LME
An HKEX Company

LMEsource v4

Developer Guide

THE LONDON METAL EXCHANGE

10 Finsbury Square, London EC2A 1AJ | Tel +44 (0)20 7113 8888

Registered in England no 2128666. Registered office as above.

LME.COM

1 Document Control

1.1 Change History

Version	Change Date	Summary of Change
1.0	28 Jan 22	Released
1.1	23 Nov 22	Added 5.3 Fragmented Messages Added 8.7 LMEsource Component Failover for Non-Electronic channels

1.2 Associated Documents

Title
LMEsource Client Interface Specification

2 Introduction

This document contains guidelines and suggestions for LMEsource Market Data Platform feed handler developers. All the information included in this document is for reference only. Clients should design and implement their own LMEsource feed handler that is tailored to their specific business and technical requirements.

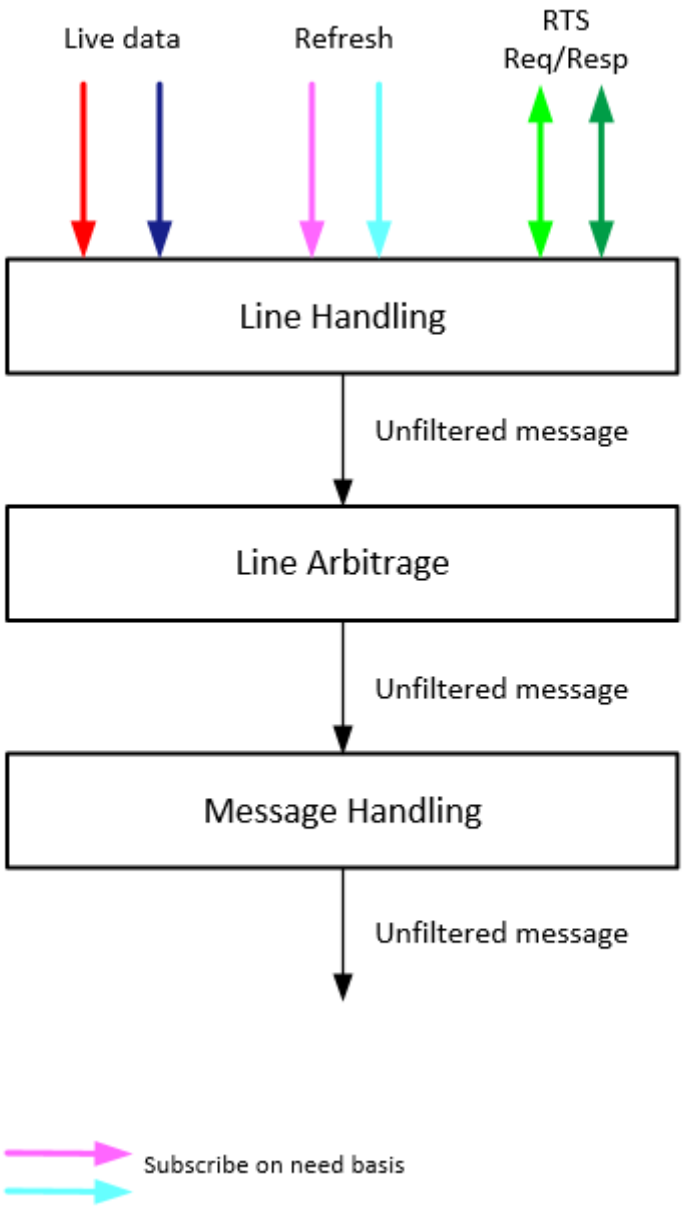
The scope of this document covers line arbitrage, packet and message processing, retransmission and refresh mechanisms, order book maintenance and exception handling.

The purpose of this document is to answer questions that developers may have after reading the LMEsource Client Interface Specification. It shows examples of usage and code snippets to help developers understand the logic behind the market data disseminated from LMEsource.

2.1 Acronyms used in this document

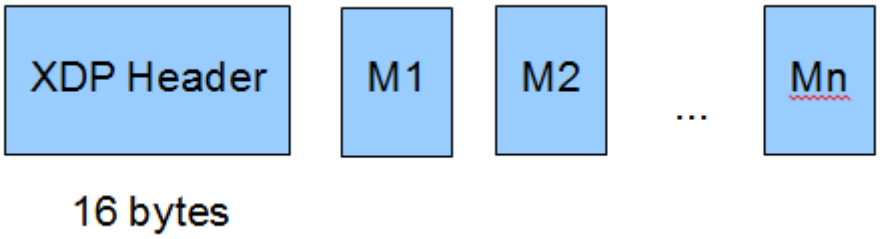
Acronym	Meaning
FH	Feed handler
HA	High Availability
MC	Multicast
RFS	Refresh Server
RTS	Retransmission Server
UDP	User Datagram Protocol
XDP	Exchange Data Publisher

2.2 Basic Client Application Layout



3 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.

