DEVELOPERS GUIDE

HKEX Orion Market Data Platform
Securities Market & Index Datafeed Products (OMD-C)

Version 1.15
14 July 2026

CONTENTS

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 – Securities Market & Index Datafeeds ("OMD-C").

This document contains guidelines and suggestions for OMD-C feed handler developers. All information included in this document is presented for reference only. Clients should design and implement their own OMD-C feed handlers that are tailored to their business and technical requirements.

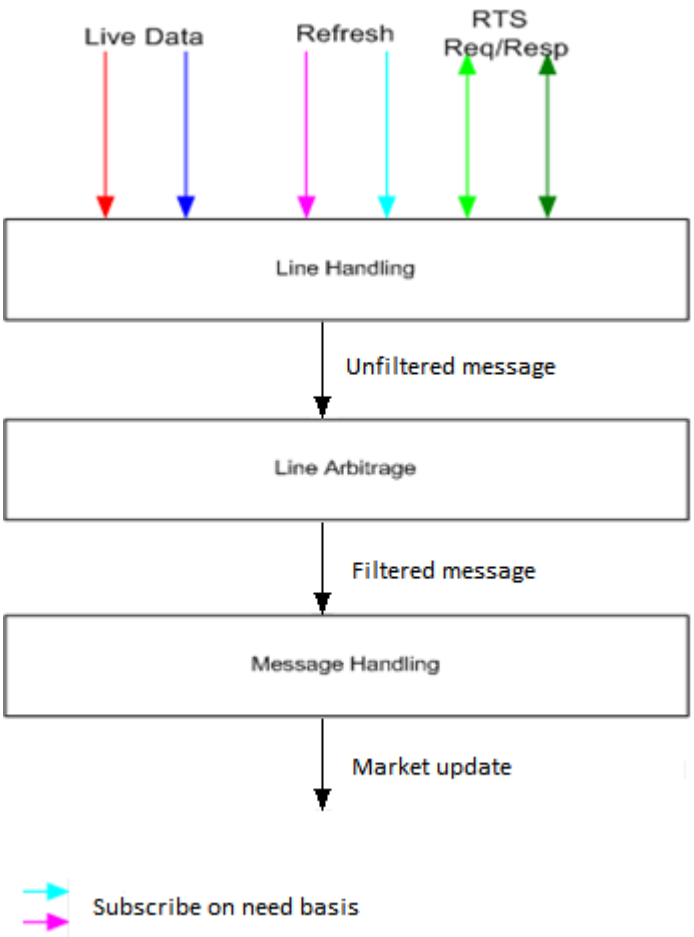
Its scope covers line arbitrage, packet and message processing, retransmission and refresh mechanisms, order book maintenance and exceptional handling procedures.

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

Table 1. Acronyms used in this document

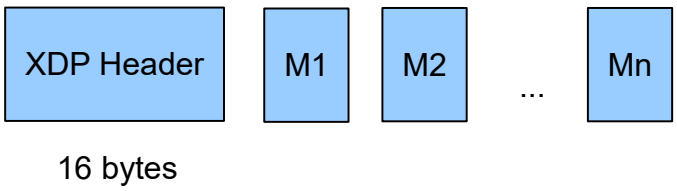
FH	Feed handler
HA	High Availability
MC	Multicast
RFS	Refresh Server
RTS	Retransmission Server
UDP	User Datagram Protocol
XDP	Exchange Data Publisher

Diagram 1. A Basic Client Application Layout



2 DATA STRUCTURE

Multicast packets 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 will only ever contain complete messages. A single message will never be fragmented across packets.

2.1 Packet Header

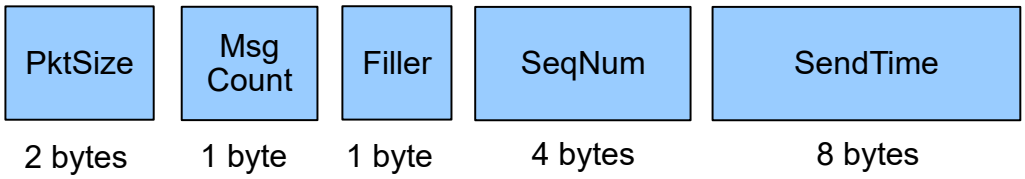
All packets disseminated from the OMD-C feed have a common packet header. This format is consistent across live, retransmission and refresh. XDP packet consists of 16-byte header followed by messages.

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

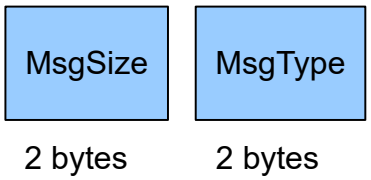
Heartbeats consist of a packet header with MsgCount set to 0 and do not increment the sequence number of the multicast channel. SeqNum in packet header is set to the sequence number of the previous message sent on the channel.

