# DEVELOPERS GUIDE

## HKEX Orion Market Data Platform
## Derivatives Market Datafeed Products (OMD-D)

Version 2.1
28 April 2026

# Contents


1 INTRODUCTION ..... 4
2 DATA STRUCTURE ..... 5
2.1 Packet Header ..... 5
2.2 Heartbeats ..... 6
2.3 Message Header ..... 6
2.4 Message Formats ..... 7
3 ENDIAN ..... 7
4 FIELD ATTRIBUTES ..... 7
4.1 Null Values ..... 7
4.2 Currency Values ..... 8
5 MESSAGE PROECSSING ..... 8
5.1 Start of Day ..... 8
5.2 Normal Transaction ..... 9
5.2.1 Receive Multicast ..... 9
5.2.2 Line Arbitration ..... 9
5.2.3 Process Data Message ..... 11
5.2.3.1 Market Status update arrangement ..... 11
5.2.3.2 Building up Definitions ..... 11
5.2.3.3 Retrieval of outstanding orders from Add Order (330) messages in Order Feed Channel (For D-Lite only) ..... 12
5.2.3.4 Partitions Assignment ..... 13
5.2.3.5 Trade Amendment ..... 13
5.2.3.6 Calculated Opening Price reset when PRE-OPEN session end ..... 13
5.2.3.7 Next Day Effective Instruments ..... 13
5.2.3.8 Intra-Day created Instruments ..... 13
5.2.3.9 Message routing for SOM and non-SOM ..... 13
5.2.3.10 After Hours Trading (T+1) – Clarification on Trading Information ..... 14
5.2.3.11 Trade (350) message for Full OrderBook when Preopen session ends ..... 14
5.2.3.12 Alert ID in Market Alert (323) message ..... 14
5.2.4 Process Control Message (Heartbeats) ..... 14
5.2.5 Process Disaster Recovery Signal Message (105) ..... 15
5.2.6 Compression on Payload Message ..... 15
5.3 Recovery ..... 16
5.3.1 Retransmission Service ..... 16
5.3.1.1 Multiple Retransmission Servers ..... 16
5.3.1.2 Retransmission Logon ..... 16
5.3.1.3 Retransmission Logon Response ..... 17
5.3.1.4 Retransmission Heartbeats ..... 17
5.3.1.5 Retransmission Request ..... 17
5.3.1.6 Retransmission Response ..... 18
5.3.1.7 Retransmission Message ..... 19
5.3.1.8 Retransmission Limits ..... 19
5.3.1.9 Processing of RTS retransmission data ..... 20
5.3.1.10 Compression on Payload Message from Retransmission ..... 20
5.3.2 Refresh Service ..... 21
5.3.2.1 RFS Snapshot ..... 21
5.3.2.2 Processing a Refresh ..... 21
6 RACE CONDITIONS ..... 25
7 AGGREGATE ORDER BOOK MANAGEMENT ..... 25
8 FULL NORMAL ORDER BOOK MANAGEMENT ..... 26
9 FULL NORMAL ORDER BOOK AND AGGREGATE IMPLIED QUANTITY ..... 26
10 EXCEPTION HANDLING AND SPECIAL TRADING CONDITION ..... 28
10.1 Late Connection / Startup Refresh ..... 29

10.2 Intra-day Refresh ..... 29
10.3 Client Application Restarts ..... 30
10.4 Sequence Reset Message ..... 30
10.5 OMD-D Restarts Before Market Open ..... 30
10.6 OMD-D Restarts After Market Open (Intraday Restart) ..... 30
10.7 OMD-D Component Failover ..... 31
10.8 OMD-Index Component Failover ..... 33
10.9 Site Failover ..... 33
10.10 Special Trading Condition ..... 34
APPENDIX A – Pseudo code to connect and receive multicast channel ..... 35
APPENDIX B – Pseudo code of Line Arbitration ..... 36
APPENDIX C – Pseudo code for processing retransmission data ..... 38
APPENDIX D – Pseudo code for processing Refresh snapshot packet ..... 40
APPENDIX E – Pseudo code for processing Aggregate Order Book Message ..... 41
APPENDIX F – Pseudo code for processing compressed multicast data ..... 42

# 1 INTRODUCTION

The target readers of this document are the technical personnel of Market Data Vendors, End-Users, Application Service Providers ("ASPs") and Independent Software Vendors ("ISVs") of HKEX Orion Market Data Platform – Derivatives Market ("OMD-D").

This document contains guidelines and suggestions for 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.

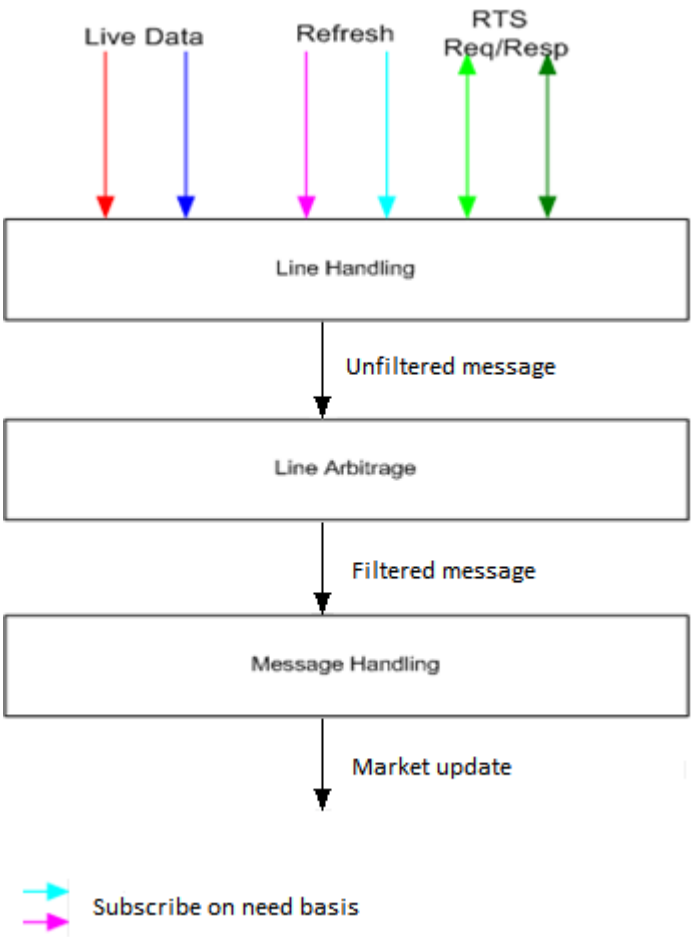
The scope of this document covers line arbitrage, packet and message processing, retransmission and refresh mechanisms, order book maintenance and exception handling.

