DEVELOPERS GUIDE

HKEx Orion Market Data Platform
Derivatives Market Datafeed Products

Version 1.0
2 August 2013

DOCUMENT HISTORY

Distribution Version

Version	Date of Issue	Comments
V1.0	2 August 2013	First Distribution Issue

CONTENTS

1 INTRODUCTION

This document contains guidelines and suggestions for HKEx Orion Market Data Platform - Derivatives Market ("OMD-D") feed handler developers. All information included in this document is presented for reference only. Clients should design and implement their own OMD-D feed handler that is tailored to their business and technical requirements.

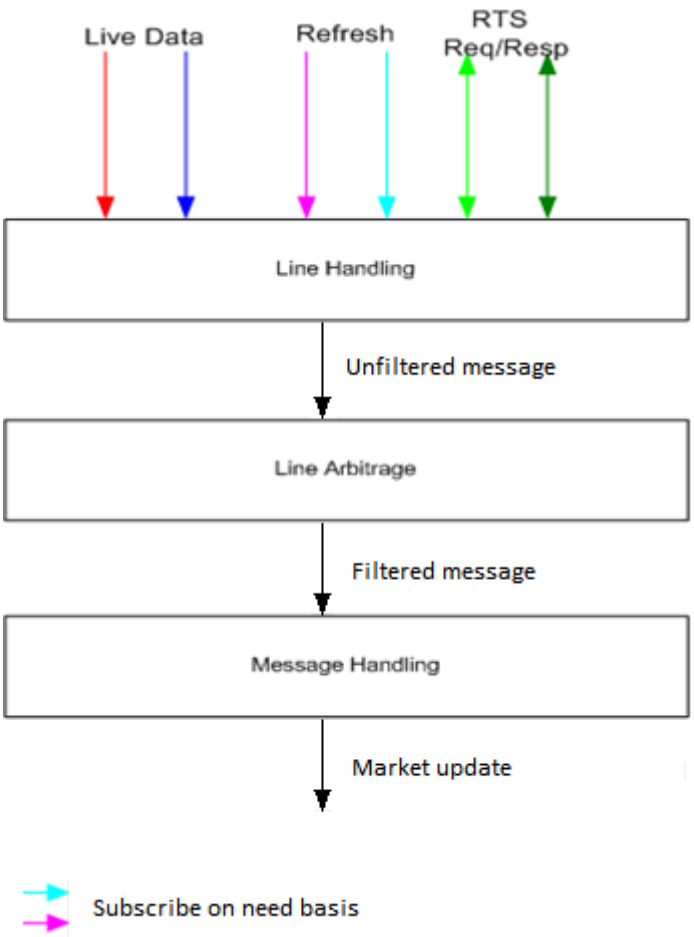
The scope of this document covers line arbitrage, packet and message processing, retransmission and refresh mechanisms, order book maintenance and exception handling.

The purpose of this document is to answer questions that developers may have after reading the OMD-D interface specification. It shows examples of usage and code snippets to help developers understand the logic behind the market data disseminated from OMD-D.

Table 1. Acronyms used in this document

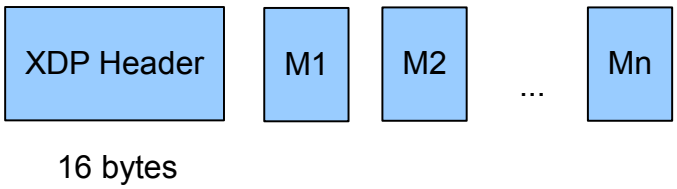
FH	Feed handler
HA	High Availability
MC	Multicast
RFS	Refresh Server
RTS	Retransmission Server
UDP	User Datagram Protocol
XDP	Exchange Data Publisher

Diagram 1. A Basic Client Application Layout



2 DATA STRUCTURE

Multicast messages are structured into a common packet header followed by zero or more messages. Messages within a packet are laid out sequentially, one after another without any spaces between them.



A packet only contains complete messages. In other words, a single message will never be fragmented across packets.

2.1 Packet Header

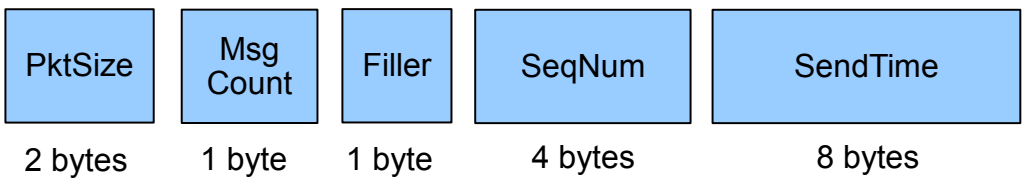
All packets disseminated from the OMD-D feed have a common packet header. This format is consistent across live, retransmission and refresh. An XDP packet consists of a 16-byte header followed by messages.

There are no delimiters between the packet header and messages or between messages themselves. One has to use the size of the header and in each individual message to determine the start of each message.

Table 2 below shows the packet header structure. The offsets in the table represent the number of bytes away from the beginning of the packet.

Table 2. Packet Header

Field	Offset	Length	Format	Description
PktSize	0	2	Uint16	Binary integer representing size of the packet (including this header)
MsgCount	2	1	UInt8	Binary integer representing number of messages included in the packet
Filler	3	1	String	
SeqNum	4	4	Uint32	Binary integer representing the sequence number of the first message in the packet
SendTime	8	8	UInt64	Binary integer representing the number of nanoseconds since January 1, 1970, 00:00:00 GMT, precision is provided to the nearest millisecond



2.2 Heartbeats

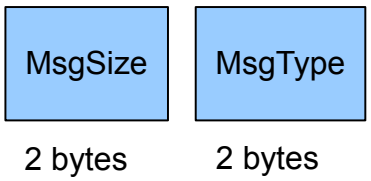
Heartbeats consist of a packet header with MsgCount set to 0 and do not increment the sequence number of the multicast channel. SeqNum in the packet header of Heartbeats is set to the sequence number of the previous message sent on the channel.

The Heartbeat message syntax is identical across OMD-D services.

2.3 Message Header

The format of each message within a packet varies according to message type. However, regardless of the message type, all messages start with a two-byte message size followed by a two-byte message type.

MsgSize Binary integer representing the length of the message (including the header)
MsgType Binary integer representing the type of message. Please refer to the HKEx OMD-D Interface Specification for the full list of message type and the layout of each message type



2.4 Message Formats

Please refer to the HKEx OMD-D Interface Specification for details on the following message types:

- Control messages
- Retransmission
- Refresh
- Reference Data
- Status Data
- Order Book Data
- Trade and Price Data
- News
- Clearing Information

3 ENDIAN

All binary values are in Little Endian byte order, which means the first byte (lowest address) is the least significant one.

In C/C++, one solution is to create a structure containing all the fields from the packet header and cast the pointer to a packet, to a pointer to such a structure. For instance:

```
struct XdpPacketHeader
{
        unsigned short mPktSize;
        unsigned char mMsgCount;
        unsigned char mFiller;
        unsigned long mSeqNum;
        unsigned long long mSendTime;
```

```
};
```

Assume the packet is passed as a pointer to const unsigned char, which could look like this:

```
struct PacketHeader* hdr = static_cast<PacketHeader*> (packetPtr);
```

One packet can contain multiple messages. Clients should locate the beginning of each message based on the message length and process each message separately. The number of messages within a packet is indicated by MsgCount field in the packet header.

4 FIELD ATTRIBUTES

4.1 Null Values

From time to time certain fields cannot be populated and specific values are used to represent null. This is currently used within Int32 fields of the Aggregate Order Book Update (353) message, the Calculated Opening Price (364) message as well as the Add (330) / Modify (331)/ Delete (332) Order messages.

The Int32 null representation is 0x80000000 (Hex 2's complement).

4.2 Currency Values

See the ISO-4217 Currency Codes for a full list of possible data values. Currently, the system uses the following codes:

'HKD' – Hong Kong dollars
'USD' – US dollars
'CNY' – Chinese Renminbi

HKEx may add or delete currency code(s), whenever applicable, in the future.

5 MESSAGE PROCESSING

Each multicast channel maintains its own session. A session is limited to one business day. During the day, message sequence number is strictly increasing and therefore unique within a channel.

5.1 Start of Day

OMD-D will normally be brought up around 5:00am. This start up time, however, is not rigid and HKEx has the right to adjust this time according to the different trading situations.