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

2.3 Message Header

The format of each message within a packet will vary 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-C Interface Specification for the full list of message type



2.4 Message Formats

Please refer to the HKEX OMD-C Interface Specification for details on the following message types:

- Control messages
- Retransmission

- Refresh
- Reference Data
- Status Data
- Order Book Data
- Trade and Price Data
- Value Added Data
- News
- Index Data
- Stock Connect Data

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. For example, it is currently used within Int64 fields of the Index Data (71) message.

The Int64 null representation is 0x8000000000000000 (Hex 2's complement) or -9223372036854775808 (Decimal).

4.2 Currency Values

Please refer to the Third Schedule of Rules of the Exchange in HKEX website for possible ISO-4217 Currency Codes used in OMD-C. Apart from the Currency Codes listed in the aforesaid Schedule, OMD-C will also use the Currency Code listed below:

Currency Code	Currency
CNH	Chinese Renminbi (Offshore)

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

The maintenance window of OMD Securities Market ("OMD-C") 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-C 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-C at or after 6:00am every business day to ensure that the data received from OMD-C are good for the start of the day.

Clients start at OMD-C startup time

- Clients subscribe to real-time multicast channels (Each OMD-C data product is delivered via a group of real-time multicast channels)

- OMD-C sends Sequence Reset message (100). Please refer to section ***10.4 Sequence Reset Message*** for processing details.

- OMD-C sends Reference Data messages below
  - Market Definition (10)
  - Security Definition (11)
  - Liquidity Provider (13)
  - Currency Rate (14)

- 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-C 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 ***10.4 Sequence Reset Message*** for details.
  - Sequence Reset message are sent individually on each channels. Client may experience long time difference between the Sequence Reset messages in different channels during OMD-C start up

Clients start after OMD-C startup time and missed sequence reset message and reference data

- Please refer to section ***6.2.2 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-C sets to every 2 seconds) on each channel when there is no line activity.

UDP multicast network/transport protocol is used in OMD-C 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-C 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
  - Gap Detection

- 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 B – 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-C is UDP multicast. The data in OMD-C 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-C 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 HKEX OMD-C 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 C – 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 processed 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 spool message 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 C – 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

Message carrying information about a particular instrument has a Security Code field. This field is unique instrument identifiers. The Security name, ISIN code, etc. are only carried in the Security Definition (11) message, so clients must associate the Security Code with instrument's characteristics during the reference data processing. The Security Code, once allocated for an instrument, does not change.

5.2.3.1 Process Fields Carrying Decimal Value

Message with fields carrying decimal value are usually defined with a fixed decimal place value. For example, PreviousClosingPrice in Security Definition (11) message is defined with 3 implied decimal places. There is exception to this, however, for some of the fields carrying decimal value are defined with variable decimal place value which is provided in a separate field in the same message. For example, CallPrice in Security Definition (11) messages is defined with variable decimal place value which is provided in DecimalsInCallPrice in the same message. Clients need to make use of value defined in DecimalsInCallPrice to obtain the actual CallPrice. For example, CallPrice = 1234567 with DecimalsInCallPrice = 5 will imply a Call Price of $12.34567 for the CBBC security.

5.2.3.2 Process Index Data and News

Index Status, Index Values and other index attributes in the Index Data (71) message corresponds to the time as specified in the field “IndexTime”. It does not correspond to the time provided in the packet header.

Similarly, the news headline and other news attributes in the News (22) message corresponds to the time as specified in the field “ReleaseTime”. It does not correspond to the time provided in the packet header.

As such, Client should process and refer to both fields “IndexTime” and “ReleaseTime” in Index Data (71) and News (22) message respectively for the correct time information.

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 HKEX OMD-C 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 that there is an outage in the system infrastructure.

Note that OMD-C sends heartbeats only when there is no market data being disseminated. When there is market data on the line, no heartbeat is available.

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	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-C is operating in the primary site or the backup site. See Section ***10.8 Site Failover*** on how to handle OMD-C site failover making use of the DR Signal message.

6 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, which depend on varies factors such as message gap size, recovery time/event and etc.

6.1 Retransmission Service

Both OMD-C Feed and OMD-Index Feed have their own dedicated RTSs. For small number of message gap, clients can recover lost messages using the RTS. The connection between the RTS and the client is reliable (TCP/IP). 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 RTS contains only a relatively small number of messages (50,000) from each broadcast stream. The RTS should not be thought of as a means of recovering intraday. 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 retransmission daily limit of 1000 requests for OMD-C 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, 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 message.