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

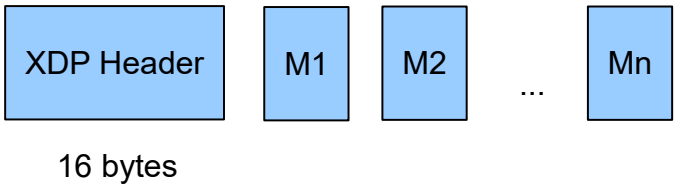
| | |
|---|---|
| FH | Feed handler |
| HA | High Availability |
| MC | Multicast |
| RFS | Refresh Server |
| RTS | Retransmission Server |
| T Session | Regular Trading Session |
| T+1 Session | After Hours Trading Session |
| UDP | User Datagram Protocol |
| XDP | Exchange Data Publisher |

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

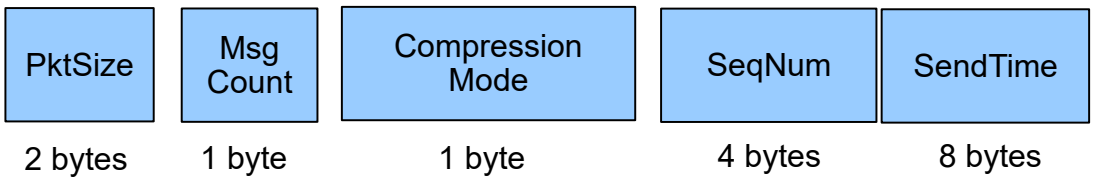
All packets disseminated from the OMD-D feed have a common packet header. This format is consistent across live, retransmission and refresh. An XDP packet consists of a 16-byte header followed by messages.

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 |
| Compression Mode | 3 | 1 | Uint8 | Indicate the messages in packet are compressed data and require decompression<br>- 0 indicates no compression applied on messages in the packet<br>- 1 indicates compression applied on messages in the packet |
| 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 |




The Compression Mode field in the packet header declares whether compression by using RFC-1950 ZLIB is applied on the message payload (i.e. a cargo of all messages within the same packet). For the detail of compression operation, please refer to 5.2.6 Compression on Payload.

## 2.2 Heartbeats

Heartbeats consist of a packet header with Compression Mode and 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.

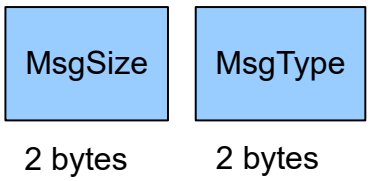
The Heartbeat message syntax is identical across OMD-D services.

## 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
{
        uint16_t mPktSize;
        uint8_t mMsgCount;
        uint8_t mMode;
        uint32_t mSeqNum;
        uint64_t 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 Trade (350), the Aggregate Order Book Update (353) message,

the Trade Statistics (360) message, the Trade Amendment (356) message, the Calculated Opening Price (364), the Open Interest (366) message as well as the Add (330) / Modify (331)/ Delete (332) Order messages.

The Int32 null representation is 0x80000000 (Hex 2's complement).
The Int64 null representation is 0x8000000000000000 (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 PROECSSING

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

The maintenance window of OMD Derivatives Market ("OMD-D") starts right after the system shuts down for the day until 6:00am the next business day. During the maintenance window, there could be system maintenance and housekeeping operations where OMD-D may be started up and shut down intermittently with the natural consequence of messages (e.g. Sequence Reset) sent via some multicast channels. In this regard, Clients are advised to make connection to OMD-D at or after 6:00 a.m. every business day to ensure that the data received from OMD-D are good for the start of the day.

**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)

  During the startup of OMD-D server (5:00 a.m. – 6:00 a.m.), Sequence Reset (100) messages and large volume of market initiation messages may be sent at high speed and a client may fail to receive the Sequence Reset (100) message because of the timing of its connection to the OMD-D server. Clients should therefore build in the flexibility in their feed handlers to handle such situations. Or they are advised to make connection to the OMD-D server after 6:00 a.m. and obtain the latest market image from Refresh service. For details of processing Sequence Reset (100) message, please refer to section ***10.4 Sequence Reset Message***

- OMD-D sends Reference Data messages below
  - Commodity Definition (301)
  - Class Definition (302)
  - Instrument Definition (303)
  - 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 received from the channel. Please refer to section ***10.5 OMD-D Restarts Before Market Open***

- 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 ***10.4 Sequence Reset Message*** for details.

- Sequence Reset message are sent individually on each channels. Client may experience long time difference on Sequence Reset messages between different channels during OMD-D startup.

**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<br>OrderUpdate2<br>OrderUpdate3 | 3 | 101 |
| Trade1<br>OrderUpdate4 | 2 | 104 |
| Trade2<br>Statistics1 | 2 | 106 |

| Secondary | | |
|---|---|---|
| **SN** | **MC** | **Messages** |
| 101 | 2 | OrderUpdate1<br>OrderUpdate2 |
| 103 | 3 | OrderUpdate3<br>Trade1<br>OrderUpdate4 |
| 106 | 2 | Trade2<br>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**

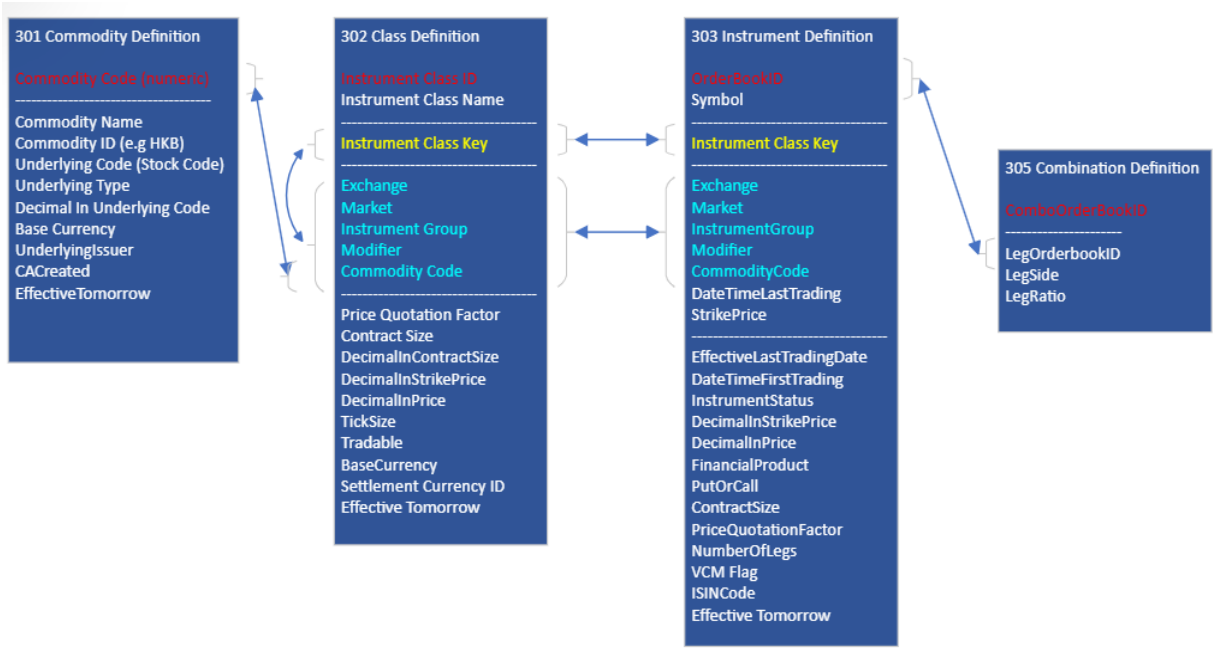
If the missed packets/messages come on the redundant line before they come from the retransmission, clients will simply process them and discard the retransmitted ones.

