarity/uniformity.

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 on the OASIS platform, as well as APA OTC Pre-Trade and Post-Trade reports.

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, fields, tags and tag values from FIX Extension Packs to 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 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. FAST encoding is widely used by institutions serving market data to reduce the data stream size and remove unnecessary overhead, allowing for reduced latency and bandwidth consumption.

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

Throughout this document there are distinct sections for UDP and TCP/IP clients. UDP clients may/should make use of certain TCP/IP features, for this reason most TCP/IP sections are relevant to all clients.

A brief comparison to the legacy IDS Service (IOCP) can be found in Appendix A.

2. Architecture Overview

MDFS offers both UDP multicast and TCP/IP for the dissemination of market data to clients. Each client can opt to utilize either service, according to their specific need. Identical content is available via either protocol and all market data received is interoperable. The main features of each network transfer layer are as follows:

UDP Multicast:

- Lower latency due to less protocol overhead.
- Available via leased line only.
- Higher implementation cost due to need for specialized networking infrastructure.
- More complex networking configuration.
- Guaranteed fairness in transmission.

- Possibility of packet loss, although the MDFS ensures data consistency and availability, through using concurrent Sources (A & B), the Snapshot functionality and the TCP/IP retransmission service.
- Data is sent in FAST encoded format.
- No need to implement the FIX session protocol, unless the TCP/IP retransmission service is utilized.

TCP/IP:

- Lower implementation cost. No need for specialized networking infrastructure.
- Less complex networking configuration.
- More resilient to packet loss, as the protocol handles retransmission of lost packets implicitly.
- Increased latency due to protocol overhead.
- Available via internet or lease lines.
- Data is sent in either FIX or FAST encoded format.
- Need to implement the FIX session protocol.

The following sections will describe the core concepts of the MDFS, as well as each service in depth.

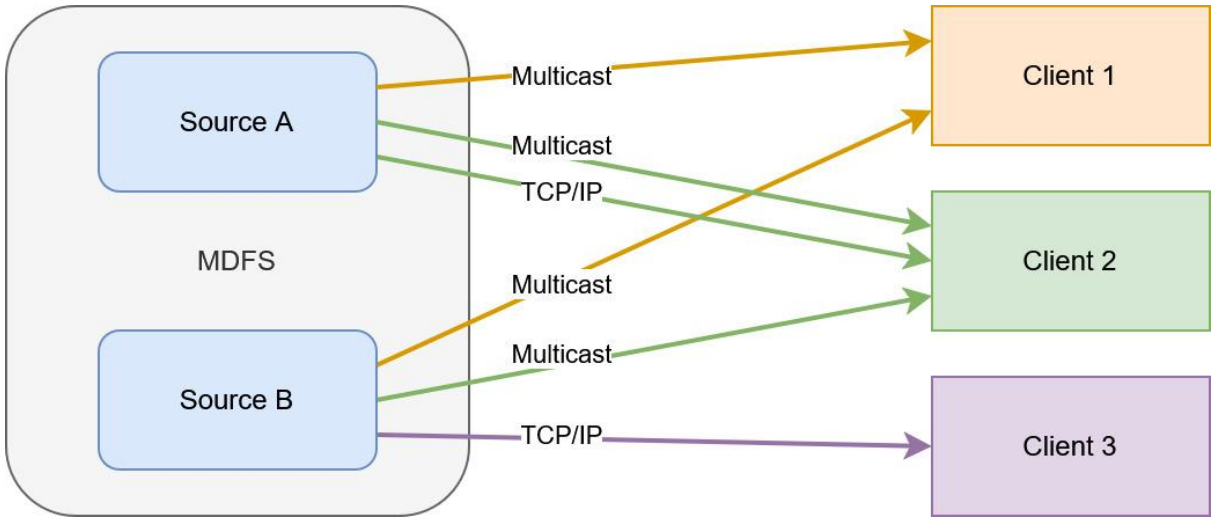

Figure 1 - Architecture Overview

2.1. Incremental Feed Approach

The MDFS follows the paradigm of incremental data feed messages, as outlined by the FIX Trading guideline. This approach relies on an 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. The Snapshot functionality can be utilized to receive the current state with minimal processing, or the Retransmission functionality can be utilized to construct the current state, along with all previous data for the trading session.

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

2.2. Market Data Groups

The MDFS disseminates market data that organized into different groups, with each group receiving messages pertaining to specific Venues, Instrument Types, and message types. Each group has an Incremental feed and a Snapshot feed. The following tables are an example of how these groups are organized:

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

Figure 2 - Incremental Groups

Venue	Instrument Type	Group Type	Venue	Instrument Type	Group Type
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 Groups

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

Instrument Type Grouping	Value of FIX Field "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 groups which do not follow the general structure described in the tables above, the details of which will be made available through other means.

2.3. UDP Multicast Service

Each Market Data Group described in the previous section is served by a multicast group to a specific IP Address & UDP Port combination. All **Incremental** multicast groups are transmitted via the **UDP port 10000**, and all **Snapshot** multicast groups are transmitted via the **UDP port 20000**. Each client connects to multiple feeds that disseminate information relevant to them.

The MDFS replicates all feeds on two identical Sources (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 the possibility of lost packets. Although such events are highly improbable for colocation clients, it is strongly recommended that clients connect to both Sources at all times for redundancy.

Clients connected to the UDP Multicast Service may/should utilize TCP/IP Service functionalities. For this reason, most TCP/IP sections in this document are relevant to all clients.

2.4. TCP/IP Service

The TCP/IP Service provides the following options:

1. Subscription to receive real-time data from a group.
2. Request a snapshot from a group.
3. Request for retransmission of a range of messages from a group.

Clients connected to the UDP Multicast Service can also utilize options 2 & 3 (snapshots and retransmission) for synchronization / recovery reasons.

3. General Guidelines

The following sections cover the general guidelines that should be followed when connecting to either the UDP Multicast Service or the TCP/IP Service. Sections dedicated to the specifics of each service are also included.

3.1. Handling Incremental & Snapshot Traffic

All messages received via Incremental and Snapshot feeds will contain the field “1180 = ApplID” this field 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 field “1180 = ApplID” should be removed, effectively removing “_INCR” or “_SNAP” suffixes.

3.2. Application Sequence Control

The “Application Sequence Control” (ApplSeqCtrl) component is a FIX component (a collection of fields) that appears in all Market Data messages and Heartbeats, after the header component.

It is comprised of two fields:

- **“1180 = ApplID”:** Used to identify each group. Is comprised of the group’s name and the “_INCR” or “_SNAP” suffix (e.g. XATH_CASH_GENERAL_INCR, XATH_CASH_GENERAL_SNAP).
- **“1181 = ApplSeqNum”:** Sequence number per group. Will always be “0” for heartbeats.

These fields are critical for identifying which group the message belongs to and for detecting gaps in that group.

3.3. Heartbeat Messages

MDFS will transmit a heartbeat message for an incremental group if no data has been sent for 30 seconds as a keep-alive mechanism. A client will receive a heartbeat for each incremental group they are receiving data for. Heartbeat messages are not sent for Snapshot groups.

A heartbeat message has field “35 = MsgType” with a value of “0 = Heartbeat” and contains the field “369 = LastMsgSeqNumProcessed”.

The field “369 = LastMsgSeqNumProcessed” contains the value of field “1181 = ApplSeqNum” of the last message sent in that group. This is used for detecting possible gaps in received messages.