Clients start at OMD-D startup time

- Clients start listening to real-time multicast channels (Each OMD-D data product is delivered via a group of real-time multicast channels)
- OMD-D sends Sequence Reset message (100). Please refer to section *9.4 Sequence Reset Message* for processing details.
- OMD-D sends Reference Data messages below
  - Commodity Definition (301)
  - Class Definition (302)
  - Series Definition Base (303)
  - Series Definition Extended (304)
  - Combination Definition (305)

- Remark:
  - Clients may receive multiple Sequence Reset messages during the start of day. The general handling should be reset the next expected sequence number and clear all cached data for all instruments. Please refer to section *9.5 OMD-D Restarts Before Market Open* for details.
  - After receiving the Sequence Reset message, clients should also check the sequence number of next incoming packet. If the sequence number is not equal to 1, it indicates that there is packet loss. Please refer to section *9.4 Sequence Reset Message* for details.

Clients start after OMD-D startup time and miss sequence reset message and reference data

- Please refer to section *5.3.2.2 Processing a Refresh* for exception handling of late connection

5.2 Normal Transaction

Normal message transmission is expected between when the market opens for trading and when the market is closed. Heartbeats are sent regularly (currently OMD-D sets to every 2 seconds) on each channel when there is no line activity.

UDP multicast network/transport protocol is used in OMD-D and data is sent to different broadcast streams (known as multicast channels).

UDP is not a reliable transport protocol. So packets may be lost or come out of order. The data on each channel comes from two redundant lines, A and B, to minimize the risk of losing a packet.

Clients receive and process OMD-D data

- Receive real-time multicast messages from Line A and Line B
  - Create two sockets using Multicast IP / Ports of Line A and Line B
  - Read data from multicast channel for Line A and Line B

- Line Arbitration using sequence number in packet header
  - Discard duplicate packets
  - Reorder packets
  - Detect message gap

- Process multicast messages
  - Process Data Message
  - Process Control Message (Heartbeat)

5.2.1 Receive Multicast

Clients join particular multicast group in order to receive the desired data. Data is categorized and available from dedicated multicast groups.

Clients connect and receive real-time multicast messages from Line A and Line B.

Please refer to *APPENDIX A – Pseudo code to connect and receive multicast channel* for example on connecting multicast channels.

5.2.2 Line Arbitration

The network/transport protocol used in OMD-D is UDP multicast. The data in OMD-D is divided into broadcast streams (known as channels). The data on each channel comes from two redundant lines, A and B. UDP is not a reliable transport protocol like TCP but because of this it is much faster, although this means it is possible that packets may be lost or come out of order. Two lines with identical data minimize the risk of losing a packet; however the risk still exists.

Note

1. Clients should not prioritize line A over line B. They should listen to both line A and B at the same time. Line A is not guaranteed to be faster than B. They should both be treated with the same priority. The approach that assumes listening to line B only if there is a gap detected on line A is incorrect. In general, it is recommended to have an abstraction layer between the gap detection module and the source of packets. In other words, the gap detection module does not have to know where the packets are coming from, it just needs to monitor packet sequence numbers.
2. The packaging of messages between Line A and Line B may be different. In the example below, three packets are sent on each line, but message 'OrderUpdate3' appears in one packet from Line A but in the subsequent packet on Line B.

Diagram 2. Normal Message Delivery of Primary and Secondary Line (Line A and B)

Primary		
Messages	**MC**	**SN**
OrderUpdate1 OrderUpdate2 OrderUpdate3	3	101
Trade1 OrderUpdate4	2	104
Trade2 Statistics1	2	106

Secondary		
SN	**MC**	**Messages**
101	2	OrderUpdate1 OrderUpdate2
103	3	OrderUpdate3 Trade1 OrderUpdate4
106	2	Trade2 Statistics1

Note
- MC: Message Count in a Packet

Clients receiving OMD-D feed are recommended to implement the following functionality in order to provide appropriate line handling:

1. Discarding duplicate messages
2. Reordering messages
3. Gap Detection

All of the above can be achieved by remembering the next expected sequence number. Please refer to the Gap Detection Diagram in the OMD-D Interface Specification for reference. Basically, a gap detection mechanism may work like this:

When clients receive a packet from Line A or Line B,

- Handle the first packet, process each message within the packet and advance the next expected sequence number (nextSeqNum) by 1

- When subsequence packet is received, compare the current seqNum in the packet header with the nextSeqNum
  - If seqNum > nextSeqNum, it is a gap and spool the message
  - If (seqNum + msgCount in packet) < nextSeqNum, it is a duplicate packet and skip

- When processing each message within the packet
  - If (seqNum + message processed count in this packet) < nextSeqNum,
    It is a duplicate message and skip
  - If (seqNum + message processed count in this packet) = nextSeqNum,
    Process it and advance the next expected sequence number (nextSeqNum) by 1

Please refer to *APPENDIX B – Pseudo code of Line Arbitration* for example on detecting gap or duplicate packet.

Possible approaches for handling message gap

Approach 1: Clients wait some time to fill the gap from the redundant line (or the packet may come from the same line, possibly out of order)

If a given amount of time has passed and there still is a gap, the clients should send a retransmission request. While awaiting for retransmission all packets coming from the live feed should be spooled. After processing the retransmitted packets, clients should process the spooled packets/messages.

Note

1. While waiting for the retransmission, another gap can occur. Clients should take this into account. One possible solution would be to keep track of how many gaps have been detected and for which gaps a retransmission request has already been sent.
2. Only a continuous series of packets/messages from the spool should be processed.
3. Any gaps should await to be filled either from the redundant line or the retransmission server.
4. Check if the gap in spooled messages can be filled at regular interval.
5. If the gap cannot be recovered for specified time, clients should recover from refresh server.

Please refer to *APPENDIX **B** – Pseudo code of Line Arbitration* for example on processing spooled messages.

Approach 2: Issue a retransmission request immediately after detecting a gap

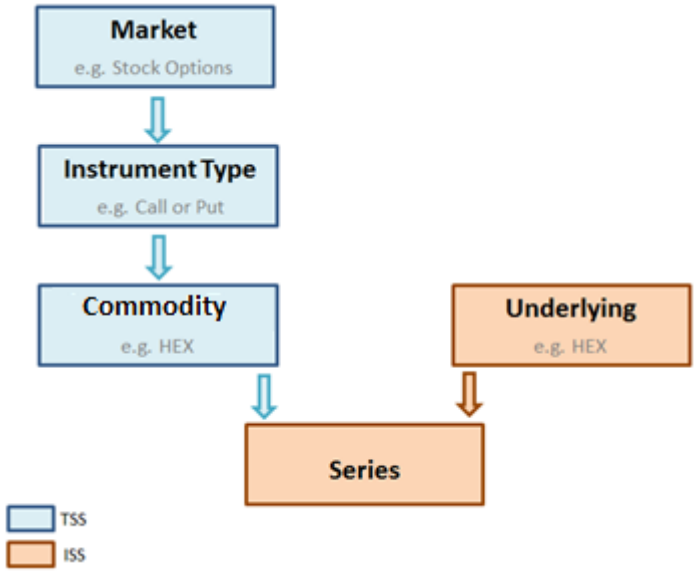
If the missed packets/messages come on the redundant line before they come from the retransmission, clients will simply process them and discard the retransmitted ones.

5.2.3 Process Data Message

5.2.3.1 Market Status update arrangement

The Market Status (320) message provides the trading status of derivatives products at 5 levels, namely, Market, Instrument Type, Instrument Class, Instrument Series (or simply Series) and Underlying, which is specified in the field “State Level”. For Clients who need to present the status of an instrument, this section provides essential information for Clients’ programs to arrive at the current trading status (also known as the Active Instrument State) of each individual series.