### 5.2.3 Process Data Message

#### 5.2.3.1 Market Status update arrangement

This section will be further updated with details in the next version.

#### 5.2.3.2 Building up Definitions




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

Tradable Instrument

- All tradable instruments, including suspended instruments, of the current business day are provided under Instrument Definition (303) message at the start of day. Intra-day created instruments are also declared in this message at real time. In Instrument Definition (303) message, the field "Instrument Status" declares the initial status (active/suspended) of the instrument in derivatives trading system.

Instrument Key Elements

- In Instrument Definition (303), the 7 key elements (Exchange, Market, Instrument Group, Modifier, Commodity code, Last Trading Date and Strike Price fields) are available for each tradable instrument for clients to map across the current application.

Instrument Class Key

- The field "InstrumentClassKey" not only allows clients to build the linkage between Class Definition (302) message and Instrument Definition (303) message, but can also be used as a short-cut key to link with Market Status (320), Commodity & Class Status (322) and THM Trigger (325) messages. Instead of using multiple key elements, this new short-cut key can be used as an alternative in looking up a particular group of instruments in a more simplified and efficient way.

### 5.2.3.3 Retrieval of outstanding orders from Add Order (330) messages in Order Feed Channel (For D-Lite only)

All outstanding orders of selected instruments are transmitted through Add Order (330) message on a snapshot basis and refreshed every second.

The transmission of Add Order (330) message is similar to transmission of refresh cycles and each cycle is ended with a Refresh Complete (203) message. For detail on processing refresh, please refer to ***5.3.2.2 Processing a Refresh.***

The snapshot interval may take longer than one second if the volume of the outstanding orders is exceptionally high, so clients should always rely on the Refresh Complete (203) message to determine the end of a full cycle. Heartbeat messages may not be sent between two cycles.

#### 5.2.3.4 Partitions Assignment

There are 6 partitions assigned in OMD-D and different derivatives trading products are assigned to different partitions. For details of the mapping between the partitions and products, please refer to OMD-D Interface Specification.

#### 5.2.3.5 Trade Amendment

When there is a trade amendment or trade cancellation, OMD-D broadcasts the Trade Amendment (356) message. Upon receiving a Trade Amendment (356) message for amendment or cancellation, a client's application could utilize the fields "OrderBookID" and "TradeID" to locate which trade is being amended or cancelled.

#### 5.2.3.6 Calculated Opening Price reset when PRE-OPEN session end

During the Pre-Open session, Calculated Opening Price (364) message is sent for instruments for which pre-market opening is available. At the end of the Pre-Open 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-Open session. At the end of the Pre-Open 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 (335) message and market status are sent in separate channels. Clients may experience different message inbound order due to race condition.

#### 5.2.3.7 Next Day Effective Instruments

OMD-D disseminates the next day effective instruments after market close of T Session. For those next day effective instruments, "EffectiveTomorrow" field is defined as 'True' in Instrument Definition (303) message. Please note that no OrderBookID is provided in Instrument Definition (303) message for those next day effective instruments, and no calendar spread instrument (i.e. combination instrument) is covered in next day effective instrument declaration.

#### 5.2.3.8 Intra-Day created Instruments

OMD-D disseminates the intra-day created instrument at real time once the instrument is created in the derivatives trading system. Tailor-Made Combo Instrument and Flexible Options are examples of intra-day created instruments.

When a new instrument is created in market, Instrument Definition (303) is sent for the definition of the instrument. If the instrument is a calendar spread instrument (i.e. combination instrument), 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 instrument.

#### 5.2.3.9 Message routing for SOM and non-SOM

In OMD-D, messages are classified into SOM (Stock Option market) and Non-SOM (Non-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 | |
| Commodity & Class Status (322) | Yes | *When the field "InstrumentClassKey" = 0* |
| Market Alert (323) | Yes | |

### 5.2.3.10 After Hours Trading (T+1) – Clarification on Trading Information

| Information | OMD-D Message | Notes |
|---|---|---|
| Open Interest (OI)<br>(both Gross and Net OI) | Open Interest (366) | When issued at start of day, the latest OI<br>▪ Day Indicator = 1, i.e. for Previous Trading Day<br>▪ For products tradable in T+1 session, it is the OI after the T+1 session. However, it could be different from the OI of the previous day published in the Daily Market Report on the HKEX website<br>▪ For products not tradable in T+1 session, it is the same OI as provided around end of day of previous day in general |
| | | When issued around the end of day,<br>▪ Day Indicator = 0, i.e. for Current Trading Day<br>▪ The OI as of the end of the current day, not covering the transactions in the following T+1 session<br>▪ 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<br>▪ Settlement price is determined once only at the end of a settlement cycle which includes the T+1 session of the previous day and the regular trading session of the current day |

### 5.2.3.11 Trade (350) message for Full OrderBook when Preopen session ends

For instruments under the market with Pre-Open session, Trade messages (350) will be sent for those committed trades at the end of the Pre-Open session. Those Trade (350) messages will not be able to reference in OrderBook because no OrderBook (330, 331 & 332) messages are sent during Pre-Open session. Trades concluded in Pre-Open session can be identified in Trade (350) message where by the field 'Match Type' has the value of 1 (Opening Uncross).

### 5.2.3.12 Alert ID in Market Alert (323) message

In any business day, the field 'Alert ID' uniquely identifies the key of the Market Alert (323) message.

Alert ID will be repeated in separated Market Alert (323) messages when a message is broken down into multiple Market Alert (323) messages. The 'Header' in those Market Alert (323) messages contains the same information and the last Market Alert (323) message under the same Alert ID is indicated by the 'LastFragment'.

## 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-D Interface

Specification Section 2.2.2 for the multicast heartbeat interval & Section 4.3 for the unicast heartbeat interval currently set in OMD-D), 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 | |

### 5.2.5 Process Disaster Recovery Signal Message (105)

The Disaster Recovery (DR) Signal message indicates to clients whether OMD-D is operating in the primary site or the backup site. See Section *10.9* on how to handle OMD-D site failover making use of the DR Signal message.