3.1 Packet Header

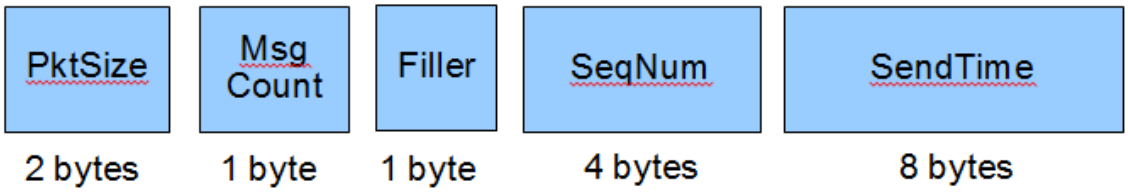
All packets disseminated from the LMEsource feed have a common packet header. The format of the packet header is consistent across live, retransmission and refresh services. An XDP packet consists of a 16-byte header followed by zero or more messages.

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.

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

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 microsecond.

Field	Offset	Length	Format	Description
				During BST, the date value in this field will have the previous day's date between the times of 00:00:00 BST and 00:59:59:999 BST.



3.2 Heartbeats

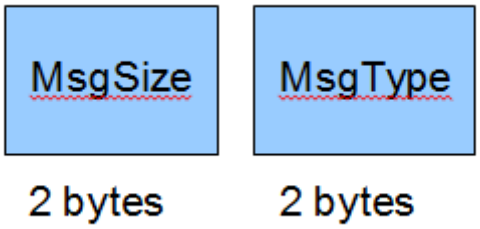
Heartbeats consist of a packet header with MsgCount set to 0 and do not increment the sequence number of the multicast channel. SeqNum in the packet header of Heartbeats is set to the sequence number of the previous message sent on the channel.

The Heartbeat message syntax is identical across LMEsource services.

3.3 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 LMEsource Interface Specification for the full list of message type and the layout of each message type



3.4 Message Formats

Please refer to the LMEsource Client Interface Specification for details on the following message types:

- Control messages
- Retransmission
- Refresh
- Instrument Reference Data
- Security Status Data
- Order Book Data
- Trade and Price Data

4 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 the MsgCount field in the packet header.

5 Field Attributes

5.1 Null Values

From time to time certain fields cannot be populated and specific values are used to represent null. The following table shows the values used to represent null for different data types:

Format	Null representation (Hex 2's complement)
Int8	0x80
Int32	0x8000 0000
Int64	0x8000 0000 0000 0000
Uint8	0xFF
Uint16	0xFFFF
Uint32	0xFFFF FFFF
Uint64	0xFFFF FFFF FFFF FFFF

5.2 Currency Values

Please see ISO 4217 for a full list of possible currency codes. Currently, the system uses the following codes:

'USD' – US dollars

'GBP' – GB Pounds

'EUR' – Euro

'JPY' – Japanese Yen

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

5.3 Fragmented Messages

Data that cannot be sent in a single message will be fragmented into multiple messages, for example, Reference Prices Carries (402) and Reference Volatility Prices (404).

These message types include a RemainingPrice and RemainingVolatilities field which carries the number of record available to be published. The value remaining decreases along with messages published. When the value remaining equals the number of record in the message it indicates that the message is the last in the series.

Example 1: Number of available records 50, message capacity 100.

The following message will be published:

Message Serial Number	Remaining Records	Number of Records	Remark
1	50	50	One message is sufficient to carry all records

Example 2: Number of available records 350, message capacity 100.

The following messages will be published.

Message Serial Number	Remaining Records	Number of Records	Remark
1	350	100	Not the last message
2	250	100	Not the last message
3	150	100	Not the last message
4	50	50	The value in the two fields is equal which indicates this is the last message of the 4 messages serial.

6 Message Processing

Each multicast channel maintains its own session. A session is limited to one business day. During the day, the message sequence number is strictly increasing and therefore unique within a channel. Messages from the LME base metals market and LMEprecious market must be processed independent of each other.

6.1 Start of Day

The maintenance window of LMEsource starts immediately after the system shuts down for the day until 00:30 the next business day. During the maintenance window, there could be system maintenance and housekeeping operations where LMEsource 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 LMEsource at or after 00:30 every business day to ensure that the data received from LMEsource are good for the start of the day.

During the startup of LMEsource server (00:30), Sequence Reset (100) messages and a 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 LMEsource server. Clients should therefore build in the flexibility in their feed handlers to handle such situations. For details of processing Sequence Reset (100) messages, please refer to section 8.4

Clients may receive multiple Sequence Reset messages during the start of day as a result of LMEsource restarting. The general handling should be to reset the next expected sequence number and clear all cached data for all instruments. Please refer to section 8.5

After receiving the Sequence Reset message, clients should also check the sequence number of the next incoming packet. If the sequence number is not equal to 1, it indicates that there is packet loss. Please refer to section 8.4 for details.