A value of “0” in field “369 = LastMsgSeqNumProcessed” indicates that no messages have been sent for that group yet.

3.4. Detecting Gaps

It is crucial for a client to detect any gaps in the data received by MDFS, as all information is disseminated using an incremental approach, thus processing any message without having successfully processed all previous messages will lead to an incorrect state.

For each group (identified by the value of field “1180 = ApplID”) a gap can be detected in the following ways:

- Two consecutive messages (excluding heartbeats, which are covered below) are received for that group with non-contiguous values in field “1181 = ApplSeqNum”.
- A heartbeat message is received for a group with a value in field “369 = LastMsgSeqNumProcessed” that is not contiguous with the value of “1181 = ApplSeqNum” of the last non-heartbeat message received for that group.

If a gap is detected, the client should suspend all processing and initiate one of the available recovery procedures (covered in their respective sections for the TCP/IP Service and the UDP Multicast Service) in order to synchronize with MDFS.

3.5. Snapshot Cycles

Every 1 minute a Snapshot Cycle is generated for each group. A Snapshot Cycle is a collection of messages that contain the current state of the instruments and markets that belong to that group.

For the information contained in a Snapshot Cycle to be valid, the full cycle needs to be processed in sequential order. Messages from two different Snapshot Cycles should not be used to determine the current state of a group.

Each message in a Snapshot Cycle contains the field “20009 = ATHEXSnapshotIndicator”, with possible values of: “0 = Start of cycle”, “1 = End of cycle” and “2 = Start and end of cycle (applies when the cycle is comprised of a single message)”.

Each message in a Snapshot Cycle also contains the field “369 = LastMsgSeqNumProcessed” which indicates which incremental message was the last one sent when this cycle was generated. This relates snapshot to incremental messages, effectively meaning that the cycle contains information up to and including that incremental message.

A Snapshot Cycle is considered “complete” when a message with “20009 = ATHEXSnapshotIndicator” having a value of “0 = Start of cycle” is received and a message having a value of “1 = End of cycle” is received afterwards, or if a single message with value “2 = Start and end of cycle (applies when the cycle is comprised of a single message)” is received. Any messages in between the start and end of the cycle should have contiguous sequence numbers (field “1181 = ApplSeqNum”). If any gap is detected the cycle is unusable and the client need to discard all messages for that cycle and wait to receive the next cycle when it is transmitted.

3.6. Updating the Order Book

As long as the values of field “1181 = ApplSeqNum” in the messages received from the incremental feed are contiguous, the client should keep processing them and applying them to the corresponding order book.

4. TCP/IP Service

This section provides information related to the TCP/IP Service of the MDFS. A client utilizes a **single session for all market data groups and requests** (subscription to Incremental Feeds, Snapshots, Retransmission).

Notes:

- Each MDFS account corresponds to one FIX session with each MDFS Source at its designated port. A client may utilize multiple concurrent sessions if they utilize multiple MDFS accounts. A client may also use the same MDFS account to connect to multiple MDFS Sources concurrently.
- TCP/IP FIX sessions use **SSL** encryption. To establish an **SSL** connection with the MDFS **TLS v1.3** should be used.
- FIX session messages with field "35 = MsgType" having value "4 = SequenceReset"/ 1 = TestRequest" are supported by the MDFS and follow the standard FIX specification. Their functionality will not be covered in this document.
- Standard FIX resend functionality by using session messages with field "35 = MsgType" having value "2 = ResendRequest" is not supported by the MDFS. Instead, upon receiving a valid ResendRequest the MDFS will reply with a sequence reset message ("35 = MsgType" with value "4 = SequenceReset") to perform a gap fill and synchronize with the client. In case of an invalid ResendRequest the MDFS will reply with a rejection message ("35 = MsgType" with value "3 = Reject").

4.1. Logon Procedure

After establishing a TCP/IP connection, an "A = Logon" message must be sent containing the correct credential fields, "553 = Username" and "554 = Password".

If this is the first time a client is connecting to the MDFS, the password will be the default one, and the client will have to update it upon logon using the "925 = NewPassword" field.

Logon attempts may be rejected for the following reasons:

- Provided credentials are incorrect.
- Client has another active TCP/IP connection on the particular MDFS Source (only one connection per account is allowed).
- Client has not changed the default password.
- New password does not fulfill the minimum password requirements.

If a TCP/IP session is opened and a Logon message is not sent within 30 seconds, the MDFS will terminate the connection.

4.2. Updating the Password

When updating a client's password, the change will take place immediately on the MDFS Source it was requested from and will take effect on all other MDFS Sources the next trading day.

If a client wishes to alter passwords on multiple MDFS Sources on the same day, it is important to use the same "925 = NewPassword" on all Sources. Otherwise, the last updated password will be effective on all MDFS Sources the next day.

Minimum Requirements

Passwords must be at least 12 characters long and contain at least one of each: uppercase letters, lowercase letters, numbers, and special characters.

4.3. Sending a Request

After a client is logged in, they can send requests via "BW = ApplicationMessageRequest" messages. Field "1347 = ApplicationRequestType" is used to specify the desired action, with acceptable values being:

- 0 = Retransmission of application messages for the specified Applications
- 1 = Subscription to the specified Applications
- 4 = Unsubscribe to the specified Applications
- 100 = Snapshot for the specified Applications

Each request type is covered in the following sections.

Whenever a client sends a "BW = ApplicationMessageRequest message, they will receive either a "3 = Reject" (for Session-Level validation errors) or a "BX = ApplicationMessageRequestAck" message as a response.

It is the client's responsibility to send unique (for each day) values for field "1346 = ApplReqID", which are used by the exchange to identify Retransmission Requests.

Notes:

- Each "BW = ApplicationMessageRequest" message pertains to a single group. One cannot make requests for multiple groups using a single "BW = ApplicationMessageRequest" message.
- There is no limitation on the number of requests the client can make in a single FIX session, or day.
- The client can request concurrent retransmissions for multiple groups.

4.3.1. Request Acknowledgement

A "BX = ApplicationMessageRequestAck" message will be sent for either successful or rejected application message requests. The possible values for field "1348 = ApplicationResponseType" are:

- "0 = SuccessfullyProcessed"
- "1 = ApplicationNotExist"
- "2 = MessagesNotAvailable"
- "100 = UserNotAuthorized"

In the case of a successful request the "BX = ApplicationMessageRequestAck" message will contain field "1348 = ApplicationResponseType" with the value "0 = SuccessfullyProcessed" and field "58 = Text" confirming the requested action. MDFS will then proceed to perform the requested action.

In case of a rejected request the "BX = ApplicationMessageRequestAck" message will contain field "1348 = ApplicationResponseType" with one of the remaining values which indicate an error, and the value "58 = Text" field will contain a detailed reason specifying why the request was not accepted.

Note that the contents of the "58 = Text" field are subject to change, so clients should not rely on parsing the rejection text for implementing application logic.