### 5.2.6 Compression on Payload Message

In packet header, the field 'Compression Mode' indicates whether the compression has been applied to the payload (i.e. a cargo of all messages within the same packet). Clients should read the Compression Mode in every packet header and determine if decompression (i.e. RFC-1951 DEFLATE) is required to apply on the received payload before processing the messages. Compression feature is enabled in specific channel in OMD-D. Below are the reference of data compression and decompression method employed in OMD.

- RFC-1950, ZLIB Compressed Data Format Specification version 3.3
- RFC-1951, DEFLATE Compressed Data Format Specification version 1.3

Please note: Message traffic will be reviewed regularly, and would turn on compression when needed. Therefore, it is suggested client to implement system logic to determine whether executing or bypassing the decompression based on the "Compression Mode" field in packet header, to minimise future system change.

## 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

Both OMD-D Feed and OMD-Index Feed have their own dedicated RTSs. For small message gaps, clients can recover lost messages using the 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 50000 messages 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. The RTS daily limit of 1000 requests for OMD-D and OMD-Index are counted separately.

**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.

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

#### 5.3.1.1 Multiple Retransmission Servers

Four RTSs, two on primary site and two on backup site, are all active and available for connection after OMD-D has been brought up. Client should connect to any one of the other three RTSs in case the first connected RTS encounters any issue. This is a part of the High Availability design and is meant to provide clients with a seamless service in case of the failure on any one of the RTSs.

The above design is also applicable to the RTSs dedicated for OMD-Index Feed.

#### 5.3.1.2 Retransmission 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<br><br>(Note: Please padded with NULLs after the username to fill up the 12 characters field) |

#### 5.3.1.3 Retransmission 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 or IP Address |
| 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 Retransmission 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 Retransmission 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<br>Msg2<br>Msg3 | Msg 1 (101)<br>Msg 2 (102)<br>Msg 3 (103) | N |
| 1 | 104 | Msg4<br>Msg5<br>Msg6 | Msg 4 (104)<br>Msg 5 (105)<br>Msg 6 (106) | N |
| 1 | 109 | Msg7<br>Msg8 | Msg 7 (109)<br>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 Retransmission 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. Clients may contact us for assistance.

### 5.3.1.7 Retransmission 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 Retransmission Limits

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

Table 10a. Retransmission System Limits for OMD-D

| System Limit | Value |
|---|---|
| Last number of messages available per channel ID | 50,000 |
| 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 |

Table 10b. Retransmission System Limits for OMD-Index

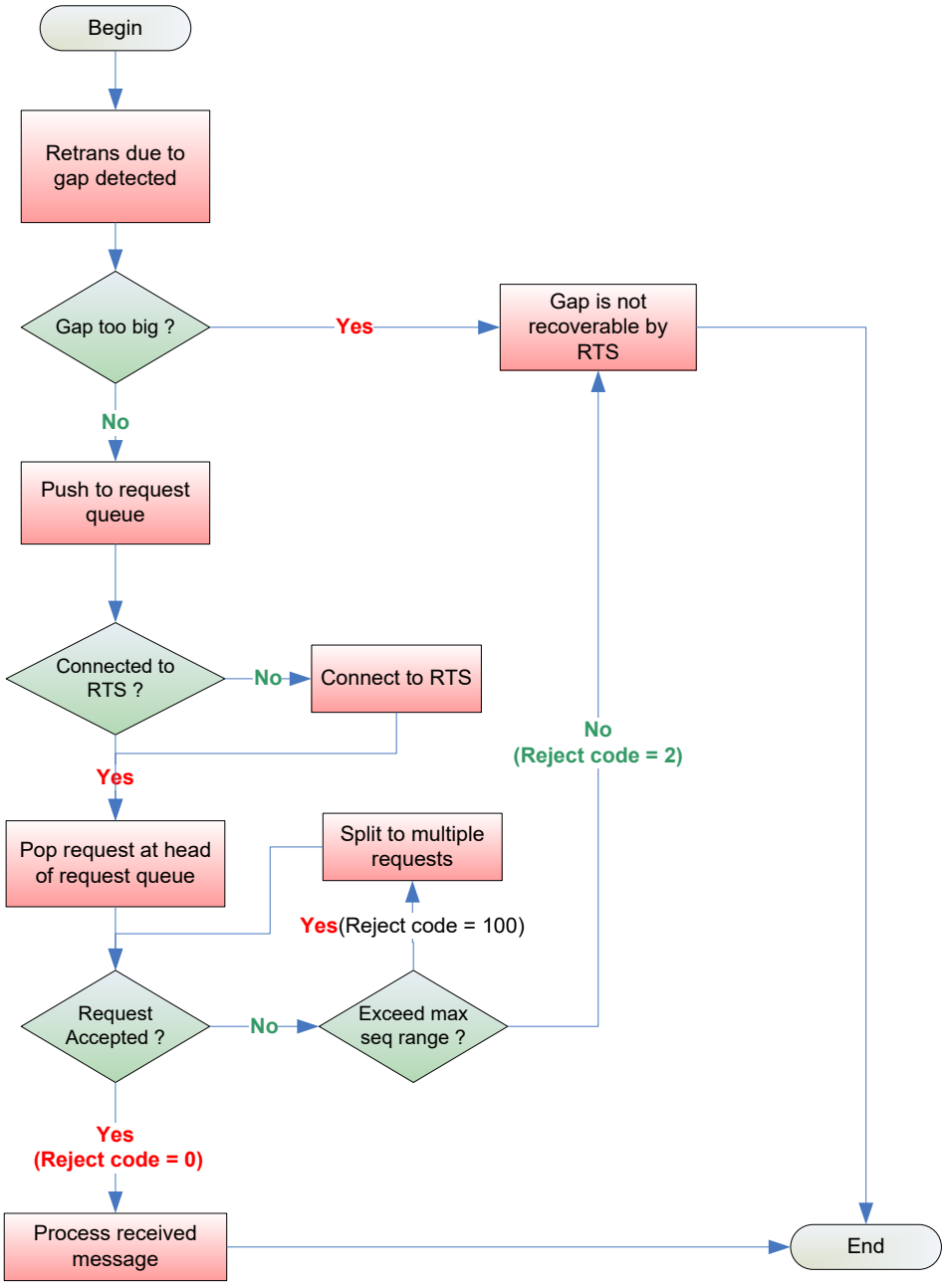
| System Limit | Value |
|---|---|
| Last number of messages available per channel ID | 50,000 |
| Maximum sequence range that can be requested | 10,000 |
| Maximum number of requests per day | 1,000 |
| Logon timeout (seconds) | 5 |
| Heartbeat interval (seconds) | 30 |

**Note**

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

### 5.3.1.10 Compression on Payload Message from Retransmission

Same as real time and refresh messages, the field 'Compression Mode' in packet header indicates whether compression has been applied on the payload messages returned from the retransmission

server. The returned payload, which is from the channel involving the compression feature, would normally have the packet header with the Compression Mode = 1.