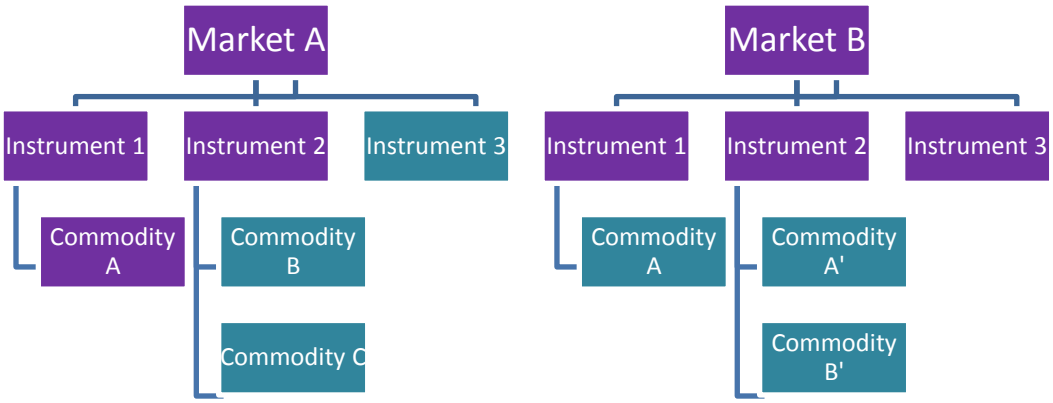
The diagram below illustrates the hierarchy structure of derivatives products. The Trading Session State (TSS) is a state applicable to 3 levels – Market, Instrument Type and Instrument Class (or Commodity) whereas the Instrument Session State (ISS) is applicable to 2 levels – Underlying and Series.



The current trading status of a series, also known as its Active Instrument State (AIS), is determined by two state parameters namely Trading Session State (TSS) and Instrument Session State (ISS) and their priorities set at that point of time.

Trading Session Status – (State Levels 1, 2 & 3)

The hierarchy of TSS is generally predefined at the start of the day and stays unchanged for the day. The diagram below presents an example of such predefined hierarchy; assuming only two Markets (Market A and Market B) exist.



a. At start of day, clients can obtain the predefined hierarchy of TSS via the Market Status (320) messages in OMD-D in real time and refresh channels.
b. In the above hierarchy diagram, the purple block represents the predefined hierarchy and therefore Market Status (320) is sent for purple block only. Please note that
   - There is always a dedicated Market Status (320) for any update of trading status under purple block. E.g. If Market Status (320) is received for "MarketA + Instrument1" level, the trading status under "MarketA + Instrument1 + CommodityA" should **not** inherit the update from "MarketA + Instrument1" as it will have its own dedicated Market Status (320) for its own status update.

- If a series under an instrument class or instrument type is not included in the predefined hierarchy, the series should inherit the status from its next upper level which is within the predefined hierarchy. E.g. Instrument series under "MarketA + Instrument3" inherit the trading status of "MarketA". No dedicated Market Status (320) will be sent for "MarketA + Instrument3".

c. All TSS are always declared at the start of business day. No new TSS for undeclared block is created during day time (i.e. non-purple block).

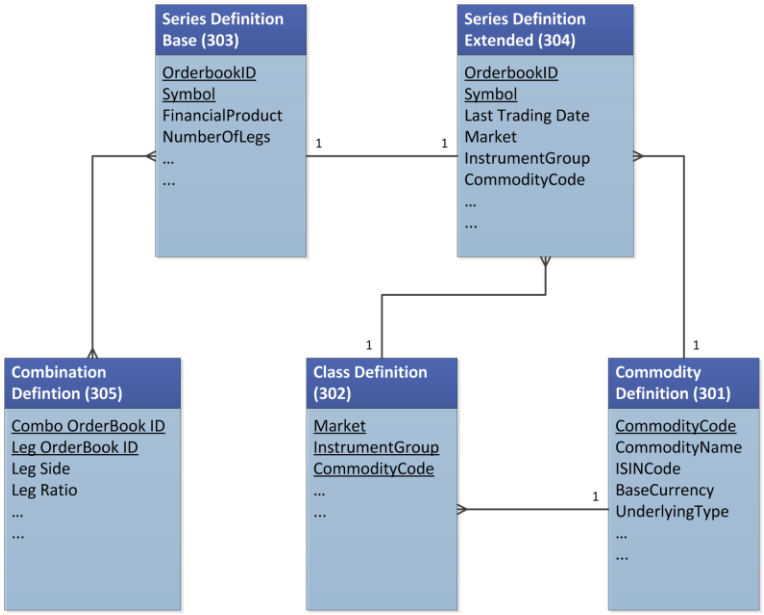
Instrument Session State – (State Level 4 & 5)

a. ISS is determined by the Series or Underlying levels. Series level means the individual Instrument series. Underlying level means the series under same CommodityCode as declared in Commodity Definition (301).
b. Refer to the above diagram, "MarketA + Instrument1 + CommodityA" and "MarketB + Instrument1 + CommodityA" are under the same Underlying level (i.e. CommodityCode A). The Underlying Level ISS for CommodityCode A should be applied to all Instrument series under both "MarketA + Instrument1 + CommodityA" and "MarketB + Instrument1+ CommodityA".
c. ISS is not necessarily being sent on a business day.

Active Instrument State

Whenever there is a Market Status (320) message received, comparison of the priority is required for all the instrument series in affected levels. If the priority of ISS is higher than the priority of TSS, then AIS is the ISS, else the AIS is the TSS.

5.2.3.2 Building up Definitions



Below are the remarks needed to be noted when building the relationship of Definitions.

a. Tradable Series
- All tradable series, including suspended series, of the current business day are provided under Series Definition Extended (304) message at start of day. Intra-day created series are also declared in Series Definition Extended (304) message at real time. In Series Definition Extended (304) message, "Series Status" field declares the (suspend/resume) status of the series in HKATS.

b. Suspended Series
- No Series Definition Base (303) is sent if the series is suspended in market before market open. At start of business day, if the series is suspended, no Series Definition Base (303) is

sent for the series, while Series Definition Extended (304) is available. When the series is resumed for trading in market, Series Definition Base (303) of the resumed series is broadcast immediately before the first quotation or trade is done in market. (I.e. Series Definition Extended (304) is a superset of the Series Definition Base (303) message.)

5.2.3.3 Full Order Book management

Unlike OMD-C, Trade (350) message in OMD-D has implicit logic to remove orders in order book. No Modify Order (331) message is sent when order is partially traded. Also, no Delete Order (332) is sent when the order is fully traded. The tradable quantity of orders in market should be reduced by the traded quantity as declared in each Trade (350) message. Orders in market should be implicitly removed when quantity reached zero. For Detail, please refer to *8 FULL ORDER BOOK MANAGEMENT*

5.2.3.4 Partitions in HKATS

In the derivatives market trading system (known as "HKATS"), multiple markets are defined for trading different futures and options products. Markets are grouped in two partitions. Please see below assignment.

Partition Number	Markets ID
1	1 – 20
2	21 - 255

Note: The above partitioning is subject to change by market operation development.

5.2.3.5 Traded quantity of a deal

Trade Ticker is a deal committed by one single matching which involves multiple bid and ask orders and results in multiple Trade (350) messages. The key "Trade ID" of each Trade (320) message is identical under the same Trade Ticker.

Trade ID is the unique key of a Trade Ticker for a series. For calendar spread series trading, Combo Group ID cannot be used to associate the Trade ID of different series as a single deal. A deal may create multiple Trade (350) messages, and Combo Group ID may be declared in different values on the outright series under a combo trade.

In a Trade (350) message, if "Printable" is set OFF in "Deal Type", the Quantity of the trade should not be included in the total turnover of a business day, otherwise it would be double counted.

During auction period before the market opens, all trades under the same COP prices will be declared under a single Trade Ticker, and therefore have the same Trade ID.

5.2.3.6 Trade Amendment

Unlike Trade (350) message which is for individual trades, Trade Amendment (356) message is always provided at Trade Ticker level.

When there is a trade amendment in market, OMD-D broadcasts the Trade Amendment (356) message for the cancellation of the Trade Ticker, and then OMD-D broadcasts another Trade Amendment (356) message for rectification to declare the new volume of the Trade Ticker. If the Trader Ticker is cancelled, no Trade Amendment (356) message will be sent for declaring the new volume of the Trade Ticker.

Upon receiving a Trade Amendment (356) message for cancellation, a client's application should treat all trades with the same Trade ID in the Trade Amendment (356) message as cancelled. Then later on if a Trade Amendment (356) message with the same Trade ID for rectification arrives, it can

be added as a new trade with the new volume. Unlike the processing of a Trade (350) message, no order book updates would be required when processing a Trade Amendment (356) message.

5.2.3.7 Calculated Opening Price reset when PREOPEN market end

During pre-market opening session, Calculated Opening Price (364) message is sent for instruments for which pre-market opening is available. At the end of the pre-market opening session after completion of matching, "CalculatedOpeningPrice" value will be reset to N/A.