6.1.1 Multiple Retransmission Servers

Four RTSs, two on primary site and two on backup site, are all active and available for connection after OMD-C 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.

6.1.2 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	Optional
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

6.1.3 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.

6.1.4 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.

6.1.5 Retransmission Request

A retransmission request consists of a PacketHeader and a Retransmission Request (201).

Table 7. Retransmission Request Packet Header

PktSize	32
MsgCount	1
Filler	
SeqNum	Optional
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

6.1.6 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. Client may contact us for assistance.

6.1.7 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.

6.1.8 Retransmission Limits

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

Table 10. Retransmission System Limits for both OMD-C and OMD-Index

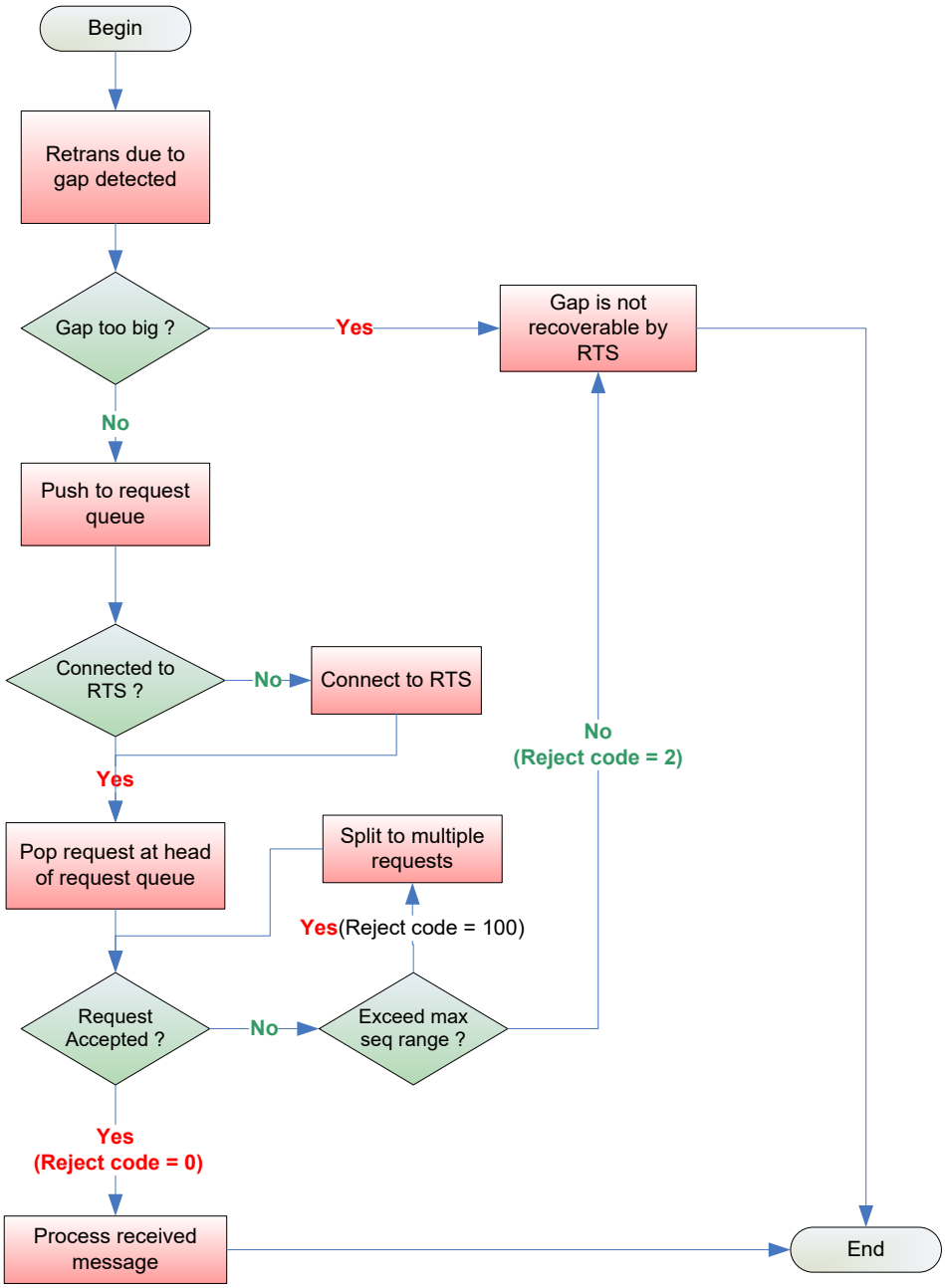
System Limit	Value
Last number of messages available per channel ID	50,000
Maximum sequence range that can be requested	10,000
Maximum number of requests per day	1,000
Logon timeout (seconds)	5
Heartbeat interval (seconds)	30

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.