Clients should always read the Compression Mode in the packet header and apply RFC-1951 DEFALTE on the returned payload before processing the messages.

Important : Before applying the RFC-1951 DEFLATE on payload, clients should ensure the full message has been received (i.e. same length as packet size).

Please note: Message traffic will be reviewed regularly, and would turn on compression when needed. Therefore, it is suggested client to implement system logic to determine whether executing or bypassing the decompression based on the "Compression Mode" field in packet header, to minimise future system change.

### 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, in 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. The Refresh Complete message marks the beginning of the next refresh cycle as well as the end of the previous refresh cycle.
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, discard all refresh messages received in the current refresh cycle. and restart refresh processing afresh in the next refresh cycle which will be marked by the appearance of the next Refresh Complete message
6. Unsubscribe from the refresh MC channel.
7. Discard the cached real time messages with sequence number less than or equal to LastSeqNum found in the Refresh Complete message, except for Sequence Reset messages which need to be processed. Please refer to *Section 10.4* 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 channels, clients should handle the following situations to recover a full image of the market:

1. The first message received is a Heartbeat

   If there is no message transmission (channel idle) in a refresh multicast channel, OMD-D sends Heartbeat messages at a regular time interval (currently it is set to 2 seconds) in the interim period. Heartbeat message can be sent in the middle of a refresh cycle or between two refresh cycles.

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 and ends with the arrival of the second Refresh Complete message.

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

   Clients are in the middle of a refresh cycle and cannot receive a full refresh snapshot from this cycle. They 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 may be disseminated in between two rounds of refresh snapshots, or there can be two rounds of refresh snapshots without Heartbeat in between if time gap is 2 seconds or less. With or without Heartbeats, Clients can obtain a full market snapshot in the next refresh interval after the first Refresh Complete message received.

   **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-D is running normally and the Clients have not missed any packets in the real-time channel.

**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**

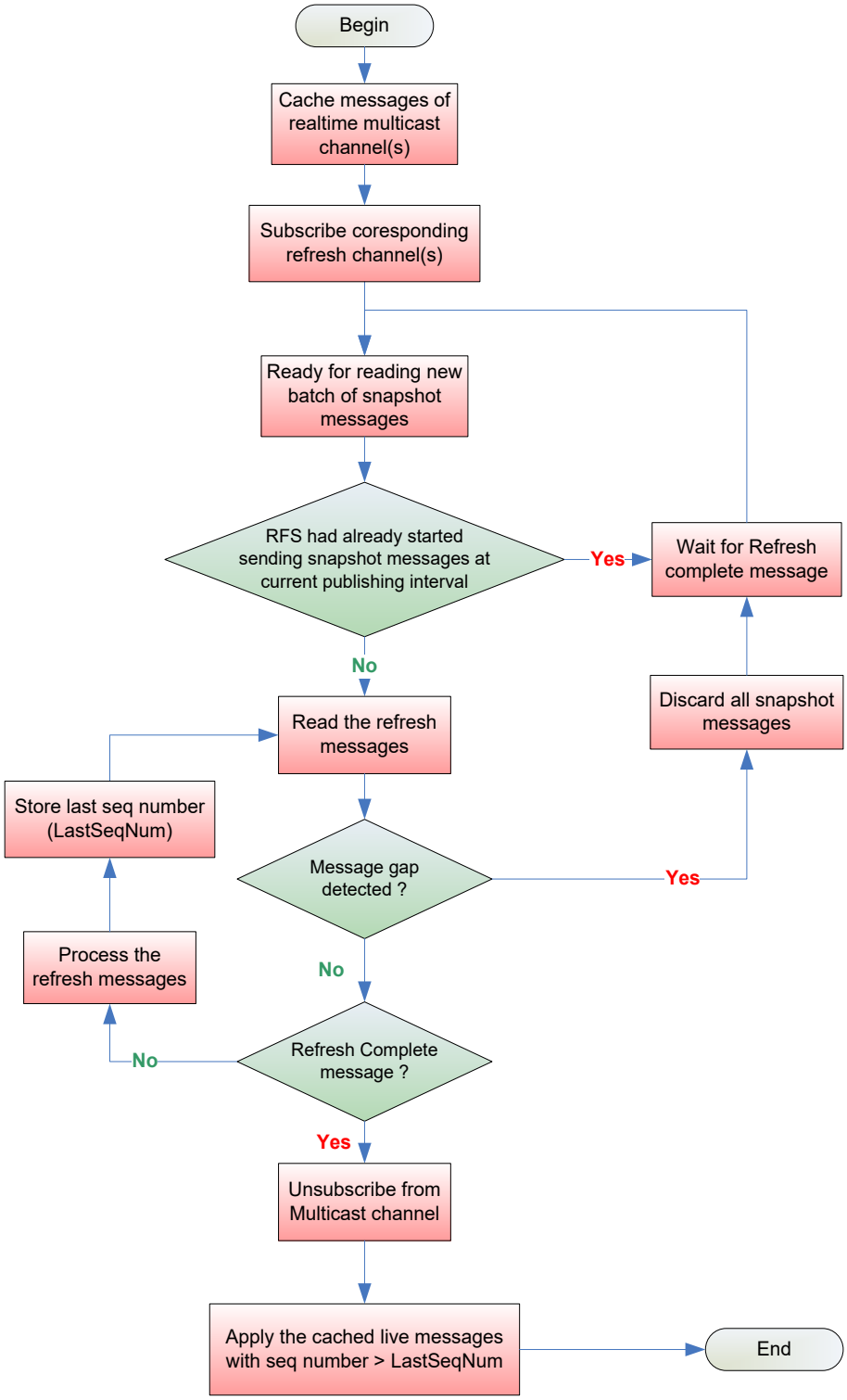
There is no TCP retransmission for refresh. Clients must monitor for packet loss on the refresh channels and wait for the next snapshot if loss is detected.

***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 the 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.

# 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.

Some examples as follows:

- An Instrument Definition (303) message and Commodity & Class Status (322) may be sent marking an instrument as suspended. However, for a very short time after this message, the regular order and trade information for this instrument may continue to arrive.
- A Market Status (20) message marking the trading session as closed, but real time data for the same market may continue to arrive for a short time afterwards.
- During start of the day, an Add Order (330) message, mainly for "Good Till Cancel" orders carried forward from the previous trading day, may be sent before the static data arrives.

# 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 |
| 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 clear aggregate order book message is received, clients should clear all entries in the order book.
- The field "NumberOfOrders" represents the number of normal orders only. When only implied quantity is presented at the particular price level, but no normal order, the value of the field "NumberOfOrders" would be equal to 0.
- Implied order is derived from normal orders, and it is always placed in the last position on each price level. ODP-T provides the aggregate quantity of the implied orders on price level as separated information. Subject to clients' business needs, clients may decide if it is necessary to provide the implied order information in their services.
- The field "AggregateQuantity" declares the aggregated quantity of normal order while the field "AggregateImpQuantity" declares the aggregated quantity of implied order. Subject to clients' business needs, clients may consider summing up these two quantities in displaying the aggregate order book in their service.