For Derivatives Fulltick (DF) clients, it should be noted that Add Order (330), Modify Order (331) and Delete Order (332) messages are not transmitted during the pre-market opening session. At the end of the pre-market opening session, OrderBook Clear (335) message is sent to signal that clients should clear out the order book in the memory to get ready for the transmission of all outstanding orders via the Add Order (330) messages. Please note that the OrderBook Clear (355) message and market status are sent in separate channels. Clients may experience different message inbound order due to race condition.

5.2.3.8 Next Day Tradable Series

OMD-D disseminates the next day tradable series after market close. For those next day tradable series, "EffectiveTomorrow" field is defined as 'True' in Series Definition Extended (304) message. Please note that no OrderBookID is provided in Series Definition Extended (304) message for those next day tradable series, and no calendar spread series is covered in next day tradable series declaration.

5.2.3.9 Intra-Day created Series

OMD-D disseminates the intra-day created series at real time once the series is created in HKATS. Tailor-Made Combo Series and Flexible Options are examples of intra-day created series.

When a new series is created in market, Series Definition Base (303) and Series Definition Extended (304) are sent for the definition of the series. If the series is a calendar spread series, Combination Definition (305) will also be sent.

Clients should process the Definition Messages immediately and prepare for the orders and trades to be broadcast on the newly created series.

5.2.3.10 Message routing for SOM and non-SOM

In OMD-D, messages are classified into SOM (Stock Option market) and Non-SOM (Not Stock Option Market) for distribution via separate channels. The message types in the table below are not classifiable and therefore will be routed to both SOM and Non-SOM channels.

OMD-D message types	Both channels	Remarks
Commodity Definition (301)	Yes	
Series Definition Base (303)	Yes	*For combo series, both channel will receive the same information.*
Market Status (320)	Yes	
Commodity Status (322)	Yes	
Market Alert (323)	Yes	
Estimated Average Settlement Price (365)	Yes	

5.2.3.11 After Hours Trading (AHT) – Clarification on Trading Information

Information	OMD-D Message	Notes
Open Interest (OI) (both Gross and Net OI)	Open Interest (366)	When issued at start of day, the latest OI ▪ Day Indicator = 0, i.e. for Previous Trading Day ▪ For products tradable in after-hour trading (AHT) session, it is the OI after the AHT session ▪ For products not tradable in AHT session, it is the same OI as provided around end of day of previous day in general
		When issued around the end of day, ▪ Day Indicator = 1, i.e. for Current Trading Day ▪ The OI as of the end of the current day, not covering the transactions in the AHT session ▪ Same OI as published in the Daily Market Report on the HKEx website in general
Settlement Price	Open Interest (366)	▪ Settlement price as of the end of the regular trading session ▪ Settlement price is determined once only at the end of a settlement cycle which includes the AHT session of the previous day and the regular trading session of the current day
Trade/Series Statistics – Day High – Day Low – Cumulative Volume, etc.	Trade Statistics (360) Series Statistics (363)	Trade/Series Statistics will be reset to zero before AHT starts and then again after the end of AHT for the following futures and options products: ▪ Products tradable in AHT including Hang Seng Index (HSI) Futures and H-share Index (HHI) Futures ▪ HSI Options, HHI Options and Mini H-share Index (MCH) Futures Trade/Series statistics will not be reset until the next day for all other products

5.2.3.12 Expiration Date field in Series Definition Extended (304) Message

A bit pattern is used in the field. The seven most significant bits are used for year, the next four for month and the five least significant bits for day. All these bits make up an unsigned 16 bits field. The year-field starts counting from 1990. Thus, 1990=1, 1991=2 ... 2001=12.

Example: January 1, 1990: Binary: 0000001 0001 00001 year month day 7 bits 4 bits 5 bits
Decimal: 545

5.2.4 Process Control Message (Heartbeats)

Heartbeats are disseminated at regular time intervals. Clients can use heartbeats to check if the feed is alive. If there is no heartbeat for longer than a configurable time (please refer to OMD Interface Specification Section 2.2.2 for the multicast heartbeat interval & Section 4.3 for the unicast heartbeat interval currently set in OMD), then it indicates an outage of data transmission.

Note that OMD-D sends heartbeats only when there is no market data being disseminated. When there is market data on the line, no heartbeats will be sent.

Heartbeats consist of a packet header with MsgCount set to 0 and do not increment the sequence number of the multicast channel. SeqNum in packet header is set to the sequence number of the previous message sent in the channel.

When receiving heartbeat packet, clients should ignore this packet in gap detection. Otherwise, clients may fail to detect the actual message gap.

Table 3. Gap Detection Example

Time	Packet sent from OMD-D	Packet received by Client	Remark
T1	101	101	
T2	102	102	
T3	103		Packet with SeqNum 103 is lost
T4	103 (Heartbeat)	103 (Heartbeat)	If client receives heartbeat message but cannot find the corresponding packet with same sequence number, it should be a message gap and client should recover the lost message
T5	104	104	
T6	105	105	
T7	106	106	

5.3 Recovery

Since UDP multicast is not a reliable protocol, there is a risk of packet lost. Clients can recover lost messages using the retransmission server or the refresh service with consideration on various factors such as message gap size, recovery time/event and etc.

5.3.1 Retransmission Service

For small message gaps, clients can recover lost messages using the retransmission server (RTS) which is connected to clients by the more reliable TCP/IP protocol. In order to receive lost messages, clients need to send a Retransmission Request. The RTS will respond with a Retransmission Response which can indicate that either the request has been accepted or rejected (the RetransStatus field). If accepted, the RetransStatus field will be 0, and if rejected, the values can be 1, 2, 100 or 101.

The retransmission server contains only a relatively small number of messages from each broadcast channel and covers the market activities for the last 15 to 30 seconds under normal market conditions. The RTS should not be thought of as a means of full data recovery. It serves only as real time retransmission of a relatively small number of lost messages.

Clients can have only one connection with the RTS.

The sequence number range as well as the number of requests per day is limited to 1000 requests, and 10,000 messages per request.

Note

If clients need to issue a retransmission request for a gap bigger than the allowed limit, they need to split the requests into appropriate amount of smaller requests.

RTS Logon, RTS Logon Response, RTS Request and RTS Response message will begin with packet header which is same format as real time. Clients should ignore the sequence number in RTS packet header when sending or processing the RTS messages.

5.3.1.1 Secondary Retransmission Server

There is a secondary RTS which would be used in case of problems encountered with the primary RTS. This is a part of the High Availability design and is meant to provide customers with a seamless service in case of the primary RTS failure.

5.3.1.2 RTS Logon

In order to receive retransmission, clients must establish a TCP/IP connection with the RTS and initiate a session by sending a Logon message within the logon timeout interval (5 seconds). If clients do not send a Logon message within the logon timeout interval, the server will close the connection.

Table 4. Logon Packet Header

PktSize	32
MsgCount	1
Filler	
SeqNum	Not used
SendTime	The number of nanoseconds since January 1, 1970, 00:00:00 GMT, precision is provided to the nearest millisecond.

Table 5. Logon Request Message

MsgSize	16
MsgType	101
Username	Username to logon in plain text

5.3.1.3 RTS Logon Response

The RTS immediately sends a LogonResponse message after it receives a Logon request. The SessionStatus field indicates if the Logon was successful. The possible values of this field are:

Table 6. Logon Session Statuses

Logon Session Status	Meaning
0	Session Active
5	Invalid Username
100	User already connected

The session, once established, can be reused for sending any subsequent retransmission requests. To maintain the session, a client must respond to heartbeats sent by the RTS within 5 seconds.

5.3.1.4 RTS Heartbeats

To determine the healthiness of the client connection on the TCP/IP channel, the RTS will regularly send heartbeats to the client. The heartbeat frequency is 30 seconds. The client must respond with a Heartbeat Response. The timeout of this heartbeat response is set at 5 seconds. If no response is received by the RTS within this timeframe, RTS will disconnect the session.

A Heartbeat Response is an exact copy of the incoming Heartbeat.

5.3.1.5 RTS Request

A Retransmission Request consists of a Packet Header and a Retransmission Request (201).

Table 7. Retransmission Request Packet Header

PktSize	32
MsgCount	1
Filler	
SeqNum	Not used
SendTime	The number of *nanoseconds* since *January 1, 1970, 00:00:00 GMT*, precision is provided to the nearest millisecond.