Acknowledgement Examples:

Field 1348 = ApplicationResponseType	**Field 58 = Text**
0 = SuccessfullyProcessed	Accepted Retransmission request for Group: [XATH_CASH_GENERAL], Encoding: FIX
1 = ApplicationNotExist	Group [XATH_CASH_GENERAL] does not exist
2 = MessagesNotAvailable	Group [XATH_CASH_GENERAL] is in recovery mode.
2 = MessagesNotAvailable	ApplBegSeqNum<1182> cannot be 0.
2 = MessagesNotAvailable	ApplBegSeqNum<1182> exceeds number of sent messages for group.
2 = MessagesNotAvailable	ApplEndSeqNum<1183> exceeds number of sent messages for group.
2 = MessagesNotAvailable	ApplEndSeqNum<1183> must be equal or greater than ApplBegSeqNum<1182>.
100 = UserNotAuthorized	User does not have permission for Group: [XATH_CASH_GENERAL].
100 = UserNotAuthorized	User does not have permission to subscribe for incremental updates

4.3.2. Request Rejection (Session-Level validation error)

A “3 = Reject” message will be sent for malformed request messages (missing required fields or invalid values).

Example:

An application message request with a missing “1355 = RefApplID” field will receive a “3 = Reject” response message with the following text in field “58 = Text”:

“Bad message. Required field is missing. Field [tag=1355, scope=Repeating Group Instance (numInGroupTag=1351)]. Message [type=BW, seqNum=2, dictionary=MDFS_FIX50SP2].”

Note that the contents of the “58 = Text” field are subject to change, so clients should not rely on parsing the rejection text for any application logic.

4.3.3. Message Encoding

The optional field “20012 = ATHEXMessageEncoding” can be included in “BX = ApplicationMessageRequestAck” messages sent by a client and specifies the encoding of the messages that will be sent out in response to this request. This applies to requests with field “1347 = ApplicationRequestType” having a value of:

- “0 = Retransmission of application messages for the specified Applications”
- “1 = Subscription to the specified Applications”
- “100 = Snapshot for the specified Applications”

It has no effect for requests with value:

- “4 = Unsubscribe to the specified Applications”

The possible values for field “20012 = ATHEXMessageEncoding” are:

- 0 = FAST
- 1 = FIX

If the field is missing from a request, then the value is of “0 = FAST” is implied.

4.4. FAST Encoded Message Encapsulation

If a client requests for messages to be sent using FAST encoding (see section Message Encoding for details), the market data messages send via the TCP/IP Service for that request will have the value “UEFD = EncapsulatedFASTData” in field “35 = MsgType”. These are FIX messages that contain an encapsulated FAST message.

The included field “95 = RawDataLength” contains the number of bytes contained in field “96 = RawData” (the encapsulated FAST message), exactly as it was when it was first transmitted including header fields such as “52 = SendingTime”.

This format allows for FAST encoded messages to be sent via a standard FIX session. Upon receiving such a message, the client must decode the encapsulated FAST message before processing it.

Note: The header field “52 = SendingTime” for messages with field “35 = MsgType” having the value “UEFD = EncapsulatedFASTData” contains the time when the message was sent to a specific client’s FIX session. The encapsulated FAST message, when decoded, contains the actual value of “52 = SendingTime”.

4.5. Subscribe Request

A client may request the transmission of real-time incremental traffic for a specific group, starting from the point of subscription onwards, not including past messages.

This is done by sending a “BW = ApplicationMessageRequest” message with field “1347 = ApplicationRequestType” having a value of “1 = Subscription to the specified Applications”.

The desired group must be specified in field “1355 = RefApplID”. Note that the “_INCR” or “_SNAP” suffixes are redundant and must be omitted in this field.

Optionally, the encoding of the real-time messages sent by the MDFS for this group subscription can be set as described in section Message Encoding.

Note: When a user disconnects from the TCP/IP Service, they will automatically be unsubscribed from all market data groups. Upon reconnecting they will need to re-subscribe to any groups as appropriate.

4.6. Unsubscribe Request

A client may request to stop the transmission of real-time incremental traffic for a specific group to which they were previously subscribed.

This is done by sending a “BW = ApplicationMessageRequest” message with field “1347 = ApplicationRequestType” having a value of “4 = Unsubscribe to the specified Applications”.

The desired group must be specified in field “1355 = RefApplID”. Note that the “_INCR” or “_SNAP” suffixes are redundant and must be omitted in this field.

4.7. Retransmission Request

A client may request the retransmission of messages for a specific group.

This is done by sending a "BW = ApplicationMessageRequest" message with field "1347 = ApplicationRequestType" having a value of "0 = Retransmission of application messages for the specified Applications".

The desired group must be specified in field "1355 = RefApplID". Note that the "_INCR" or "_SNAP" suffixes are redundant and must be omitted in this field.

The range of messages for a request must be specified. The starting point must be provided in field "1182 = ApplBegSeqNum" and the ending point must be provided in field "1183 = ApplEndSeqNum". The values of these fields relate to the values of field "1181 = ApplSeqNum" for that specific group.

The range can be explicit, e.g. [1,1000] or have the ending point be the last available message by setting it to "0", e.g. [1,0] (all ranges are inclusive).

Optionally, the encoding of the messages sent by the MDFS as a result of this retransmission request can be set as described is section Message Encoding.

Note: The field "52 = SendingTime" for FIX messages sent by the MDFS as a result of a retransmission request will contain the timestamp of the original message. For encapsulated FAST encoded messages see section FAST Encoded Message Encapsulation.

4.7.1. Retransmission Request Report

After a retransmission has finished successfully, the client will receive a "BY = ApplicationMessageReport" message which signals the end of the retransmission.

The report includes field "1357 = RefApplLastSeqNum" which contains the value of field "1181 = ApplSeqNum" of the last market data message with this retransmission.

4.8. Snapshot Request

A client may request the transmission of the last available Snapshot cycle for a specific group. A new cycle is generated every 1 minute.

This is done by sending a "BW = ApplicationMessageRequest" message with field "1347 = ApplicationRequestType" having a value of "100 = Snapshot".

The desired group must be specified in field "1355 = RefApplID". Note that the "_INCR" or "_SNAP" suffixes are redundant and must be omitted in this field.

The snapshot messages for the received cycle will contain the field "369 = LastMsgSeqNumProcessed", whose value is equal to the value of field “1181 = ApplSeqNum” of the last available incremental message at the time the cycle was generated (i.e. included in the snapshots). This field is used for synchronization and recovery purposes.

Optionally, the encoding of the messages sent by the MDFS as a result of this snapshot request can be set as described is section Message Encoding.

Note: The field “52 = SendingTime” for FIX messages sent by the MDFS as a result of a snapshot request will contain the timestamp of the original message (which is the time the snapshot was generated). For encapsulated FAST encoded messages see section FAST Encoded Message Encapsulation.

4.8.1. Snapshot Request Report

After the transmission of a snapshot cycle has finished successfully, the client will receive a “BY = ApplicationMessageReport” message which signals the end of the snapshot cycle transmission.

The report includes field “1357 = RefApplLastSeqNum” which contains the value of field “1181 = ApplSeqNum’ of the last market data message sent for this snapshot cycle.

4.9. Disconnecting from the Service

To disconnect from the Service, the client must send a “5 = Logout” message. This message will also be sent from the MDFS when the server shuts down or in case of session errors (e.g. not sending/responding to heartbeats).

To gracefully complete the disconnection procedure a “5 = Logout” message will be sent by the MDFS to acknowledge the client’s request.

Note: When a user disconnects from the TCP/IP Service, they will automatically be unsubscribed from all market data groups, meaning that upon reconnecting they will need to re-subscribe to any groups they want to receive market data for.

4.10. Heartbeat Messages

The MDFS will transmit heartbeat messages for all incremental groups a client is subscribed to, as described in section Heartbeat Messages.