6.1.9 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 D – Pseudo code for processing retransmission data*** for example on handling data from OMD-C Retransmission server.

6.2 Refresh Service

The OMD-C feed provides a refresh facility, 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 channel, the refresh data also come from two redundant lines, A and B. Clients can apply the Line Arbitration mechanism described in section ***5.2.2 Line Arbitration,*** except that there is no retransmission.

If clients connect to refresh multicast channel after OMD-C start up, they do not need to check any message gap before the first arrived packet. Clients can make reference to the sequence number of the first message and increment the next expected sequence number by 1 when processing incoming messages.

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

6.2.1 RFS Snapshot

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

6.2.2 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 where 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 refresh 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 refers to Section 10.4 Sequence Reset Message for details
8. Process the remaining cached real-time messages and resume normal processing.

Determine a full refresh snapshot

When subscribing to OMD-C 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

   If there is no message transmission (channel idle) in a refresh multicast channel, OMD-C sends Heartbeat message 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-C sends refresh snapshot at a regular interval. Heartbeat may be disseminated in between two rounds of refresh snapshot, 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 (e.g. no order/trade transactions in NASD or ETS market).

   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-C 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

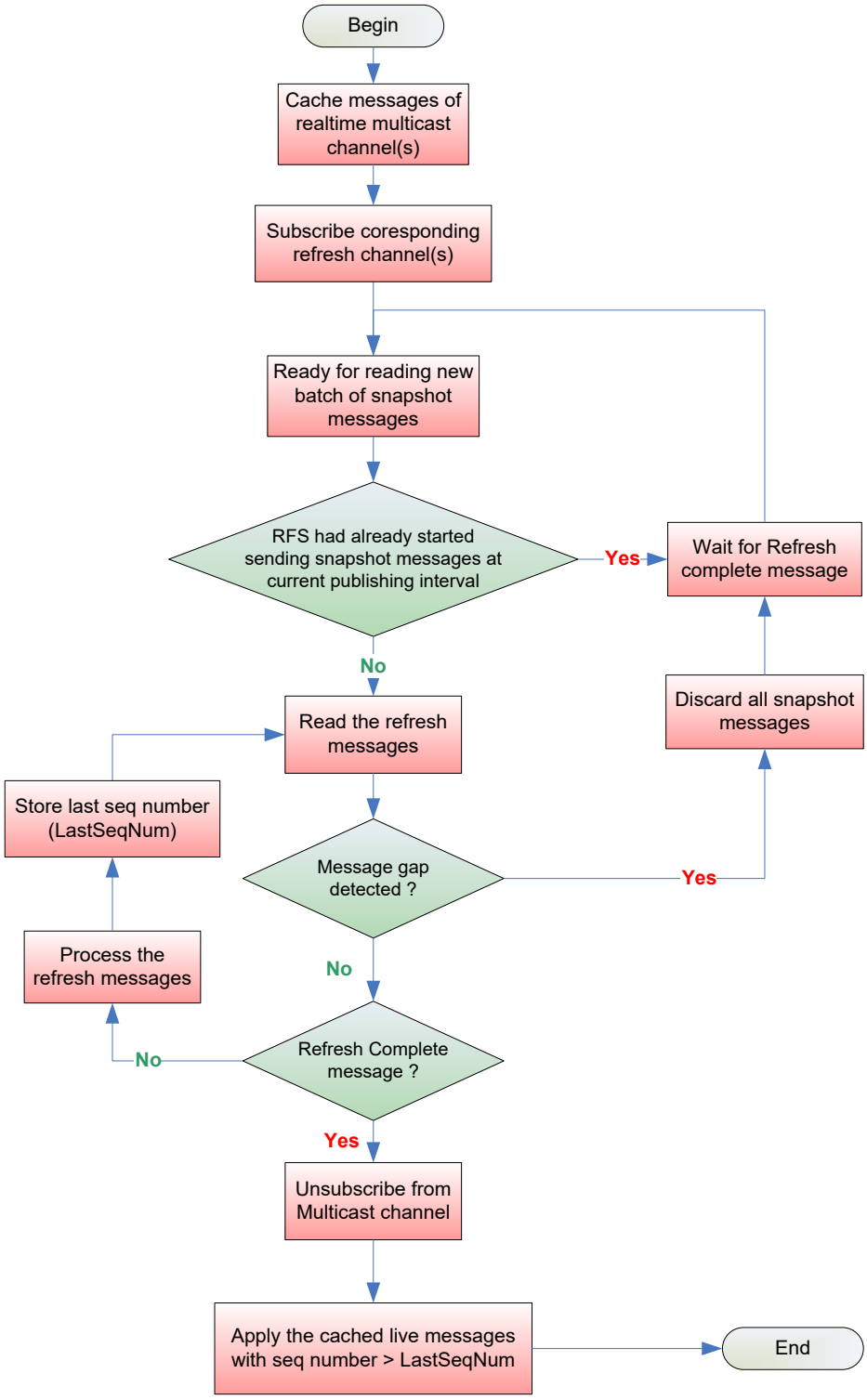
There is no TCP retransmission for refresh. Clients must monitor for packet loss on the refresh channels and wait for the next snapshot if loss is detected.