Table 8. Retransmission Request Message

MsgSize	16
MsgType	201
ChannelID	Depending on the broadcast stream
Filler	
BeginSeqNum	Message sequence number of first message in range to be resent
EndSeqNum	Message sequence number of last message in range to be resent

Example of Retransmission Request

Assume client application received following packets from real time multicast channel 1

Channel	Packet Sequence number	Message	Message received	Message Gap (Y/N)
1	101	Msg1 Msg2 Msg3	Msg 1 (101) Msg 2 (102) Msg 3 (103)	N
1	104	Msg4 Msg5 Msg6	Msg 4 (104) Msg 5 (105) Msg 6 (106)	N
1	109	Msg7 Msg8	Msg 7 (109) Msg 8 (110)	Y (Missing messages with sequence number 107-108)

Client application should send following Retransmission Request Message to recover missing messages (Seq # 107-108)

MsgSize	16
MsgType	201
ChannelID	1
Filler	
BeginSeqNum	107
EndSeqNum	108

5.3.1.6 RTS Response

After sending a Retransmission Request, the RTS will respond with a Retransmission Response message. The most important field in the response message is the RetransStatus. Below are the possible values and what they indicate:

Table 9. Retransmission Response statuses

Retransmission Response Status	Meaning
0	Request accepted
1	Unknown/Unauthorised Channel ID
2	Messages not available
100	Exceeds maximum sequence range
101	Exceeds maximum requests in a day

Note

It is very important to stop sending retransmission requests for the current day after being rejected with reason 101. Client may contact HKEx OMD-D help desk for assistance.

5.3.1.7 RTS Message

Upon receiving Retransmission Response with Status '0', the RTS will start sending packets containing the requested messages. The sequence number of first requested message will be used as sequence number in packet header.

5.3.1.8 RTS Limits

Below is a table detailing the limits imposed on the Retransmission Service:

Table 10. Retransmission System Limits

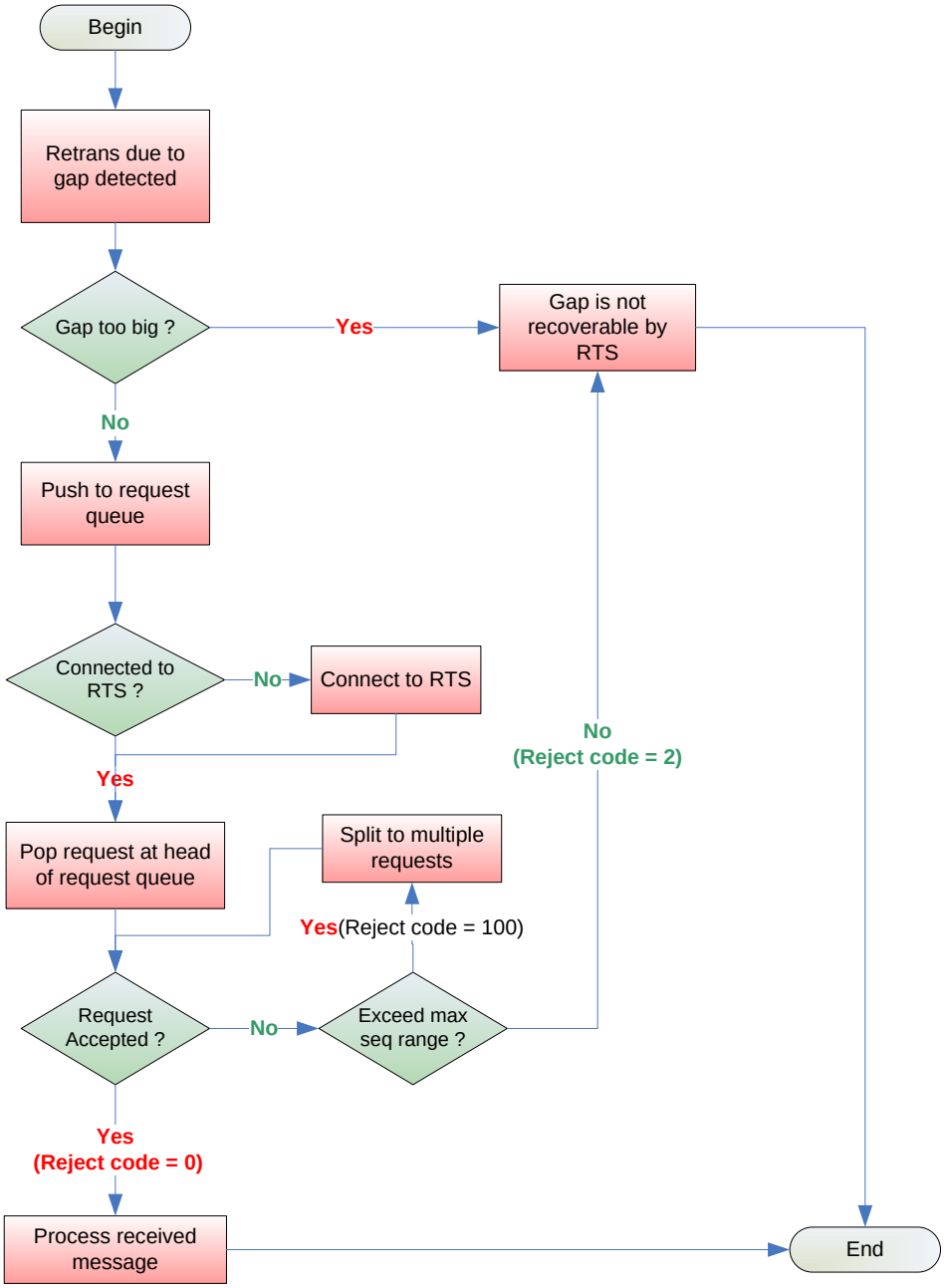
System Limit	Value
Last number of messages available per channel ID	Market activities of the last 15 to 30 seconds
Maximum sequence range that can be requested	10,000
Maximum number of requests per day	1,000
Logon timeout (seconds)	5
Heartbeat interval (seconds)	30

Note

Clients cannot make further retransmission requests due to the number of requests for the day exceeding the maximum (i.e. 1000) may contact HKEx Vendor Support for assistance.

5.3.1.9 Processing of RTS retransmission data

Figure 1. Workflow of Retransmission



(Assuming a client is authorised to that channel ID and has not reached the maximum request limit.)

Please refer to *APPENDIX C – Pseudo code for processing retransmission data* for example on handling data from OMD-D Retransmission server.

5.3.2 Refresh Service

The OMD-D feed provides a refresh service (RFS), which allows clients to start intraday or recover from significant packet loss. The refresh is available per channel.

RFS periodically provides a full snapshot of the market. Not all the messages available from the live feed can be recovered from the refresh. However, all the message types, necessary for reconstructing an up-to-date image of the market, are available from the refresh.

The refresh packets are disseminated via dedicated multicast streams.

Similar to real time data transmission, the refresh data also come from two redundant lines, A and B. Clients can apply the Line Arbitration mechanism described in *5.2.2 Line Arbitration,* except that there is no retransmission.

It is advisable that clients utilise the refresh service under the following situations:

1. Intraday start
2. Large message gap
3. Delay in the RTS retransmission
4. RTS retransmission failure

5.3.2.1 RFS Snapshot

Please refer to the OMD-D Interface Specification for the coverage of snapshot data.

5.3.2.2 Processing a Refresh

Processing the refresh while coping with the live feed may be a challenging piece of functionality in the feed handler. There are several things to think about in order to process the refresh properly. The 4 main areas, which problems may perhaps arise, are:

1. Connectivity
2. Synchronisation
3. Determine a full refresh snapshot
4. Sequencing of events

Connectivity

There are 2 data streams that need to be handled during the refresh:

1. Live feed multicast
2. Refresh feed multicast

Synchronization

1. Subscribe to the real time MC channel and cache received messages.
2. Subscribe to the corresponding refresh multicast channel. Once subscribed, if messages are received instantaneously, clients should discard all messages till the arrival of a Refresh Complete message.
3. Wait for the next wave of snapshot data. Process all messages until the next Refresh Complete message is received.
4. Store the LastSeqNum sequence number provided in the above message.

5. If Sequence Reset message is received, reset the next expected sequence number to a value of NewSeqNo (1) field in current Sequence Reset message
6. Unsubscribe from the refresh MC channel.
7. Discard the cached real time messages with sequence number less than or equal to LastSeqNum, except for Sequence Reset messages which need to be processed. Please refer to Section 9.4 for details.
8. Process the remaining cached real-time messages and resume normal processing.