For DS and DP

- If a new book entry causes the bottom entry of a book to be shifted out of the book, i.e. below the 10$^{th}$ price level,
  1. Clients must delete all entries below the 10$^{th}$ price level since there would not be any Aggregate Order Book Update message sent on those price levels (Implicit delete).
  2. 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.

For D-Lite

- If a new book entry causes the bottom entry of a book to be shifted out of the book, i.e. below the 5th price level,
  1. Clients must delete all entries below the 5th price level since there would not be any Aggregate Order Book Update message sent on those price levels (Implicit delete).
  2. If the book shrinks again and these entries shifted back to the top 5 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 5 levels visible, then OMD-D will also automatically send the additional price level(s) to make up the 5 price levels.

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 NORMAL ORDER BOOK MANAGEMENT

Developers maintain order books from Order Update Messages for all normal orders. 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 |

### 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
- Clear the orderbook when OrderBook Clear (335) message received

### Managing Full Normal 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.
- 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 clients 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 Pre-Open session

# 9 FULL NORMAL ORDER BOOK AND AGGREGATE IMPLIED QUANTITY

Clients may consider building one consolidated order book by using the full normal order book (Reference to ***8 FULL NORMAL ORDER BOOK MANAGEMENT***) and Aggregate Implied Quantity (337) messages, subject to their business needs.

### General Rules

- At the same price level, the priority of Normal order is always higher than the aggregate implied quantity.
- Aggregate implied quantity is not considered as normal order and would only be updated via Aggregate Implied Quantity (337) message.
- Add the aggregate implied quantity into the consolidated order book when Aggregate Implied Quantity (337) message at the given price level is received at the first time.
- Remove the aggregate implied quantity at the given price level from the consolidated order book when Aggregate Implied Quantity (337) message with "ImpliedQuantity = 0" is received.

An example is illustrated as below:

Full Normal Order Book (Built by Add Order (330) / Modify Order (331) / Delete Order (332) messages)

| Bid | | | Ask | | |
|---|---|---|---|---|---|
| Order # | Quantity | Price | Price | Quantity | Order # |
| 1 | 2 | 9999 | 10000 | 7 | 1 |
| 2 | 3 | 9999 | 10005 | 2 | 2 |
| 3 | 5 | 9999 | 10005 | 1 | 3 |
| 4 | 1 | 9998 | 10005 | 1 | 4 |
| 5 | 1 | 9998 | 10005 | 3 | 5 |
| 6 | 2 | 9998 | 10005 | 2 | 6 |
| 7 | 1 | 9998 | 10005 | 1 | 7 |
| 8 | 4 | 9997 | 10005 | 4 | 8 |
| 9 | 1 | 9997 | 10006 | 2 | 9 |
| 10 | 5 | 9992 | | | |
| 11 | 6 | 9991 | | | |

Implied Order Book (Built by Aggregate Implied Quantity (337) message)

| Bid | | Ask | |
|---|---|---|---|
| Aggregate Quantity | Price | Price | Aggregate Quantity |
| 15 | 9999 | 10000 | 7 |

| | | | |
|---|---|---|---|
| 8 | 9998 | 10001 | 9 |
| 5 | 9996 | 10003 | 6 |
| 10 | 9995 | 10005 | 4 |
| | | 10006 | 8 |

Consolidated Order Book

| Bid | | | Ask | | |
|---|---|---|---|---|---|
| Order # | Quantity | Price | Price | Quantity | Order # |
| 1 | 2 | 9999 | 10000 | 7 | 1 |
| 2 | 3 | 9999 | 10000 | 7 | Implied |
| 3 | 5 | 9999 | 10001 | 9 | Implied |
| Implied | 15 | 9999 | 10003 | 6 | Implied |
| 4 | 1 | 9998 | 10005 | 2 | 2 |
| 5 | 1 | 9998 | 10005 | 1 | 3 |
| 6 | 2 | 9998 | 10005 | 1 | 4 |
| 7 | 1 | 9998 | 10005 | 3 | 5 |
| Implied | 8 | 9998 | 10005 | 2 | 6 |
| 8 | 4 | 9997 | 10005 | 1 | 7 |
| 9 | 1 | 9997 | 10005 | 4 | 8 |
| Implied | 5 | 9996 | 10005 | 4 | Implied |
| Implied | 10 | 9995 | 10006 | 2 | 9 |
| 10 | 5 | 9992 | 10006 | 8 | Implied |
| 11 | 6 | 9991 | | | |

# 10 EXCEPTION HANDLING AND SPECIAL TRADING CONDITION

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

**Exception Handling**
- Late connection
- Intra-day refresh

- Client application restarts
- Sequence Reset Message
- OMD-D restarts before and after market open
- OMD-D component failover
- OMD-Index component failover
- Site failover

**Special Trading Conditions**
- Hong Kong Holiday which is not a Holiday in Mainland

In order to facilitate clients' verification of their exception handling ability, some of the exceptional scenarios are included in the Readiness Test which clients have to pass in order to get on board. Emergency drills are also held in a production-like environment on a regular basis for clients to test their systems and practise their operations on the handling of other exceptional scenarios such as backup site failover. Please refer to the following sub-sections for the availability of test/drill session for each of the exceptional scenarios. Client notice will be issued to announce the schedule and coverage of a regular rehearsal in advance.

## 10.1 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

Clients can test late connection in the Readiness Test environment as they wish.

## 10.2 Intra-day Refresh

For each real time multicast channel (except those for Trade, Trade Amendment & Quote Request which are not recoverable via the Refresh service), 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. Clear all cached data on that channel
2. Caches real time messages in the multicast channel that previously experienced packet loss
3. 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)*
4. Processes all refresh messages until the arrival of a Refresh Complete message
5. Store the LastSeqNum sequence number provided in the Refresh Complete message
6. Disconnects from the refresh multicast channel
7. 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.

This exceptional scenario is covered in the Readiness Test.

## 10.3 Client Application Restarts

Similar to "Late Connection" as described earlier.

## 10.4 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 on that channel.
4. Subscribe to the corresponding refresh channel of the real time multicast channel 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)*

This exceptional scenario is covered in the Readiness Test.

## 10.5 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.

This exceptional scenario is covered in the Readiness Test.

## 10.6 OMD-D Restarts After Market Open (Intraday Restart)

When OMD-D fails during trading hours, besides failing over to the backup site to resume service, OMD-D may be recovered by Intraday Restart where OMD-D will be shut down and then restarted. When OMD-D is shut down, retransmission services in both primary and backup sites will be disconnected and no multicast traffic, including heartbeat, will appear on all real-time and refresh channels. Clients will be informed if Intraday Restart needs to take place.