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 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 E – Pseudo code for processing Refresh snapshot packet*** for example on handling data from OMD-C RFS.

7 RACE CONDITIONS

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

For example, the Trading Session 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.

8 AGGREGATE ORDER BOOK MANAGEMENT

Book updates are sent by OMD-C via Aggregate Order Book (53) 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. Note - Potential level adjustments must be carried out after each single entry in Aggregate Order Book message.
- If a new book entry causes the bottom entry of a book to be shifted out of the book,
  1. If the shifted out entry is within 10 Price level, OMD-C will send an explicit deletion entry (Explicit delete).
  2. If the shifted out entry is outside the 10 Price level, clients must delete the excess entry (Implicit delete).
  3. If the book shrinks again, the server resends the entries that have temporarily fallen out.
- If a match causes an order to be removed so that there are now less than 10 levels visible, then the server will also automatically send the additional level(s) that are now revealed.
- If a clear aggregate order book message is received, client should clear all entries of the order book.

Please refer to *Section 5 – Aggregate Order Book Management in the Interface Specification of HKEX Orion Market Data Platform Securities - Market & Index Datafeed Products* for different scenarios on how OMD-C sends Aggregate Order Book message.

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

9 FULL ORDER BOOK MANAGEMENT

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

Table 12. Order Update Message Types

Message Type	Name
30	Add Order
31	Modify Order
32	Delete Order
33	Add Odd Lot Order
34	Delete Odd Lot Order

General Rules

- Clients should be able to uniquely identify the order (OrderID is the unique identifier per security)
- Determine the price and size of an order

Order Types

Table 13. Order Types

Order Type	Description
Market (1)	Executed at current market price
Limit (2)	Contains an execution condition to buy or sell at a specified price or better

Note

The price field of Market Orders is always set to 0.

10 EXCEPTION HANDLING AND SPECIAL TRADING CONDITION

Listed below are some common exception handling procedures or special trading conditions that clients must be capable of when subscribing to OMD:

Exception Handling
- Late connection
- Intra-day refresh
- Client application restarts
- Sequence Reset Message
- OMD-C restarts before market open
- OMD-C Component failover
- Site failover

Special Trading Conditions
- Hong Kong Holiday
- Typhoon Signal Number 8 or above / Black Rainstorm

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 disaster recovery 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 ***6.2.2 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, 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 section **6.2.2 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 section **6.2.2 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 section **6.2.2 Processing a Refresh** – for handling sequence reset message from Refresh channels.*

This exceptional scenario is covered in the Readiness Test.

10.5 OMD-C Restarts Before Market Open

In case of OMD-C 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-C Restarts After Market Open (Intraday Restart)

When OMD-C fails during trading hours, besides failing over to the backup site to resume service, OMD-C may be recovered by Intraday Restart where OMD-C will be shut down and then restarted. When OMD-C 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-C is subsequently restarted. Upon receipt of the notification from us, all clients should clear all their internal cache and reconnect to OMD-C afresh in the same way as their systems start up late and connect to OMD-C only after the cash 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 ***6.2.2***) to establish the latest market status.

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

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

10.7 OMD-C Component Failover

Securities Full Order Book Channels, Securities Level 2 Streaming Channels, Securities News Channel and Securities Market Odd Lot Order Channel (Streaming Channels)

Component failover on Streaming Channels would only impact one of the OMD-C lines. In case no live data can be received in one of the OMD-C lines (say line A), there is no impact to clients as OMD-C would continue publishing data via the alternative line (line B). Clients can reapply the line arbitration as usual when OMD-C 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.

Securities Reference Data Channels, Trading Session Status Channel, Securities Level 2 Conflated Channels and Securities Market Broker Queue Channel ("Conflated Channels")

Component failover could cause Conflated Channels to experience an outage for a few seconds. 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.