Determine a full refresh snapshot

When subscribing to OMD-D refresh multicast channel, clients should handle the following situations to recover a full image of the market:

1. The first message received is a Heartbeat

   When there is no message transmission (channel idle) in a refresh multicast channel, OMD-D sends Heartbeat message at a regular time interval (currently it is set to 2 seconds). Clients wait for the full refresh snapshot starting with a message other than Heartbeat till the arrival of Refresh Complete message.

2. The first message received is a Refresh Complete message

   Clients ignore the first Refresh Complete message and the subsequent Heartbeat message(s) in a refresh multicast channel. The next refresh snapshot starts with a message other than Heartbeat till the arrival of the second Refresh Complete message.

3. The first message received is neither Heartbeat nor Refresh Complete message

   Clients cannot receive a full refresh snapshot and should **NOT** process any message at the moment. Simply skip message(s) until a Refresh Complete message is received. Usually, OMD-D sends refresh snapshot at a regular interval. Heartbeat is disseminated in between two rounds of refresh snapshot. Similar to point 1, clients will then obtain a full market snapshot in the next refresh interval

4. Receive refresh messages follow by Sequence Reset message

   Receive “Sequence Reset Message” from refresh channel and clear the cached refresh messages from that refresh channel before. The next refresh snapshot starts with a message other than Heartbeat with sequence number start from 1 till the arrival of the Refresh Complete message.

   Note

   When the Clients listen to the refresh channels for the latest images, they may receive Refresh Complete message with “0” LastSeqNum in some channels. This indicates that no messages have been published in the corresponding real-time channels. That normally happens before market opens or may be due to no market activities.

   In the case of LastSeqNum being “0”, if the Clients receive only Heartbeat messages with SeqNum = “1” in real-time channels and they detect no packet loss by Line Arbitration or retransmission (i.e. the RTS returns “2 – Message not available”), OMD is running normally and the Clients have not missed any packets in the real-time channel.

   Clients should also be aware of the refresh message behaviour during Site Failover. On Site Failure, Sequence Reset message will be disseminated on all real-time and refresh channels. Clients should still use the standard handling to process refresh

messages under this situation. Clients may find the first Refresh Complete message carry a LastSeqNum value greater than that in the second Refresh Complete message but this should not be a concern because the processing would be the same as the Clients joining in the middle of an incomplete Refresh cycle, in which case, according to the standard refresh handling, Clients will discard all previous refresh messages and wait for the start of the next clean Refresh cycle.

The first clean Refresh cycle should start after the first Refresh Complete message. If no new messages have been disseminated in the real-time channels immediately after the Failover, the Refresh Complete message of the first Refresh cycle would carry a LastSeqNum of "0" but in this special case the refresh messages found in this first Refresh cycle would still carry the last valid snapshot data, if any, before the Site Failover. Clients should therefore apply the snapshot data under this situation.

Sequencing of events

It is important for clients to know which multicast channels hosts reference data for what other channels. Clients should not process any data (except for the reference data) until the full reference data is processed.

This implies the order of requesting refresh that clients should obey.

If the feed handler is started intraday, clients should **first go for the refresh of channels that serve reference data**. Only after the refresh of the reference data is received, clients should ask for the refresh of trades, order books, etc.

Note

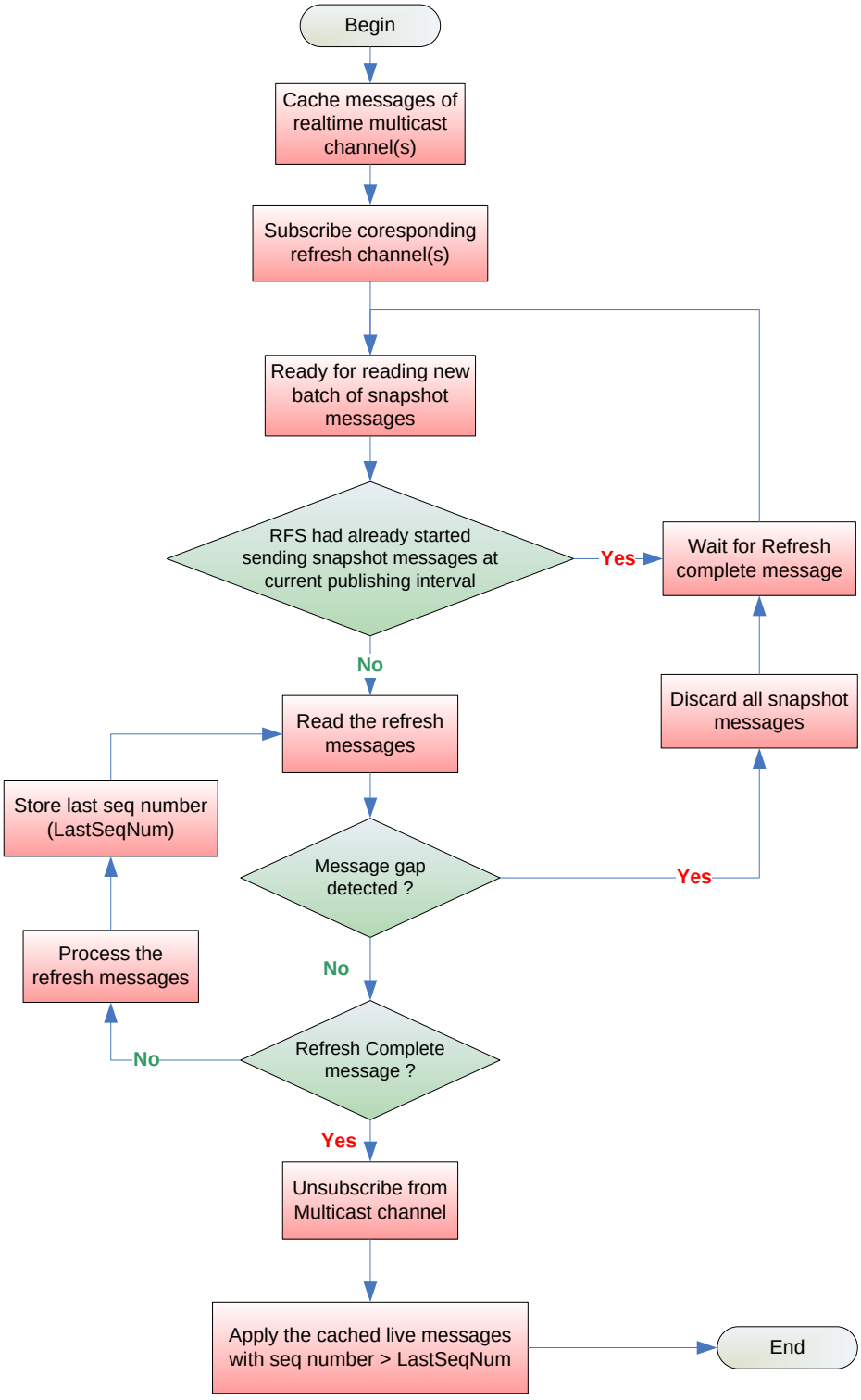
There is no TCP retransmission for refresh. Clients must monitor for packet loss on the refresh channels and wait for the next snapshot if loss is detected.

Exception Handling
– Arrival of Sequence Reset message in real-time channel in the middle of a Refresh cycle

The following steps are suggestions for handling of a rare situation where Sequence Reset messages are received while processing Refresh messages

1. Discard the Refresh messages received during the current Refresh cycle
2. Reset the next expected sequence number to 1 which should be the same as the value of NewSeqNo field in Sequence Reset message
3. Clear all cached data for all instruments.
4. Subscribe to the corresponding Refresh channels of all subscribed real time multicast channels to receive the current state of the market, following the same steps in this section for handling messages from Refresh channels

Figure 2. Workflow of Refresh



Please refer to *APPENDIX D - Pseudo code for processing Refresh snapshot packet* for example on handling data from OMD-D RFS.

5.3.2.3 Refresh for Active Instrument State

In order to compile the 'Active Instrument State' according to the methodology described in 5.2.3.1, clients are required to receive the complete Market Status (320) from Refresh in order to determine the latest market status. The complete pre-defined TSS hierarchy can be obtained from the Refresh channel for Market Status (320) messages but with only the latest market and instrument status for each pre-defined level.