If a client is not subscribed to any incremental group and no message is sent from either side for the duration specified by the client upon logon (field “108 = HeartBtInt”), then a heartbeat message will be sent by the MDFS as a keep-alive mechanism. Heartbeat messages sent for this reason will not contain the field “369 = LastMsgSeqNumProcessed” or the application sequence control component, as they are not associated with any market data group but rather the client’s session.

4.11. Differentiating Between Incremental / Snapshots / Retransmissions

A client connecting to the MDFS TCP/IP Service will be receiving real-time incremental data, snapshots and retransmissions via a single FIX session. It is fundamental for the client to be able to distinguish the respective messages.

- **Snapshots & Incremental / Retransmissions:** it is important to differentiate snapshot traffic from real-time incremental / retransmission traffic, in order to be able to follow the MDFS' Incremental Feed Approach. This can be done by examining the suffix "_INCR" or "_SNAP" in field "1180 = ApplID" as described in the Handling Incremental & Snapshot Traffic section.
- **Incremental & Retransmissions:** due to the utilization of the Application Sequence Control component, there is no need to differentiate between real-time incremental messages and retransmissions as the way they are handled is uniform. Whenever a message is received, regardless of whether it originated from a group subscription or a retransmission, it can only be processed after having completed processing all previous messages. Thus, upon receiving a message that cannot be immediately processed, the client needs to buffer it until it can be processed.

4.12. Initial Connection Procedure

A client connecting to the MDFS via TCP/IP can follow these steps to connect to the data feed and receive real-time information:

Note: As a client may be receiving data related to multiple groups via a single FIX session, it is important to identify which group each message refers to, by utilizing the application sequence control component. Steps 3-6 apply to a single market data group, and as such it is implied that they apply to that specific group, in order to avoid repetition.

1. Download reference data using the RDS service.
2. Connect to the TCP/IP Service and complete the logon procedure.
3. Request to subscribe to the desired group.
4. Determine if all data from the start of the day has been received. This is done by checking if the first message received has field "1181 = ApplSeqNum" with a value of "1" or is a heartbeat with field "369 = LastMsgSeqNumProcessed" having a value equal to "0". If so, then no further action is required so skip to step 7.
5. If the first message received has field "1181 = ApplSeqNum" with a value greater than "1" or is a heartbeat with field "369 = LastMsgSeqNumProcessed" having a value greater than "0", then the client needs to buffer all incoming incremental messages for this group and synchronize with MDFS before proceeding to apply the received messages. This can be done in the following ways:
    a. **via** TCP/IP Snapshot, this method does not include historical data for the day:
        i. Identify the sequence number of the last missing incremental message. This can be done by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat.

ii. Request a snapshot cycle for the group.
iii. Check if the received snapshots include data up to (or exceeding) the last missing incremental message. This is done by checking if the value of field "369 = LastMsgSeqNumProcessed" of the received snapshots is less than the sequence number of the last missing incremental message. If so, request a retransmission with a starting point equal to the next sequence number from one specified by field "369 = LastMsgSeqNumProcessed" of the received snapshots and an ending point equal to the sequence number of the last missing incremental message.
iv. Discard all buffered incremental messages with a sequence number up to and including the value of field "369 = LastMsgSeqNumProcessed" provided in snapshot messages received in this snapshot cycle.
v. Use the information contained in the snapshot cycle as a baseline to sequentially apply the messages received by the retransmission.

b. **via** TCP/IP Retransmission, this method includes historical data for the day:
i. Identify the sequence number of the last missing incremental message. This can be done by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat.
ii. Request a retransmission with a starting point equal to "1" to indicate the first message of the day and the ending point equal to the sequence number of the last missing incremental message.
iii. Apply all incremental messages received via the retransmission in sequential order.

6. Apply all the remaining buffered incremental messages.
7. Keep processing the incoming incremental messages and applying them in real time.
8. Repeat steps 3-6 for each group of interest.

4.13. Recovery Procedure

In the unlikely occasion where a message is not received via the TCP/IP Service then the client should follow the following procedure to perform recovery:

Note: As a client may be receiving data related to multiple groups via the same FIX session, it is important to identify which group each message refers to by utilizing the application sequence control component. Steps 2-5 apply to a single market data group, and as such it is implied that they apply to that specific group, in order to avoid repetition.

1. When a gap in field "1181 = ApplSeqNum" is observed, stop processing and buffer all incoming incremental messages. See section Detecting Gaps for details.
2. Identify the sequence number of the first and last missing incremental messages.
3. The client needs to synchronize with MDFS in order to be able to process any further messages. This can be done in the following ways:
    a. **via** TCP/IP Snapshot, this method discards historical data for the day:
        i. Request a snapshot cycle for the group.
        ii. Check if the received snapshots include data up to (or exceeding) the last missing incremental message. This is done by checking if the value of field "369 = LastMsgSeqNumProcessed" of the received snapshots is less than the sequence number of the last missing incremental message. If so, request a retransmission with a starting point equal to the next sequence number from one specified by field "369 = LastMsgSeqNumProcessed" of the received snapshots and an ending point equal to the sequence number of the last missing incremental message.
        iii. Discard all buffered incremental messages with a sequence number less or equal than the value of field "369 = LastMsgSeqNumProcessed" included in snapshot messages received in this snapshot cycle.
        iv. Clear any past state and use the information contained in the snapshot cycle as a base to apply the messages received by the retransmission.
    b. **via** TCP/IP Retransmission, this method retains any historical data for the day (recommended method):
        i. Request a retransmission with a starting point equal to the sequence number of the first missing incremental message and an ending point equal to the sequence number of the last missing incremental message.
        ii. Apply all incremental messages received via the retransmission in sequential order.
4. Apply all the remaining buffered incremental messages.
5. Resume processing the incoming incremental messages and applying them in real time.

4.14. TCP/IP Service Examples

The following sections contain examples of messages received via the TCP/IP Service that showcase the different types of traffic a client may receive. Clients need to be able to process data they receive for multiple market data groups and traffic types as noted in section TCP/IP Service.

4.14.1. Initial Connection Procedure using TCP/IP Snapshot

The following example showcases the typical connection procedure for a client utilizing the TCP/IP Snapshot functionality.

MDFS		Client	Notes
	←	**Logon**	
Logon	→		Acknowledgement
	←	**ApplicationMessageRequest** ApplReqType = Subscribe	
ApplicationMessageRequestAck	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 102	→		The client needs to request a snapshot cycle. This, and all further incremental messages for this group, must be buffered by the client for later processing.
	←	**ApplicationMessageRequest** ApplReqType = Snapshot	
ApplicationMessageRequestAck	→		
Snapshot Message ATHEXSnapshotIndicator = 0 ApplID = XATH_CASH_ORDERS_SNAP ApplSeqNum = 2000 LastMsgSeqNumProcessed = 90	→		Start of the snapshot cycle.
…			
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 103	→		Client receives a real-time message in the middle of a snapshot cycle, and must buffer it for later processing.
Snapshot Message ATHEXSnapshotIndicator = 1 ApplID = XATH_CASH_ORDERS_ SNAP ApplSeqNum = 2100 LastMsgSeqNumProcessed = 90	→		End of the snapshot cycle. The client needs to process all messages received in this cycle, then the client needs to request a retransmission for messages 91 to 101.
ApplicationMessageReport	→		
	←	**ApplicationMessageRequest** ApplReqType = Retransmission ApplBegSeqNum = 91 ApplEndSeqNum = 101	
ApplicationMessageRequestAck	→		
Retransmitted Incremental Message	→		

ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 91			
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 92	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 104	→		Synchronization not complete yet. This must be buffered by the client
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 93	→		
…			
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 101	→		All requested messages have been retransmitted. After the client has processed them, they can process the buffered messages and resume processing incoming real-time messages.
ApplicationMessageReport	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 105	→		Client can process this message on reception and continue normally.
…			

Figure 6 - Initial Connection Procedure using TCP/IP Snapshot

4.14.2. Initial Connection Procedure using TCP/IP Retransmission

In the following example showcases the typical connection procedure for a client utilizing the TCP/IP Retransmission functionality.