Clients will be notified when OMD-D is subsequently restarted. Upon receipt of our notification, all clients should clear all their internal cache and reconnect to OMD-D afresh in the same way as their systems start up late and connect to OMD-D only after the market has opened. In this situation, clients should not rely on the presence of Sequence Reset messages in any channels upon their reconnection and they need to go through the refresh service (see 5.3.2.2) to establish the latest market snapshot.

The duration between OMD-D's shutdown and the subsequent restart could be a timespan of an hour and clients need to observe announcement made.

This exceptional scenario is among the possible scenarios to be covered in a regular emergency drill.

## 10.7 OMD-D Component Failover

**Channels under Streaming Component**

Component failover on the following channels under streaming component would only impact one of the OMD-D lines. 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 alternative line (line B). Clients can reapply the line arbitration as usual when OMD-D resumes the service and publish the data from line A. Data published during the interruption would not be resent after line A has resumed the service.

The data published in the resumed line might be from backup site and thus there could be a larger time difference between two lines after the component failover.

This exceptional scenario is covered in the readiness test.

| Channels under Streaming Component | Message Type List | Datafeeds |
|---|---|---|
| Level 2 Price, COP and Trade Channel | Aggregate Order Book Update (353)<br>Calculated Opening Price (364)<br>Trade (350) | DP |
| Order, COP and Trade Channel | Add Order (330)<br>Modify Order (331)<br>Delete Order (332)<br>Orderbook Clear (335)<br>Aggregate Implied Quantity (337)<br>Calculated Opening Price (364)<br>Trade (350) | DF |
| Trade Channel | Trade (350) | DT |
| Quote Request Channel | Quote Request (336) | D-Lite, DS, DP and DF |
| Statistic Channel | Trade Statistic (360) | DP |
| Status and Trigger Channel | Market Status (320)<br>Instrument Status (321)<br>Commodity & Class Status (322)<br>VCM Trigger (324)<br>THM Trigger (325) | D-Lite, DS, DP and DF |

**Channels under Conflated Component**

Component failover could cause the following channels under conflated Component to experience an outage for a few minutes. When the service resumes, in the situation mentioned in table, the following recovery messages will be published to update clients for the latest market image.

This exceptional scenario is covered in the readiness test.

| Channels under Conflated Component | Message Type List | Datafeeds |
|---|---|---|
| Reference Data Channel | Commodity Definition (301)<br>Class Definition (302)<br>Instrument Definition Base (303) | D-Lite, DS, DP and DF |

| | | |
|---|---|---|
| | Combination Definition (305) | |
| Open Interest Channel | Open Interest (366) | D-Lite, DS and DP |
| Level 2 Price and COP Channel | Calculated Opening Price (364)<br>Aggregate Order Book Update (353) | D-Lite and DS |
| Statistic Channel | Trade Statistic (360) | D-Lite and DS |
| Market Alert Channel | Market Alert (323) | D-Lite, DS, DP and DF |
| Implied Volatility Channel | Implied Volatility (367) | DP |
| Trade Amend and Block Trade Channel | Trade (350)<br>Trade Amendment (356) | DT, DP and DF |
| Order Feed Channel | Add Order (330) | D-Lite Order Feed |

| Recovery Type | Message Type List | Remarks |
|---|---|---|
| Message Resent | Commodity Definition (301)<br>Class Definition (302)<br>Instrument Definition (303)<br>Combination Definition (305)<br>Market Alert (323)<br>Trade (350)<br>Trade Amendment (356)<br>Trade Statistic (360)*<br>Open Interest (366)<br>Implied Volatility (367) | For any new updates during the outage, corresponding messages will be resent.<br><br>Duplicate messages may be received. For example, Trade Statistics messages would be resent after the failover, however, clients may have received them before the failover. Clients' application should be able to perform normally and present the correct latest market images despite the resent messages.<br><br>* Applicable to D-Lite and DS only. Some intermediate updates may be lost during the failover, but most of them will be resent. Clients should refer to the last received message as the latest market information |
| Aggregate Order Book | Aggregate Order Book Update (353) | For instrument having updates during the outage, Aggregate Order Book Update (353) message with update action "Order Book Clear" will be sent, followed by a series of message with update action "New" for clients to rebuild the latest aggregate order book image.<br><br>For instrument without updates during the outage, Aggregate Order Book Update (353) message with update action "Order Book Clear" will not be sent. Clients can keep building the aggregate order book as usual. |

If "Sequence Gap" is detected in both data lines on the channel, clients should consider the recovery steps as documented in (Section 5.3 Recovery)

**Refresh Channels**

In case "Sequence Reset Message" is received from refresh channels, clients 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)*

This exceptional scenario is covered in the Readiness Test.

## 10.8 OMD-Index Component Failover

**Index Channels**

In case of the component failover of the index channels, the dissemination of Index Data (71) message will be resumed at the next update period.

If there is any update on index data during the failover period, a data report with those missing index data will be provided to Index Feed subscribers at the end of the business day.

**Refresh Channels**

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 section **6.2.2 Processing a Refresh** – for handling sequence reset message from Refresh channels)*

## 10.9 Site Failover

In case of severe technical problem that OMD-D needs to failover to the backup site for resumption of market data service, Disaster Recovery Signal (105) (referred to as "DR Signal" hereafter) message will indicate the status of OMD-D on backup site for client actions.

A dedicated multicast channel is assigned to transmit DR Signal. During normal days, OMD-D starts up on the primary site, the DR Signal message only carries heartbeats. In case of the backup site taking over the primary site, the DR Signal transmitted from the dedicated channel will carry a DR status instead of heartbeats. At that point, the multicast addresses of real-time and refresh data will remain the same whereas clients should switch to the backup IP addresses to connect to the backup RTSs for Retransmission Service.

When OMD-D starts operating on the backup site, DR Signal messages will be sent out with DR Status "1" (DR in progress) until the site failover process is completed where the DR Status in the DR Signal message will be changed to "2" (DR completed). At this point, OMD-D is considered operating normally on the backup site and the latest market snapshots can be obtained from the Refresh channels. OMD-D will continue transmitting the DR Signal with DR Status "2" until end of business day.

Clients are advised to clear all OMD-D data previously cached when DR Status "1" is detected in the DR Signal and prepare to execute their failover procedure if any. When DR Status "2" (DR completed) appears, clients could rebuild the market image based on the refresh service similar to the case when their feed handler systems are brought up intraday (see section *5.3.2* on how to rebuild the market image from the refresh service). Clients are recommended to build automatic recovery mechanism so as to enable timely recovery on their side.