5.3.2.4 Data Refresh at OMD-D internal failover

In very rare situation, OMD-D may re-disseminate the latest market image upon a successful failover of internal processes. During the failover, some intermediate updates may be lost but most of them will be re-disseminated. Please refer to the table below for messages which may have intermediate loss and may be re-disseminated:

Message Type	Intermediate lost	Re-disseminate
Commodity Definition (301)		✓
Class Definition (302)		✓
Series Definition Extended (304)		✓
Market Status (320)		✓
Commodity Status (322)		✓
Quote Request (336)	✓	
Trade Statistics (360)	✓	✓
Series Statistics (363)	✓	✓

Clients may receive duplicate messages of the above types in case of such failover and their feed handler should be able perform normally despite the duplicated messages.

6 RACE CONDITIONS

The real-time order/trade data and reference data are disseminated via separate channels, so users need to be aware of the possibility of a race condition.

For example, a Series definition Extended (304) message & Commodity Status (322) may be sent marking a series as suspended. However, for a very short time after this message, the regular order and trade information for this security may continue to arrive.

Another example would be a Market Status (20) message marking the trading session as halted, but real time data for the same market may continue to arrive for a short time afterwards.

7 AGGREGATE ORDER BOOK MANAGEMENT

Book updates are sent by OMD-D via Aggregate Order Book (353) messages. Each message may contain any combination of new, changed or deleted entries for a book or clear the whole book. The nature of an entry is defined by its UpdateAction.

Table 11. Actions on Aggregate Order Book Messages

Action	Description
New	Create/Insert a new price level
Delete	Remove a price level
Change	Update aggregate quantity at a price level

Action	Description
Clear	Clear the whole book

General Rules

- All entries within an Aggregate Order Book message must be applied sequentially.
- Clients must adjust the price level of entries below deleted or inserted entries.
- If a new book entry causes the bottom entry of a book to be shifted out of the book, i.e. below the 10th price level,
  1. If the shifted out entry is within 10 Price level, OMD-D will send an explicit deletion entry (Explicit delete).
  2. Clients must delete all entries below the 10th price level since there would not be any updates sent on those price levels.
  3. If the book shrinks again and these entries shifted back to the top 10 price levels, OMD-D will resend the entries at their new price levels with the latest updates.
- If a match causes an order to be removed so that there are now less than 10 levels visible, then OMD-D will also automatically send the additional price level(s) to make up the 10 price levels.
- If a clear aggregate order book message is received, client should clear all entries in the order book.

Please refer to *Section 6 – Aggregate Order Book Management in the Interface Specification of HKEx Orion Market Data Platform Derivatives Market Datafeed Products* for different scenarios on how OMD-D sends Aggregate Order Book message.

You may also refer to *APPENDIX E – Pseudo code for processing Aggregate Order Book Message* for example on handling Order Book messages from OMD-D server.

8 FULL ORDER BOOK MANAGEMENT

Developers maintain order books from Order Update Messages. Every event in the full order book is reported by OMD-D with the following message types being disseminated:

Table 12. Order Update Message Types

Message Type	Name
330	Add Order
331	Modify Order
332	Delete Order
335	OrderBook Clear
350	Trade

General Rules

- Clients should be able to uniquely identify the order (OrderID is the unique identifier per OrderBookID and Side)
- Determine the price and quantity of an order
- Subtract the quantity of the order as declared in Trade (350) message
- Delete the order from OrderBook when the contract quantity of the order is zero
- Clear the orderbook when OrderBook Clear (335) message received

Managing Full Order Book

- An order inserted at an existing position shifts the order on that position down (and all orders below as well. Value ‘1’ denotes the highest ranked order position.

- For a Modify Order (331), the order must be removed from its previous position and inserted at new 'OrderbookPosition'.
- A Delete Order (332) or fully filled Trade (350) causes existing orders below it to shift their position up one step.
- Trade (350) message signals a partial or full fill. The order quantity must be reduced by the quantity of the Trade (350) message.
- Modify Order (331) message signals that the order has been modified. The current rank may or may not be lost in the process 'OrderbookPosition' will show the new rank within the book.
- Delete Order (332) message tells the recipient to remove the order reference from the orderbook.
- The Orderbook Clear (335) message is used to inform subscribers that all existing orders should be removed from both the bid and ask sides of the specified orderbook. The message is typically used at the end of Auction

Some Cases for reference

Case 1:

- Insert one order between 2 existing orders
- The original order with the OrderBookPosition of the new orders and all other orders below are shifted down

Case 2:

- One order in the middle of the book matched/cancelled
- All orders below are shifted up

Case 3:

- Reduce quantity of an existing order
- OrderBookPosition unchanged (same as the implicitly shifted position caused by previous order book change)

Case 4:

- Increase quantity of an existing order
- The OrderBookPosition of the amended order is changed accordingly
- All orders previously below (except those below the new position of the amended order) are shifted up
- All orders at its new position and below are shifted down

9 EXCEPTION HANDLING

Listed below are some common exception handling procedures that clients must be capable of when subscribing to OMD-D:

- Late connection
- Intra-day refresh
- Client application restarts
- Sequence Reset Message
- OMD-D restarts before market open
- OMD-D node failover
- Site failover

9.1 Late Connection / Startup Refresh

When client starts late, all reference data should be recovered before the current image for all instruments across all channels.

Please refer to section *5.3.2.2 Processing a Refresh* for recovery procedures.

Note

- Some channels may host reference data for other channels.
- Channels which depend on other channels for reference data cannot be processed before full reference data has been received
- Clients must define relationships between channels

9.2 Intra-day Refresh

For each real time multicast channel, there exists a corresponding refresh multicast channel on which snapshots of the market state are sent at regular intervals throughout the business day.

When clients experienced an unrecoverable packet loss on a certain channel during the day, a snapshot is only needed for that channel.

Sequencing of events

1. Caches real time messages in the multicast channel that previously experienced packet loss
2. Listens to the corresponding refresh multicast channel and waits for the next snapshot *(refer to 5.3.2.2 Processing a Refresh - Determine a full refresh snapshot)*
3. Processes all refresh messages until the arrival of a Refresh Complete message
4. Store the LastSeqNum sequence number provided in the Refresh Complete message
5. Disconnects from the refresh multicast channel
6. Processes the cached real time messages with sequence number greater than the LastSeqNum. Otherwise, drop processing it.

Now the clients maintain the current market image.

9.3 Client Application Restarts

Similar to “Late Connection” as described earlier.

9.4 Sequence Reset Message

Sequence Reset Message from real time channels

Sequencing of events

1. Receive “Sequence Reset Message” from any real time multicast channel
2. Reset the next expected sequence number to 1 which should be the same as the value of NewSeqNo field in Sequence Reset message
3. Clear all cached data for all instruments.
4. Subscribe to the corresponding refresh channels of all subscribed real time multicast channels to receive the current state of the market. (*refer to 5.3.2.2 Processing a Refresh – for handling messages from Refresh channels)*
5. Resume to process real time messages

Packet loss detection when processing Sequence Reset Message

After a Sequence Reset, the first UDP packet should have a sequence number 1. However, this packet is lost and clients start receiving packet with sequence number 2 and onwards. Clients can try to recover it from the redundant line. If the lost packet is unrecoverable, clients should start buffering the live feed and send a retransmission request immediately.

Once clients finished processing the retransmitted messages from RTS, clients can maintain the latest market image by handling the buffered data and then the live feed.

Sequence Reset Message from Refresh channels
Refer to 5.3.2.2 Processing a Refresh – for handling sequence reset message from Refresh channels)

9.5 OMD-D Restarts Before Market Open

In case of OMD-D performs start-of-day twice (errors encountered during first start-of-day). The second start-of-day should trigger Sequence Reset in all channels. Clients should discard all reference data received in the first start-of-day and process the second start-of-day.

9.6 OMD-D Component Failover

In case no live data can be received in one of the OMD-D lines (say line A), there is no impact to clients as OMD-D would continue publishing data via the alternate line (line B). Clients can reapply line arbitration to retrieve the latest market information as usual when OMD-D resumed the service from line A.

In case "Sequence Reset Message" is received from refresh channels, client should clear the cached refresh messages and reset the sequence number of corresponding channel. (*Refer to 5.3.2.2 Processing a Refresh – for handling sequence reset message from Refresh channels)*

9.7 Site Failover

In case of any problem in primary OMD-D servers, the OMD-D servers will be brought up at the backup site. The multicast addresses of live and refresh data will remain the same. The primary RTS servers will not be available. Clients should use the backup IP addresses to connect to backup RTS servers.