Recovery Type	Message Type List	Remarks
Latest Value Update	Trading Session Status (20) Security Definition (11) Security Status (21) Nominal Price (40) Indicative Equilibrium Price (41) Reference Price (43) Yield (44) Broker Queue (54) Order Imbalance (56) Statistics (60) Closing Price (62) VCM Trigger (23)	The latest value of Trading Session Status will be resent. For securities having updates on the remaining messages during the outage, messages with latest values will be resent.
Aggregate Order Book	Aggregate Order Book Update (53)	For securities having updates during the outage, Aggregate Order Book Update (53) 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. For securities without updates during the outage, Aggregate Order Book Update (53) message with update action "Order Book Clear" will not be sent. Clients can keep building the aggregate order book as usual.
Trade Ticker	Trade Ticker (52)	Duplicate Trade Tickers (52) messages will be resent in Securities Standard (SS) . Client should check the ticker ID when processing the Trade Ticker (52) message to avoid duplication.

Securities Value Add Data Channel and Stock Connect Market Information Channel

In case of the component failover of the above channels, the dissemination of Market Turnover (61) message and Stock Connect Data will be resumed at the next update period.

This exceptional scenario is covered in the Readiness Test.

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)*

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-C 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-C on backup site for client actions.

A dedicated multicast channel is assigned to transmit DR Signal. During normal days, OMD-C 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 connect to the RTSs on backup site for Retransmission Service.

When OMD-C 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-C is considered operating normally on the backup site and the latest market snapshots can be obtained from the Refresh channels. OMD-C will continue transmitting the DR Signal with DR Status "2" until end of business day.

Clients are advised to clear all OMD-C 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 ***6.2.2*** 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-C site failover behaviour based on which clients can automate their systems to resume market data service against the OMD-C backup site:

- Multicast data including heartbeat from all real-time & refresh channels is unavailable
- Clients are unable to connect and log on RTS on primary site or, in the case of an already established RTS session, the TCP connection is disconnected or appears stale with no response from RTS. Client who has connected to RTS on backup site is not affected.
- OMD-C 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-C starts broadcasting the DR Status "2" (DR completed) status repeatedly once OMD-C 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-C 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-C 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-C site failover solution.

Please note that OMD-Index Feed service is independent from OMD-C 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-C which indicates that OMD-C keeps running in the Primary site without failover.

Please refer to the OMD-C Connectivity Guide for the two dedicated DR multicast channels assigned for OMD-C 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.