MDFS		Client	Notes
	←	**Logon**	
Logon	→		Acknowledgement
	←	**ApplicationMessageRequest** ApplReqType = Subscribe	
ApplicationMessageRequestAck	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 102	→		The client needs to request a retransmission of messages 1 to 101.
	←	**ApplicationMessageRequest** ApplReqType = Retransmission ApplBegSeqNum = 1 ApplEndSeqNum = 101	
ApplicationMessageRequestAck	→		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 1	→		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 2	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 103	→		Client receives a real-time message in the middle of a retransmission, so they must buffer it for later processing.
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 3	→		
…			
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 101	→		All requested messages have been retransmitted. After the client has processed them, they can process the buffered messages and resume processing incoming real-time messages.
ApplicationMessageReport	→		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 104	→		Client can process this message on reception and continue normally.
…			

Figure 7 - Initial Connection Procedure using TCP/IP Retransmission

4.14.3. Different Heartbeat Types

This example shows the difference between session heartbeats and market data group heartbeats.

MDFS		Client	Notes
	←	**Logon** HeartBtInt = 60	
Logon	→		Acknowledgement
…			No traffic on either direction for 60 seconds.
Heartbeat	→		Session heartbeat. Does not include the Application Sequence Control component or the field LastMsgSeqNumProcessed.
	←	**ApplicationMessageRequest** ApplReqType = Subscribe RefApplID = XATH_CASH_ORDERS	
ApplicationMessageRequestAck	→		
	←	**ApplicationMessageRequest** ApplReqType = Subscribe RefApplID = XATH_CASH_GENERAL	
ApplicationMessageRequestAck	→		
Real-time Incremental Message ApplID = XATH_CASH_ GENERAL _INCR ApplSeqNum = 1	→		As a result of this message being sent, no heartbeat will be sent for XATH_CASH_ GENERAL at this point.
…			No traffic is produced for 30 seconds for group XATH_CASH_ORDERS.
Heartbeat LastMsgSeqNumProcessed = 0 ApplID = XATH_CASH_ GENERAL _INCR ApplSeqNum = 0	→		Group heartbeat. The value 0 of LastMsgSeqNumProcessed indicates that no messages have been sent for this group.
…			

Figure 8 - Different Heartbeat Types

4.14.4. Multiple Market Data Groups via a Single FIX Session

In the following example a client receives interleaved real-time incremental traffic for multiple Market Data groups and must be able to process the messages for each group independently, by examining the Application Sequence Control component.

34 = MsgSeqNum	1180 = ApplID	1181 = ApplSeqNum	Type
57	XATH_CASH_GENERAL_INCR	100	Incremental
58	XATH_CASH_ORDERS_INCR	457	Incremental
59	XATH_CASH_GENERAL_INCR	101	Incremental
60	XATH_CASH_ORDERS_INCR	458	Incremental
61	XATH_CASH_ORDERS_INCR	459	Incremental

Figure 9 - Multiple Market Data Groups via a Single FIX Session

4.14.5. Multiple Traffic Types via a Single FIX Session

In the following example a client receives interleaved incremental, snapshot and retransmission traffic and must be able to differentiate between the traffic types and process them accordingly.

34 = MsgSeqNum	1180 = ApplID	1181 = ApplSeqNum	Type
5298	XATH_CASH_GENERAL_SNAP	32890	Snapshot
5299	XATH_CASH_GENERAL_INCR	1	Retransmission
5300	XATH_CASH_GENERAL_SNAP	32891	Snapshot
5301	XATH_CASH_GENERAL_INCR	2	Retransmission
5302	XATH_CASH_GENERAL_INCR	12005	Incremental
5303	XATH_CASH_GENERAL_INCR	3	Retransmission
5304	XATH_CASH_GENERAL_INCR	12006	Incremental

Figure 10 - Multiple Traffic Types via a Single FIX Session

5. UDP Multicast Service

This section provides information related to the UDP Multicast Service of the MDFS.

5.1. Handling Data Feeds on Sources A & B

As aforementioned the MDFS replicates all feeds on two identical Sources (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 Sources in order to handle any such incidents non-disruptively (without resorting to recovery).

In a typical scenario the client (assuming they are connected to both Source A & B) should, for each message with a distinct value in field "1181 = ApplSeqNum", keep the message received first from either Source and discard the subsequent copy they receive from the other Source.

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

Order	Field 1181 = ApplSeqNum	
	Source A	Source B
1	100	
2		100
3	101	
4		101
5		102
6	102	
7	103	
8		103

Figure 11 - Sources A & B Example

5.2. Handling Gaps in Message Sequence Numbers

The client should always check the field "1181 = ApplSeqNum" for gaps in the message sequence of any UDP multicast feed they are connected to. In the case of a gap in the sequence numbers in either of the two Sources the client should receive the message through the other Source (assuming they are connected to both Source A & B).

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

Order	Field 1181 = ApplSeqNum	
	Source A	Source B
1	100	
2		100
3	101	
4		101
5		102
6	103	
7		103

Figure 12 - Handling Message Sequence Gaps

In the example above the message with value "102" in field "1181 = ApplSeqNum" was not received through Source A, but was received through Source B. In this case the client should have no interruption of data flow as they can utilize the message received from Source B.

5.3. Differentiating Between Incremental / Snapshots / Retransmissions

It is important for a client connecting to the MDFS UDP Multicast Service may to know when there is a need to differentiate between these different types of data and how to do it.

- **Snapshots & Incremental / Retransmissions:** it is important to differentiate snapshot traffic from real-time incremental / retransmission traffic, in order to be able to follow the MDFS' Incremental Feed Approach.
  This is easily done for the UDP Multicast Service as 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.** Alternatively, this can be done by examining the suffix "_INCR" or "_SNAP" in field "1180 = ApplID" as described in the Handling Incremental & Snapshot Traffic section.
- **Incremental & Retransmissions:** Since real-time incremental data are served by UDP multicast while retransmissions are served via the TCP/IP service, no further logic is required.

  Moreover, due to the utilization of the Application Sequence Control component, there is no need to differentiate between real-time incremental messages and retransmissions as the way they are handled

is uniform. Whenever a message is received, regardless of whether it originated from an incremental multicast group or a TCP/IP retransmission, it can only be processed after having completed processing all past messages. Therefore, when receiving messages that cannot be immediately processed, the client needs to buffer these messages until processing is possible.

5.4. Initial Connection Procedure

A client connecting to the MDFS via UDP multicast can follow these steps to connect to the data feed and receive real-time information:

Note: As a client may be receiving data related to multiple groups via multicast, it is important to identify which group each message refers to by utilizing the application sequence control component. Steps 2-5 apply to a single market data group, and as such it is implied that they apply to that specific group, in order to avoid repetition.