Once the OMD-D DR site is ready, OMD-D sends sequence reset message per multicast channel. Clients subscribe to the refresh channels in order to receive the current state of the market. Please refer to section *9.4 Sequence Reset Message* for details.

APPENDIX A – Pseudo code to connect and receive multicast channel

An example shows how to set up UDP socket and join multicast channel.

```
int sock_fd;
int flag = 1;
struct sockaddr_in sin;
struct ip_mreq imreq;

// Create a socket
sock_fd = socket(AF_INET, SOCK_DGRAM, 0);

// Set socket option
setsockopt(sock_fd, SOL_SOCKET, SO_REUSEADDR, &flag, sizeof(int));

// Set IP, Port
memset(&sin, 0, sizeof(sin));
sin.sin_family = AF_INET;
sin.sin_addr.s_addr = INADDR_ANY;
sin.sin_port = htons(port);

// Bind
bind(sock_fd, (struct sockaddr *) &sin, sizeof(struct sockaddr))

// Add to Multicast Group
imreq.imr_multiaddr.s_addr = inet_addr(mcAddress);
imreq.imr_interface.s_addr = inet_addr(interface);

setsockopt(sock_fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, (const void *)&imreq, sizeof(struct p_mreq));
```

An example shows how to read data from a multicast channel.

```
size_t len;
socklen_t size = sizeof(struct sockaddr);
struct sockaddr_in client_addr;
char  mReadBuffer[2046];

memset(mReadBuffer, 0, sizeof(mReadBuffer));

// Read the data on the socket
len = recvfrom(fd, mReadBuffer, sizeof(mReadBuffer), 0, (struct sockaddr *) &client_addr, &size);
```

APPENDIX B – Pseudo code of Line Arbitration

An example shows how clients process a packet received from OMD-D. This function handles data received from Line A or Line B multicast channels.

```
void processPacket(Packet packetBuffer)
{
        if (packetBuffer.getSeqNum() > expectedSeqNum) {
                //Gap detected, recover lost messages

                //Spool Packet in memory and wait for short period
                //Gap may be filled from next few incoming packet,
                //either from same line or alternative line
                spoolMessages(packetBuffer);
        }
        else if (packetBuffer.getSeqNum() + packetBuffer.getMsgCount() < expectedSeqNum) {
                //Duplicate packet, ignore
        }
        else if (packetBuffer.containsSeqNum(expectedSeqNum)) {
                //Process the packet if it contains a message
                //with sequence number = expectedSeqNum
                int msgProcessCount = 0;

                for (int i=0; i < packetBuffer.getMsgCount(); i++) {
                    if (packet.getSeqNum() + msgProcessCount == expectedSeqNum) {
                        extractMessage(message, packetBuffer, msgProcessCount);
                        processMessage(message);
                        expectedSeqNum++;
                    }
                    else {
                        //Duplicate message, ignore
                    }
                    msgProcessCount++;
                }
        }
}
```

An example shows a timer function processes the spooled messages at a regular time interval.

```
void checkMessageSpoolTimer()
{
	MessageSpool::iterator i = mMessageSpool.begin();

	// Iterate through the message spool
	while (i != mMessageSpool.end())
	{
		Message message = i->second;

		// If the current packet sequence number is larger than expected,
		// there's a gap

		if (message.getSeqNum() > mNextSeqNum)
		{
			//No retrans request  sent for this message before
			if (! message.getRetransRequested())			{
			  sendRetransRequest(mNextSeqNum,
			                     message.getSeqNum() - 1);
			  return;
			}

			//time limit hasn't been reached, so it's still not an
			// unrecoverable gap. Return and wait..
			if (message.getTimeLimit() < poolTimeLimit) {
			  return;
			} else {
			  //The RetransRequest failed or took too long and the gap wasn't
			  //filled by the other line - The messages have been permanently
			  //missed. Recover the lost data from Refresh Server (RFS)
			  recoverFromRefresh();
			}
		}
		if (message.containsSeqNum(mNextSeqNum))
		{
			// The packet contains the next expected sequence number, so
			//process it
			processPacket (message);
		}
	}
}
```

APPENDIX C – Pseudo code for processing retransmission data

An example shows how clients process incoming data from OMD-D Retransmission server. It handles Heartbeat, RTS Logon Response, RetransRequest Response and Retrans data.

```
void read()
{
  readBuffer(mRtsBuffer);

  while (true)
  {
    // Get packet information at mRtsBuffer
    PacketHeader* packet = (PacketHeader*) mRtsBuffer;

    // If the entire packet is in the buffer, process it
    if (isEntirePacket(mRtsBuffer))
    {
      // If Heartbeat (i.e. packet with 0 MsgCount)
      if (packet->mMsgCount == 0)
      {
        sendRtsHeartbeat(mRtsBufPos, packet->mPktSize);
      }
      else
      {
        // Determine the kind of message(s) in the packet
        uint16_t msgType = packet.getMsgType();

        switch (msgType)
        {
          case LOGON_RESPONSE_TYPE:
          {
            LogonResponse *logonResponse
                              = (LogonResponse *)(mRtsBufPos + sizeof(PacketHeader));
            processLogonResponse(logonResponse);
            break;
          }
          case RETRANS_RESPONSE_TYPE:
          {
            RetransResponse* resp = (RetransResponse*) (mRtsBufPos + sizeof(PacketHeader));
            processRetransResponse(resp);
            break;
          }
          default:
            processPacket(packet);
        }
      }
    }
    // Wait for the rest of the data to come from the socket
    else
    {
      break;
    }
  }
}
```

APPENDIX D - Pseudo code for processing Refresh snapshot packet

An example shows how clients process refresh snapshot data from OMD-D RFS server and merge with realtime messages

```
void processRefreshPacket(Packet packetBuffer) {
        static int expectedSeqNum = 0;
        //First Message is Heartbeat or Refresh Complete Message
        if(! isStartOfRefresh(packetBuff) {
                return;
        }

        if (packetBuffer.getSeqNum() > expectedSeqNum) {
                //Gap detected
                clearSpoolMessage();
                return;
        }
        else if (packetBuffer.getSeqNum() + packetBuffer.getMsgCount() < expectedSeqNum) {
                //Duplicate packet, ignore
                return;
        }
        else if (packetBuffer.containsSeqNum(expectedSeqNum)) {
                spoolMessages(PacketBuff, expectedSeqNum);
        }

        if (isRefreshComplete(packetBuffer)) {
                List refreshMessageList = getRefreshSpoolMessage();
                processMessages(refreshMessageList);

                //Get spooled realtime message with seq num >= expectedSeqNum;
                List realtimeMessageList = getRealtimeSpoolMessage(expectedSeqNum);
                processMessages(realtimeMessageList);
        }
}
```

APPENDIX E – Pseudo code for processing Aggregate Order Book Message

An example shows how clients process Aggregate Order Book Message and update the internal order book.

```
OrderBook mOrderBook;

void processAggregateOrderBook(AggregateOB aggregateOB) {

      switch(aggregateOB.getAction())

      case ADD:
        int tickLevel = getTickLevel(mOrderBook, aggregateOB.getPrice());

        insertOB(mOrderBook, tickLevel, aggregateOB);

        //If Price level > 10, delete those order from OBMap
        deleteOBExceedMaxPriceLevel(mOrderBook);

      case Update:
         int tickLevel = getTickLevel(mOrderBook, aggregateOB);
         updateOB(mOrderBook, tickLevel, aggregateOB);

      case Delete:
          int tickLevel = getTickLevel(mOrderBook, aggregateOB);
         deleteOB(mOrderBook, tickLevel, aggregateOB);
         updateOBPriceLeve(mOrderBook)
      case Clear:
        clearOB(mOrderBook);

}
void insertOB(mOrderBook, tickLevel, newAggregateOB) {
      newAggregateOB.setTickLevel(tickLevel);
      mOrderBook.add(tickLevel-1, newAggregateOB);

      for (int i=tickLevel; i < mOrderBook.getSize(); i++) {
            AggregateOB aggregateOB = mOrderBook.get(i);
            aggregateOB.updateTickLevel(mOrderBook);
            aggregateOB.updatePriceLevel(mOrderBook);
      }
}
```