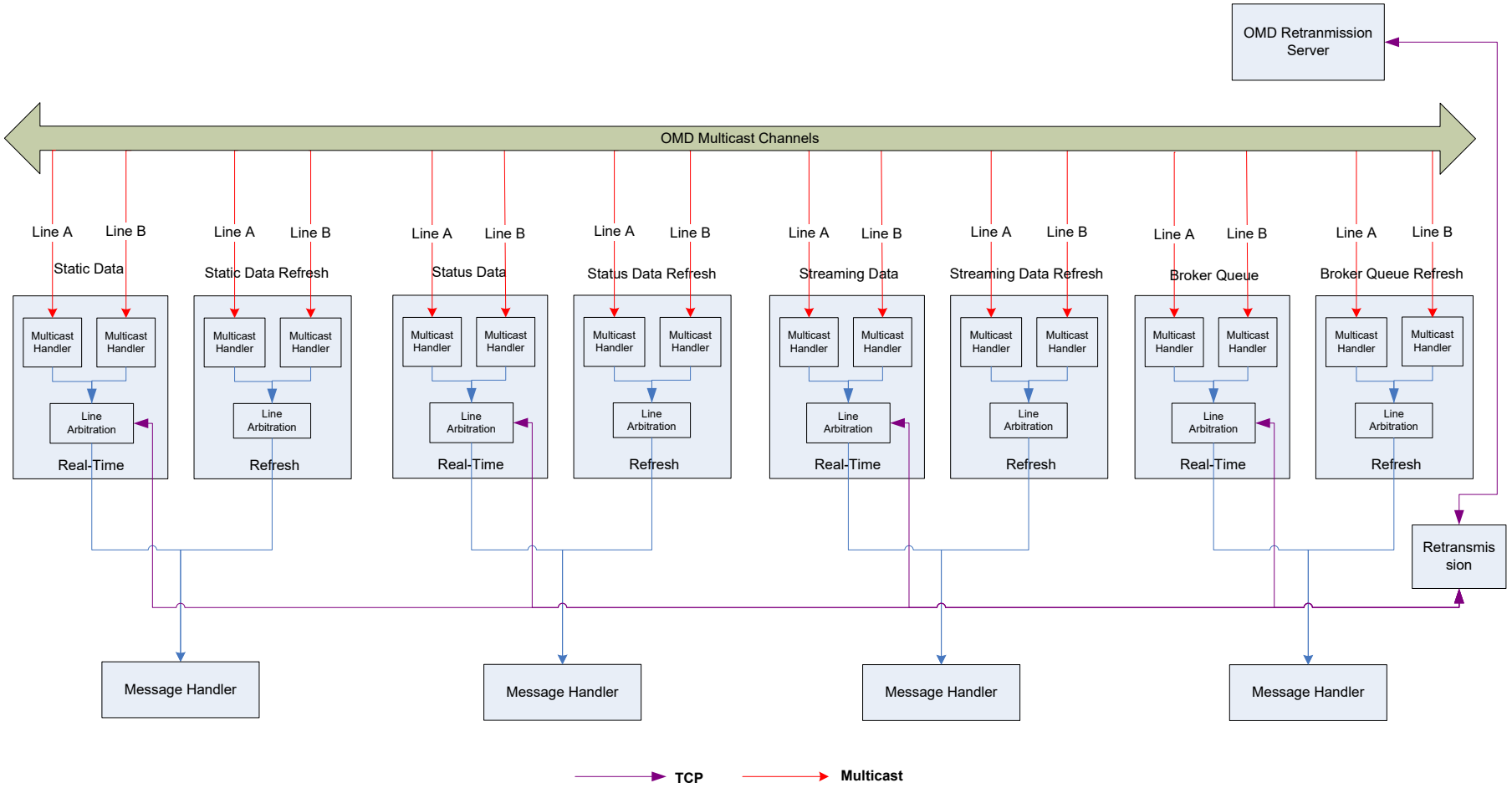
10.10 Special Trading Condition

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

Hong Kong Holiday which is not a Holiday in Mainland

On Hong Kong holidays which are not holidays in the Mainland, OMD-C and OMD-Index will be operated as usual for the clients to connect and subscribe. However, 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-C and OMD-Index channels.

APPENDIX A – Example of network diagram to OMD



Module	Description	Network Type	Number of connection
Retransmission	Module that handle Retransmission in case there is any gap	TCP	1
Multicast Handler	Module that read data from multicast channels. - Online message (Line A and Line B) - Refresh message (Line A and Line B) in case there is any missing packet that cannot be recovered from RTS	UDP	Number of MC x 4
Line Arbitration	Module that handle line arbitration.	N/A	
Message Handler	*Module that process OMD-C messages*	*N/A*	

Note

The above network diagram illustrates one of the possible network connections to OMD. Clients may have different designs when subscribing to different OMD-C multicast channels.

APPENDIX B – 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 C – Pseudo code of Line Arbitration

An example shows how clients process a packet received from OMD. 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 D – Pseudo code for processing retransmission data

An example shows how clients process incoming data from OMD-C 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(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 E – Pseudo code for processing Refresh snapshot packet