1. Download reference data using the RDS service.
2. Start listening to the Incremental feed.
3. Determine if all data from the start of the day has been received. This is done by checking if the first message received has field "1181 = ApplSeqNum" with a value of "1" or is a heartbeat with field "369 = LastMsgSeqNumProcessed" having a value equal to "0". If so, then no further action is required so skip to step 6.
4. If the first message received has field "1181 = ApplSeqNum" with a value greater than "1" or is a heartbeat with field "369 = LastMsgSeqNumProcessed" having a value greater than "0", then the client needs to buffer all incoming incremental messages for this group and synchronize with MDFS in order to be able to apply the received messages. This can be done in the following ways:
    a. **via UDP multicast Snapshot**, this method does not include historical data for the day:
        i. Start listening to the Snapshot Feed. Discard all snapshot 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 with.

        Note: in the unlikely event where a snapshot cycle is received where the value of "369 = LastMsgSeqNumProcessed" is less than the sequence number of the last missing incremental message (identified by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat), then the client should discard that snapshot cycle and repeat this step until a snapshot cycle containing information up to and including the last missing message (see Snapshot Cycles for details) is received.
        ii. Discard all buffered incremental messages with a sequence number up to and including the value of field "369 = LastMsgSeqNumProcessed" provided in snapshot messages received in this snapshot cycle.
        iii. Use the information contained in the snapshot cycle as a baseline to sequentially apply the received incremental messages on.
    b. **via TCP/IP Snapshot**, this method does not include historical data for the day:

i. Identify the sequence number of the last missing incremental message. This can be done by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat.
ii. If not already connected, connect to the TCP/IP Service and complete the logon procedure, then request a snapshot cycle for the group.
iii. Check if the received snapshots include data up to (or exceeding) the last missing incremental message. This is done by checking if the value of field "369 = LastMsgSeqNumProcessed" of the received snapshots is less than the sequence number of the last missing incremental message. If so, request a retransmission with a starting point equal to the next sequence number from one specified by field "369 = LastMsgSeqNumProcessed" of the received snapshots and an ending point equal to the sequence number of the last missing incremental message.
iv. Discard all buffered incremental messages with a sequence number up to and including the value of field "369 = LastMsgSeqNumProcessed" provided in snapshot messages received in this snapshot cycle.
v. Use the information contained in the snapshot cycle as a baseline to sequentially apply the messages received by the retransmission.

c. **via TCP/IP Retransmission**, this method includes historical data for the day:
i. Identify the sequence number of the last missing incremental message. This can be done by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat.
ii. If not already connected, connect to the TCP/IP Service and complete the logon procedure, then request a retransmission with a starting point equal to "1" to indicate the first message of the day, and the ending point equal to the sequence number of the last missing incremental message.
iii. Apply all incremental messages received via the retransmission in sequential order.

5. Apply all the remaining buffered incremental messages.
6. Keep processing the incoming incremental messages and applying them in real time.
7. Repeat steps 2-5 for each group of interest.

Note: for details regarding the TCP/IP Service, please see section TCP/IP Service.

5.5. Recovery Procedure

In the unlikely occasion where a message is not available through either Source A or B then the client should follow the following procedure to perform recovery:

Note: As a client may be receiving data related to multiple groups via the same FIX session, it is important to identify which group each message refers to by utilizing the application sequence control component. Steps 2-4 apply to a single market data group, and as such it is implied that they apply to that specific group, in order to avoid repetition.

1. When a gap in field "1181 = ApplSeqNum" is observed, stop processing and buffer all incoming incremental messages. See section Detecting Gaps for details.
2. Identify the sequence number of first and last missing incremental messages. The last missing message can be identified by subtracting 1 from the value of field "1181 = ApplSeqNum" of the first received incremental message, or the value of "369 = LastMsgSeqNumProcessed" if the first message received is a heartbeat.
3. The client needs to synchronize with MDFS in order to be able to process any further messages. This can be done in the following ways:
    a. **via UDP multicast Snapshot**, this method discards any historical data for the day:
        i. Start listening to the Snapshot Feed. Discard all snapshot 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.
        Note: in the unlikely event where a snapshot cycle is received where the value of "369 = LastMsgSeqNumProcessed" of the received snapshots is less than the sequence number of the last missing message, then the client should discard that snapshot cycle and repeat this step until a snapshot cycle containing information up to and including the last missing message (see Snapshot Cycles for details) is received.
        ii. Disconnect from the Snapshot Feed. Once you have received a full snapshot cycle you will have all the information needed to synchronize with the accompanying incremental stream.
        iii. Discard all buffered incremental messages with a sequence number less or equal than the value of field "369 = LastMsgSeqNumProcessed" included in snapshot messages received in this snapshot cycle.
        iv. Clear any past state and use the information contained in the snapshot cycle as a base to apply future incremental messages on.
    b. **via TCP/IP Snapshot**, this method discards historical data for the day:
        i. If not already connected, connect to the TCP/IP Service and complete the logon procedure, then request a snapshot cycle for the group.
        ii. Check if the received snapshots include data up to (or exceeding) the last missing incremental message. This is done by checking if the value of field "369 = LastMsgSeqNumProcessed" of the received snapshots is less than the sequence number of the last missing incremental message. If so, request a retransmission with a starting point equal to the next sequence number from one specified by field "369 =

LastMsgSeqNumProcessed" of the received snapshots and an ending point equal to the sequence number of the last missing incremental message.

iii. Discard all buffered incremental messages with a sequence number less or equal than the value of field "369 = LastMsgSeqNumProcessed" included in snapshot messages received in this snapshot cycle.

iv. Clear any past state and use the information contained in the snapshot cycle as a base to apply the messages received by the retransmission.

c. **via TCP/IP Retransmission**, this method retains any historical data for the day (recommended method):

i. If not already connected, connect to the TCP/IP Service and complete the logon procedure, then request a retransmission with a starting point equal to the sequence number of the first missing incremental message and an ending point equal to the sequence number of the last missing incremental message.

ii. Apply all incremental messages received via the retransmission in sequential order.

4. Apply all the remaining buffered incremental messages.
5. Resume processing the incoming incremental messages and applying them in real time.

Note: for details regarding the TCP/IP Service, please see section TCP/IP Service.

5.6. Multicast Service Examples

5.6.1. Initial Connection Procedure using UDP Multicast Snapshot

In the following example showcases the typical connection procedure for a client utilizing the UDP Multicast Snapshot functionality.

MDFS Messages		Client Messages	Notes
…			Client joins multicast group XATH_CASH_ORDERS_INCR.
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 102	→ MCAST		The client needs to receive a full snapshot cycle. This, and all further incremental messages for this group, must be buffered by the client for later processing.
…			
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 110	→ MCAST		
…			Client joins multicast group XATH_CASH_ORDERS_SNAP.
Snapshot Message ApplID = XATH_CASH_ORDERS_SNAP ApplSeqNum = 2000 LastMsgSeqNumProcessed = 110	→ MCAST		Start of the snapshot cycle.
…			
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 111	→ MCAST		Client receives a real-time message in the middle of a snapshot cycle, so they must buffer it for later processing.
Snapshot Message ApplID = XATH_CASH_ORDERS_ SNAP ApplSeqNum = 2100 LastMsgSeqNumProcessed = 110	→ MCAST		End of the snapshot cycle. The client needs to process all messages received in this cycle. After the client has processed them, they should discard all real-time incremental messages with ApplSeqNum less than or equal to 110, then they can process the remaining buffered messages and resume processing incoming real-time messages.
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 112	→ MCAST		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 113	→ MCAST		
…			

Figure 13 - Initial Connection Procedure using UDP Multicast Snapshot

5.6.2. Initial Connection Procedure using TCP/IP Snapshot

In the following example showcases the typical connection procedure for a client utilizing the TCP/IP Snapshot functionality.