The following describes the OMD-D site failover behaviour based on which clients can automate their systems to resume market data service against the OMD-D backup site:

- Multicast data including heartbeat from all real-time & refresh channels is unavailable
- Clients are unable to connect and log on RTS or, in the case of an already established RTS session, the TCP connection is disconnected or appears stale with no response from RTS servers
  Client who has connected to RTS on backup site is not affected.

- OMD-D starts broadcasting DR Signal with DR Status "1" (DR in progress) repeatedly during the execution of recovery procedure
- During the DR Status "1" (DR in progress), clients should clear all previously cached market data and prepare for any site failover operation if necessary.
- OMD-D starts broadcasting the DR Status "2" (DR completed) status repeatedly once OMD-D has finished the recovery procedure and is functioning normally. Refresh and real time services are ready for clients to rebuild the latest market image.
- After successful site failover, clients should follow the Interface Specifications and this Developer Guide to get the latest market snapshot from the refresh service before resuming receiving real-time data.
- In any event, client systems must be able to detect the unavailability of the primary site and the full readiness of the backup site, i.e. when DR Status "2" is received. For example, client systems which do not follow the recommended logic above but require "shutdown then restart" as part of their recovery procedure may not rely on DR Status "1" (DR in progress) for clearing all previously cached market data as this might have been done before the systems have reconnected to the OMD-D on the backup site. Nonetheless, the systems should still wait for DR Status "2" (DR completed) to start recovering through the refresh service as in the case of late client system startup when OMD-D is already disseminating data.
- The duration from the loss of all multicast data (i.e., primary site shutdown) until the dissemination of the DR Signal message with the DR Status "2" (DR completed) (i.e., service resumption in the backup site) could be as short as 5 minutes. Clients should use this 5-minute failover time as a design objective in their automated OMD-D site failover solution.

Please note that OMD-Index Feed service is independent from OMD-D service, a dedicated multicast channel for OMD-Index DR signal declares the status of OMD-Index Feed service. Clients subscribing OMD-Index Feed should follow the same procedures as above to process the index data received from the OMD Index Feed. It is rare but possible that DR Signal is disseminated for OMD-Index to indicate the failover of OMD-Index to DR site, but no DR Signal is disseminated for OMD-D which indicates that OMD-D keeps running in the Primary site without failover.

Please refer to the OMD-D Connectivity Guide for the two dedicated DR multicast channels assigned for OMD-D and OMD Index Feed.

This exceptional scenario is covered in the Readiness Test. Since the site failover processes of clients are likely to be slightly different if carried out in the production environment (for example, connection via different IP addresses), this scenario is also among the possible scenarios to be covered in a regular emergency drill.

## 10.10 Special Trading Condition

Clients' system should have the flexibility to handle the variations of OMD-D and OMD-Index data transmission pattern due to the following special trading conditions:

**Hong Kong Holiday which is not a Holiday in Mainland**

For OMD-Index, only index reference and real-time index data of non-Hong Kong indices will be disseminated via the respective channels and clients should expect receiving only heartbeat messages from all other OMD-Index channels.

No matter whether it is holiday in Mainland, OMD-D will be operated as usual to support the holiday trading day in the Hong Kong Derivatives Market on all Hong Kong holidays, except the holiday of New Year's Day.

# 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, Retransmission 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(mRtsBuffer, packet->mPktSize);
      }
      else
      {
        // check whether data is compressed or not
        if (packet->mode == 0x01) {
              // copy the packet header to uncompressed packet buffer
              char uncompressedPacketBuffer[2048];
              memcpy(uncompressedPacketBuffer, mRtsBuffer, sizeof(PacketHeader));

              // get the compressed data size
              // i.e. the packet size minus 16-byte packet header
              int compressedDataSize = packet->pktSize - sizeof(PacketHeader);

              // get the compressed data pointer
              const char* compressedData = mRtsBuffer + sizeof(PacketHeader);

              // decompress() will return the uncompressed data and size
              char* uncompressedData = uncompressedPacketBuffer + sizeof(PacketHeader);
              decompress(compressedData, compressedDataSize,
                                  uncompressedData, uncompressedDataSize);

              // change the packet pointer to uncompressed packet buffer
              packet = (PacketHeader *) uncompressedPacketBuffer;

              // update the uncompressed packet size
              packet->pktSize = uncompressedDataSize + sizeof(PacketHeader);
        }

        // Determine the kind of message(s) in the packet
        uint16_t msgType = packet.getMsgType();

        switch (msgType)
        {
          case LOGON_RESPONSE_TYPE:
          {
```

```
                LogonResponse *logonResponse
                                = (LogonResponse *)(mRtsBuffer + sizeof(PacketHeader));
                processLogonResponse(logonResponse);
                break;
              }
              case RETRANS_RESPONSE_TYPE:
              {
                RetransResponse* resp = (RetransResponse*) (mRtsBuffer + 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);
       updateOBPriceLevel(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);
    }
}
```

# APPENDIX F – Pseudo code for processing compressed multicast data

An example shows how clients process the compressed data.

```
void processMulticastData(const char* data, int len)
{
          PacketHeader* packet = (PacketHeader *) data;

          if (packet->pktSize != len) {
              // invalid packet size
              return;
          }

          If (LineArbitration(packet))
          {
              return;
          }

          if (packet->mode == 0x01) {

               // 0x01 = Data is compressed, need to decompress the data before processing

               // copy the packet header to uncompressed packet buffer
               char uncompressedPacketBuffer[2048];
               memcpy(uncompressedPacketBuffer, data, sizeof(PacketHeader));

               // get the compressed data size
               // i.e. the packet size minus 16-byte packet header
               int compressedDataSize = packet->pktSize - sizeof(PacketHeader);

               // get the compressed data pointer
               const char* compressedData = data + sizeof(PacketHeader);

               // decompress() will return the uncompressed data and size
               char* uncompressedData = uncompressedPacketBuffer + sizeof(PacketHeader);
               decompress(compressedData, compressedDataSize,
                                   uncompressedData, uncompressedDataSize);

               // change the packet pointer to uncompressed packet buffer
               packet = (PacketHeader *) uncompressedPacketBuffer;

               // update the uncompressed packet size
               packet->pktSize = uncompressedDataSize + sizeof(PacketHeader);
       }

       processPacket(packet);
}
```

# DOCUMENT HISTORY

| Version | Date of Issue | Comments |
|---|---|---|
| V1.0 | 2 Aug 2013 | First Distribution Issue under the Derivatives Trading system, HKATS |
| V2.0 | 11 Feb 2026 | First Version under Orion Derivatives Platform (ODP) |
| V2.1 | 28 Apr 2026 | Section 5.2.3.2<br>- Add a description for Instrument Key Elements and Instrument Class Key<br>Section 5.2.3.4<br>- Update the information about partitions assignment<br>Section 7<br>- Add a description for Implied order in Aggregate Order Book Update message<br>Section 10.7<br>- Bundle message 320 and 321 with other Status Data together<br>- Remove message 320 and 321 from Message Resent |