An example shows how clients process refresh snapshot data from OMD-C 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 F – 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);
      }
}
```

DOCUMENT HISTORY

<table>
<tr><th>Version</th><th>Date of Issue</th><th>Comment</th></tr>
<tr><td>V1.0</td><td>3 July 2012</td><td>First Distribution Issue</td></tr>
<tr><td>V1.1</td><td>9 May 2013</td><td>Section 5.2.4
• Added some clarifications in the first paragraph.
Section 5.3.2.2
• Added suggested exceptional handling when sequence reset is received in the middle of a refresh cycle.
• Added a note to “Determine a full refresh snapshots”.
Section 9.4
• Added some clarification under “Sequencing of events”.</td></tr>
<tr><td>V1.2</td><td>25 Sep 2013</td><td>Section 5.3.2.2
• Removed 4th condition in determining a full refresh snapshot.
Section 9.6
• Added some clarifications descriptions.
Section 9.8
• Newly added section on speical trading conditions including Hong Kong holidays and Typhoon / Black Rainstorm signals delaying market open</td></tr>
<tr><td>V1.3</td><td>4 Apr 2014</td><td>Section 9.7
• Notes on the behaviours observed during OMD-C site failover</td></tr>
<tr><td>V1.4</td><td>19 Dec 2014</td><td>Section 9
• Elaborate exceptional scenarios (newly added Section 9.6 and renumbering of subsequent subsections) and state the arrangements to facilitate client tests on the exceptional scenarios</td></tr>
<tr><td>V1.5</td><td>6 Feb 2015</td><td>Section 9.6
• Add notes on handling Intra-day Restart</td></tr>
<tr><td>V1.6</td><td>19 Feb 2016</td><td><table><tr><th>Effective Date</th><th>Changes</th></tr><tr><td>6 Mar 2017</td><td>Introduction of a new Disaster Recovery (DR) mechanism
• Section 5.2.5 - New section to introduce the new Disaster Recovery Signal (105) message
• Section 9.8 - Add paragraphs to provide guidance on how to handle the new DR mechanism</td></tr></table></td></tr>
<tr><td>V1.7</td><td>21 Sep 2016</td><td><table><tr><th>Effective Date</th><th>Changes</th></tr><tr><td>6 Mar 2017</td><td>Introduction of a new Disaster Recovery (DR) mechanism
• Section 9.8 - Add paragraphs to elborate more on DR Signal handling for mutliple datafeeds</td></tr></table></td></tr>
<tr><td>V1.8</td><td>4 Jan 2017</td><td><table><tr><th>Effective Date</th><th>Changes</th></tr><tr><td>6 Mar 2017</td><td>Introduction of a new Disaster Recovery (DR) mechanism
• Section 9.8 - Remove paragraphs related to the obsoleted DR mechanism based on Sequence Reset message</td></tr></table></td></tr>
<tr><td>V1.9</td><td>27 Mar 2017</td><td><table><tr><th>Effective Date</th><th>Changes</th></tr><tr><td>Immediate</td><td>Clarifications
• Section 5.1 - Revise description to clarify Start of Day</td></tr></table></td></tr>
</table>

<table>
<tr><th>Version</th><th>Date of Issue</th><th>Comment</th></tr>
<tr><td>V1.10</td><td>9 Mar 2018</td><td>
<table>
<tr><th>Effective Date</th><th>Changes</th></tr>
<tr><td>30 April 2018</td><td>OMD-C Reference Data Enrichment :
• Section 5.2.3.1 – New section to highlight the handling of some new fields with variable decimal place value defined in another field</td></tr>
<tr><td>Immediate</td><td>Update of Information:
• Section 4.2 – Add CNH currency
• Section 5.3.1.3 – Revise the description of Logon Session Status 5 in Logon Response (102) message
• Section 5.3.2.2 – Clarify behaviour of Heartbeat messages between refresh cycles and remove obsolete requirement in processing refresh recovery
• Section 6 – Revise the example of race conditions</td></tr>
</table>
</td></tr>
<tr><td>V1.11</td><td>12 Sep 2018</td><td>
<table>
<tr><th>Effective Date</th><th>Changes</th></tr>
<tr><td>Immediate</td><td>Clarifications:
• Section 4.2 – Revise description for Currency Value</td></tr>
</table>
</td></tr>
<tr><td>V1.12</td><td>20 May 2020</td><td>
<table>
<tr><th>Effective Date</th><th>Changes</th></tr>
<tr><td>12 Apr 2021</td><td>OMD-C Resilience Model Enhancement:
• Section 6.1.1 – Updated number of alternative RTS
• Section 10.7 – Revised streaming channels failover behaviour
• Section 10.8 – Revised sequencing of events</td></tr>
<tr><td>Immediate</td><td>Clarifications:
• Section 5.1 – Revised remarks
• Section 5.2.3.2 – Add new session for index data handling
• Section 5.2.4 – Revised description
• Section 6 – Changed Section 5.3 to Section 6
• Section 6.2.2 – Revised handling on first message is heartbeat
• Section 10.2 – Revised sequencing of events
• Section 10.4 – Revised sequencing of events
• Section 10.7 – Revised channels failover behaviour</td></tr>
</table>
</td></tr>
<tr><td>V1.13</td><td>24 Apr 2023</td><td>
<table>
<tr><th>Effective Date</th><th>Changes</th></tr>
<tr><td>Immediate</td><td>Housekeeping:
• Sections 6.1.6, 6.1.8, 10 and 10.6 – Removed the wording “HKEX” or “HKEX-IS”
• Section 1 – Added description for target readers</td></tr>
</table>
</td></tr>
<tr><td>V1.14</td><td>21 May 2024</td><td>
<table>
<tr><th>Effective Date</th><th>Changes</th></tr>
<tr><td>Upon the launch of OMD Index Feed Enhancement</td><td>OMD Index Feed Enhancement:
• Section 6.1 – Clarification on RTS daily limit
• Section 6.1.1 and 6.1.8 – Update for OMD Index Feed
• Section 10.7 – Remove OMD Index Feed
• Section 10.8 and 10.9 – Update for component failover and DR Signal for OMD Index Feed
• Section 10.10 – Update for OMD Index Feed under special trading condition</td></tr>
</table>
</td></tr>
</table>

Version	Date of Issue	Comment
V1.15	14 July 2026	<table><tr><th>Effective Date</th><th>Changes</th></tr><tr><td>Immediate</td><td>Housekeeping: Section 10.10 – Removed the typhoon and black rainstorm scenario in special trading condition </td></tr></table>