MDFS Messages		Client Messages	Notes
…			Client joins multicast group XATH_CASH_ORDERS_INCR.
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 102	→ MCAST		The client needs to request a snapshot cycle. This, and all further incremental messages for this group, must be buffered by the client for later processing.
	← TCP	**ApplicationMessageRequest** ApplReqType = Snapshot	
ApplicationMessageRequestAck	→ TCP		
Snapshot Message ApplID = XATH_CASH_ORDERS_SNAP ApplSeqNum = 2000 LastMsgSeqNumProcessed = 90	→ TCP		Start of the snapshot cycle.
…			
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 103	→ MCAST		Client receives a real-time message in the middle of a snapshot cycle, so they must buffer it for later processing.
Snapshot Message ApplID = XATH_CASH_ORDERS_ SNAP ApplSeqNum = 2100 LastMsgSeqNumProcessed = 90	→ TCP		End of the snapshot cycle. The client needs to process all messages received in this cycle, then the client needs to request a retransmission for messages 91 to 101.
ApplicationMessageReport	→ TCP		
	← TCP	**ApplicationMessageRequest** ApplReqType = Retransmission ApplBegSeqNum = 91 ApplEndSeqNum = 101	
ApplicationMessageRequestAck	→ TCP		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 91	→ TCP		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 92	→ TCP		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 104	→ MCAST		The client needs to buffer this message.
Retransmitted Incremental Message	→		

ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 93	TCP		
…			
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 101	→ TCP		All requested messages have been retransmitted. After the client has processed them, they can process the buffered messages and resume processing incoming real-time messages.
ApplicationMessageReport	→ TCP		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 105	→ MCAST		The client can process this message immediately and continue normal operation.
…			

Figure 14 - Initial Connection Procedure using TCP/IP Snapshot

5.6.3. Initial Connection Procedure using TCP/IP Retransmission

In the following example showcases the typical connection procedure for a client utilizing the TCP/IP Retransmission functionality.

MDFS Messages		Client Messages	Notes
…			Client joins multicast group XATH_CASH_ORDERS_INCR.
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 102	→ MCAST		The client needs to request a retransmission of messages 1 to 101.
	← TCP	**ApplicationMessageRequest** ApplReqType = Retransmission ApplBegSeqNum = 1 ApplEndSeqNum = 101	
ApplicationMessageRequestAck	→ TCP		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 1	→ TCP		
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 2	→ TCP		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 103	→ MCAST		Client receives a real-time message in the middle of a retransmission, so they must buffer it for later processing.
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 3	→ TCP		
…			
Retransmitted Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 101	→ TCP		All requested messages have been retransmitted. After the client has processed them, they can process the buffered messages and resume processing incoming real-time messages.
ApplicationMessageReport	→ TCP		
Real-time Incremental Message ApplID = XATH_CASH_ORDERS_INCR ApplSeqNum = 104	→ MCAST		
…			

Figure 15 - Initial Connection Procedure using TCP/IP Retransmission

6. FAST Message Encoding

The FAST Protocol 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.

Note: While the MDFS is designed to work with FAST 1.2, currently no features of the 1.2 revision are utilized for performance reasons. Thus, the **MDFS is currently backwards compatible with FAST 1.1**, but this is subject to change in the future if any FAST 1.2 features are utilized.

6.1. Template Versioning

Each version of MDFS is accompanied by a corresponding FAST templates XML file. The format of all FAST encoded FIX messages sent by the MDFS is described in this document containing templates for each message type. A sample of the FAST template XML format can be found in the appendix.

Each message type used by the MDFS is described by a <template> element in the XML file. Each <template> element has an "id" attribute that is a unique number, and a "name" attribute that is a unique string. The "name" attribute includes the template's "id" as a suffix.

In each revision of the templates XML file there are up to two <template> elements for each message type with different "id" attributes, one describes the latest version of the message and the other describes the previous version (if applicable). This is done to facilitate seamless transition from one version of the MDFS to the next, as the client may start utilizing the latest templates XML file before the latest version of the MDFS is released, as the XML will contain <template> elements for both the old and new version of the message.

The client can determine which <template> corresponds to the latest version of a message by looking at the "id" attribute. The value of the "id" attribute always increments and as such the latest version of a message is the one with the largest value in its "id" attribute.

For example:

Templates_v10.xml includes:

```
...
<!--CURRENT - Trading Session Status Message-->
<template name="TradingSessionStatus_100" id="100">
...
<!--DEPRECATED - Trading Session Status Message-->
<template name="TradingSessionStatus_65" id="65">
...
```

in this case the template with id="100" should be used when the new version of MDFS is released, and the template with id="65" is used by the previous MDFS version.

Templates_v11.xml includes:

```
...
<!--CURRENT - Trading Session Status Message-->
<template name="TradingSessionStatus_120" id="120">
...
...
<!--DEPRECATED - Trading Session Status Message-->
<template name="TradingSessionStatus_100" id="100">
...
```

in this case the template with id="120" should be used when the new version of MDFS is released, and the template with id="100" now represents the template used by the previous MDFS version.

6.2. Packet Structure

The following table is a representation of a FAST Packet:

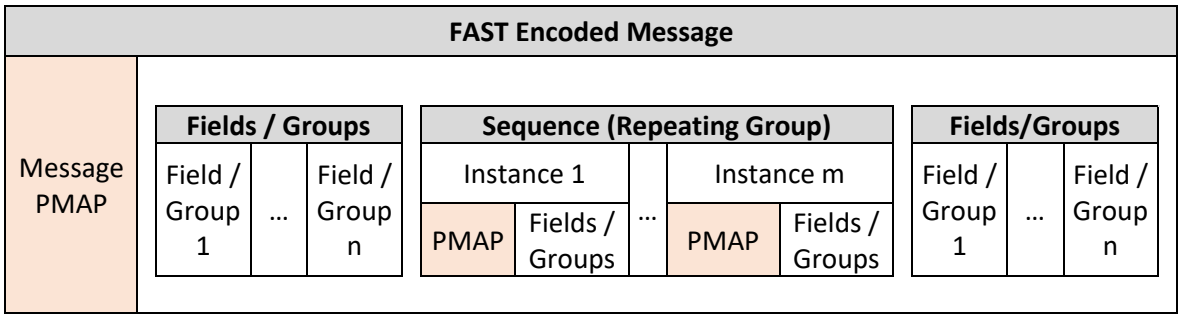

Figure 16 - FAST Packet Structure

Where:

- **Field:** A FAST-encoded FIX field.
- **Group:** A group of FAST-encoded FIX fields, 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.

6.3. 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

6.4. 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 the appendix.

6.5. 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.

6.6. 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.

6.7. 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.

6.8. 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.

6.9. 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.

6.10. 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.

6.11. 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 the appendix.

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									
#	**Field**	**Attributes / Operators**	**Type**	**Presence**	**PMAP Bit Required**	**PMAP bit**	**Encoded Value**	**Stop Bit Decoded Value**	**Value**
	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	Optional	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 17 - 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.

6.12. Partial Decoding

If latency is of critical importance, a client can perform a partial decoding of the FAST message in order to decide whether to discard a message prior to decoding it.

This can be very useful when receiving multicast traffic via both Source A and B, by quickly extracting the "1181 = ApplSeqNum" field from the FAST message and determining whether this is a packet has already been received from the other source.

Because "1181 = ApplSeqNum" field is in the header and no optional fields are before it, a client can advance the decoder state until it reaches the N-th stop-bit position where the "1181 = ApplSeqNum" is located and decode the stop-bit encoded value.

Currently the first FAST Fields encountered are: Global PMap, Template ID, SenderCompID, TargetCompID, MsgSeqNum, ApplID, ApplSeqNum.

Each of the above FAST fields take up one stop-bit encoded value, thus the "1181 = ApplSeqNum" field is at the 7th stop-bit encoded value.

7. 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 field “269=MDEntryType” having value “0 = Bid”, “1 = Offer” or “J = Empty book”. As such any other repeating group types, such as those with field “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 field “60 = TransactTime” between the Oder Depth & Top of Book/Price Depth books as these are handled independently, i.e. the value of field “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 fields 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 fields.

7.1. Market/Stop/ATO/ATC orders

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

7.1.1. Order Depth Book

Market/Stop (value “1=Market”/” = Stop” in field “40 = OrdType”) and ATO/ATC orders (value “2 = At the Opening (OPG)”/”7 = At the Close” in field “59 = TimeInForce”) do not have a set price (thus do not contain the field “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 field “269 = MDEntryType” and are ordered by their release timestamp in the matching engine.

7.1.2. 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 field “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, as well as during the closing price trading phase (ATC). These repeating group entries do not contain the field “270 = MDEntryPx”.

7.2. Empty Book

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

Example Message:

Field		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.

7.3. 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.

7.3.1. 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:

Field		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

7.3.2. 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:

Field		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

7.3.3. 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:

Field		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	-	-	-

7.4. 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 (field "264 = MarketDepth" = 3) for simplicity's sake, but the same concepts apply for any depth.

7.4.1. 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:

Field		**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

7.4.2. 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:

Field		**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

7.4.3. 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:

Field		**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

7.4.4. 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:

Field		**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	-	-	-

7.4.5. 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:

Field		**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	-	-	-

7.4.6. 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:

Field		**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

7.5. 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.

7.5.1. 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:

Field		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*

7.5.2. 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:

Field		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				

7.5.3. Change – Change of an entry's volume

The value "1 = Change" for field "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:

Field		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				

7.5.4. 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:

Field		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

7.5.5. 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:

Field		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				

7.6. 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 field "35 = MsgType" contains the value "W = MarketDataSnapshotFullRefresh".
- The field "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.

8. Market Data Guidelines

This section contains useful information related to the handling of various types of market data received via Incremental Refresh messages.

8.1. Handling 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. An "f = SecurityStatus" message with field "625=TradingSessionSubID" having value "102 = Pre-Call (Auction)" will be sent.
2. For the duration of the auction, "X = MarketDataIncrementalRefresh" messages with one repeating group with field "269=MDEntryType" having value "v = Projected Auction Price" will be sent that contain a projection of the auction price. These messages have field "279=MDUpdateAction" having value "0 = New" for the first message and "1 = Change" for all subsequent updates.
3. Once the auction concludes, an "f = SecurityStatus" message with field "625=TradingSessionSubID" having value "2 = Opening (Auction Price is calculated)" will be sent.
4. Subsequently, an "X = MarketDataIncrementalRefresh" message with two repeating groups will be sent:
    a. The first repeating group with field "269=MDEntryType" having value "v = Projected Auction Price" and field "279=MDUpdateAction" having value "2 = Delete" will be sent to signify that the auction phase has ended and therefore the projected price should be discarded.
    b. The second repeating group with field "269=MDEntryType" having value "w = Auction Price" and field "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 field "269=MDEntryType" having value "w = Auction Price" and field "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.

8.2. Handling 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 (pre-call) phase. An "f = SecurityStatus" message with field "625=TradingSessionSubID" having value "102 = Pre-Call (Auction)" will be sent.
2. For the duration of the auction, "X = MarketDataIncrementalRefresh" messages with one repeating group with field "269=MDEntryType" having value " u = Projected Closing Price" will be sent that contain a projection of the closing price. These messages have field "279=MDUpdateAction" having value "0 = New" for the first message and "1 = Change" for all subsequent updates.
3. Once the closing auction concludes, an "f = SecurityStatus" message with field "625=TradingSessionSubID" having value "2 = Opening (Auction Price is calculated)" will be sent.
4. Subsequently, an "X = MarketDataIncrementalRefresh" message two repeating groups will be sent:
    a. The first repeating group with field "269=MDEntryType" having value " u = Projected Closing Price" and field "279=MDUpdateAction" having value "2 = Delete" will be sent to signify that the auction phase has ended and therefore the projected closing price should be discarded.
    b. The second repeating group with field "269=MDEntryType" having value " 5 = Closing price" and field "279=MDUpdateAction" having value "0 = New" will be sent containing the actual closing price.

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

8.3. Bond Volumes

All volume/size fields transmitted by the MDFS for bond instruments (field "20011 = ATHEXSecurityCategory" having the value "5 = Bond" contain the "raw" volume/size, i.e. it is not pre-multiplied by the bond's Nominal Value/Contract Size.

If a client needs these volumes/sizes to be multiplied by the bond's Nominal Value/Contract Size this must be applied by the client, by utilizing the "231 = ContractMultiplier" field included in "Start of Day Price" messages for bonds in the appropriate "General" type groups. The client may multiple any transmitted volume/size by the value of this field in order to get the desired format.

8.4. APA OTC Trade Reports

The MDFS transmits APA OTC pre-trade and post-trade reports submitted to the exchange in specialized Groups.

Note that all incremental APA OTC messages transmitted via these groups will always have the field "279 = MDUpdateAction" with the value "0 = New", even when the message contains an amended or cancelled trade

report. The field “20015 = ATHEXAPAReportStatus” included in these messages must be used instead in order to determine the status (New, Amend, Cancel) of a trade report.

This is done because APA amendments/cancellations may be submitted to the exchange for trade reports that were not initially submitted on the same day, and would therefore not be available via the MDFS that day. This results in inability to use the field “279 = MDUpdateAction” to indicate amendments/cancellations as transmitting a trade report with field “279 = MDUpdateAction” having with value “1 = Change”/”2 = Delete” without having transmitted the original trade report with “279 = MDUpdateAction” with the value “0 = New” first, would break the semantics of the FIX protocol’s “279 = MDUpdateAction” field.

9. Appendix A

9.1. 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 offers both TCP/IP and UDP multicast as an option for clients, which can utilize each protocol in accordance with their needs.

The MDFS' UDP multicast service 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 Sources, the snapshot recovery mechanism and the TCP/IP retransmission service.

The MDFS' TCP/IP service can be utilized by clients that favor lower implementation costs, simpler networking infrastructure and the option to access the service over the internet.

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.

9.2. 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.

```
<template name="ExampleMessage_34" id="34">
  <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>
```