Clients start after LMEsource start-up time and miss sequence reset message and reference data

Please refer to section 6.3.2.2 for the handling of a late connection.

6.2 Normal Transaction

Normal message transmission is expected between when the market opens for trading and when the market closes. Heartbeats are sent regularly on each channel when there is no line activity. Currently LMEsource sends heartbeats every 2 seconds.

The UDP multicast network/transport protocol is used in LMEsource and the data is sent to different broadcast streams (known as multicast channels).

UDP is not a reliable transport protocol and therefore packets may be lost or arrive out of sequence. To minimize the risk of losing a packet, the data on each channel is published on two redundant lines, A and B.

Clients receive and process LMEsource 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

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

6.2.1 Receive Multicast

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

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

6.2.2 Line Arbitration

The network/transport protocol used in LMEsource is UDP multicast. The data in LMEsource 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 arrive out of sequence. Two lines with identical data minimises the risk of losing a packet; however the risk still exists.

Note

1. Clients should not prioritise 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.

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

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

Note

MC: Message Count in a Packet

SN: Sequence Number

Clients receiving LMEsource 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 LMEsource Client Interface Specification.

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 by the message count in the packet header

- When a subsequent 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 for processing retransmission data for an example on detecting a message gap or duplicate packet.

6.2.2.1 Possible approaches for handling a 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 waiting 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 wait to be filled either from the redundant line or the retransmission server.
4. Check if the gap in spooled messages can be filled at a regular interval.
5. If the gap cannot be recovered within a specified time, clients should recover from refresh server.

Please refer to APPENDIX B – Pseudo code for processing retransmission data 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.

6.2.3 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 LMEsource Client Interface Specification for the multicast heartbeat interval currently set in LMEsource), then it indicates an outage of data transmission.

Note that LMEsource 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 a heartbeat packet, clients should ignore this packet in their gap detection. Otherwise, clients may fail to detect an actual message gap.

Time	Packet sent from LMEsource	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 indicates a message gap and the client should recover the lost message
T5	104	104	
T6	105	105	
T7	106	106	

6.2.4 Process Disaster Recovery Signal Message (105)

The Disaster Recovery (DR) Signal message indicates to clients whether LMEsource is operating in the primary site or the backup site. See Section 8.8 on how to handle LMEsource site failover making use of the DR Signal message.

6.3 Recovery

Since UDP multicast is not a reliable protocol there is a risk of packet loss. 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 etc.

6.3.1 Retransmission Service

For small message gaps, clients can recover lost messages using the retransmission service (RTS) which clients connect to using 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, see section 6.3.1.6.

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

Clients can have only one connection with the RTS.

Refer to the LMEsource Client Interface Specification for details of system limits.

Note

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

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

6.3.1.1 Secondary Retransmission Service

There is a secondary RTS which would be used in the event of problems being encountered with the primary RTS. This is a part of the High Availability design and is meant to provide customers with a seamless service in case of failure of the primary RTS. As part of the High Availability design, users will connect into a load balancer behind which multiple RTS servers will be running in an Active state. If there is a problem with one of the RTS servers users will be redirected to another server. Please see the LME systems connectivity guide for network connectivity details.

6.3.1.2 RTS Logon

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

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.

Logon Request Message	
MsgSize	16
MsgType	101
Username	Username to logon in plain text (Note: Please pad with NULLs after the username to fill up the 12 characters field)

6.3.1.3 RTS Logon Response

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

Logon Session Status	
0	Session Active
5	Invalid username
100	User already connected

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

6.3.1.4 RTS Heartbeats

To determine the health of the client connection on the TCP/IP channel, the RTS will regularly send heartbeats to the client. 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, the RTS will disconnect the session.

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

6.3.1.5 RTS Request

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

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.

Retransmission Request Message	
MsgSize	16

Retransmission Request Message	
MsgType	201
ChannelID	Dependent upon 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

6.3.1.5.1 Example of Retransmission Request

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

Channel	Packet Sequence Number	Message	Message received	Message Gap (Y/N)
1	101	Msg1 Msg2 Msg3	Msg1 (101) Msg2 (102) Msg3 (103)	N
1	104	Msg4 Msg5 Msg6	Msg4 (104) Msg5 (105) Msg6 (106)	N
1	109	Msg7 Msg8	Msg7 (109) Msg8 (110)	Y

The client application should send the following Retransmission Request Message to recover missing messages (Seq #107-108):

Retransmission Request Message	
MsgSize	16
MsgType	201
ChannelID	1
Filler	

Retransmission Request Message	
BeginSeqNum	107
EndSeqNum	108

6.3.1.6 RTS Response

After the client sends 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:

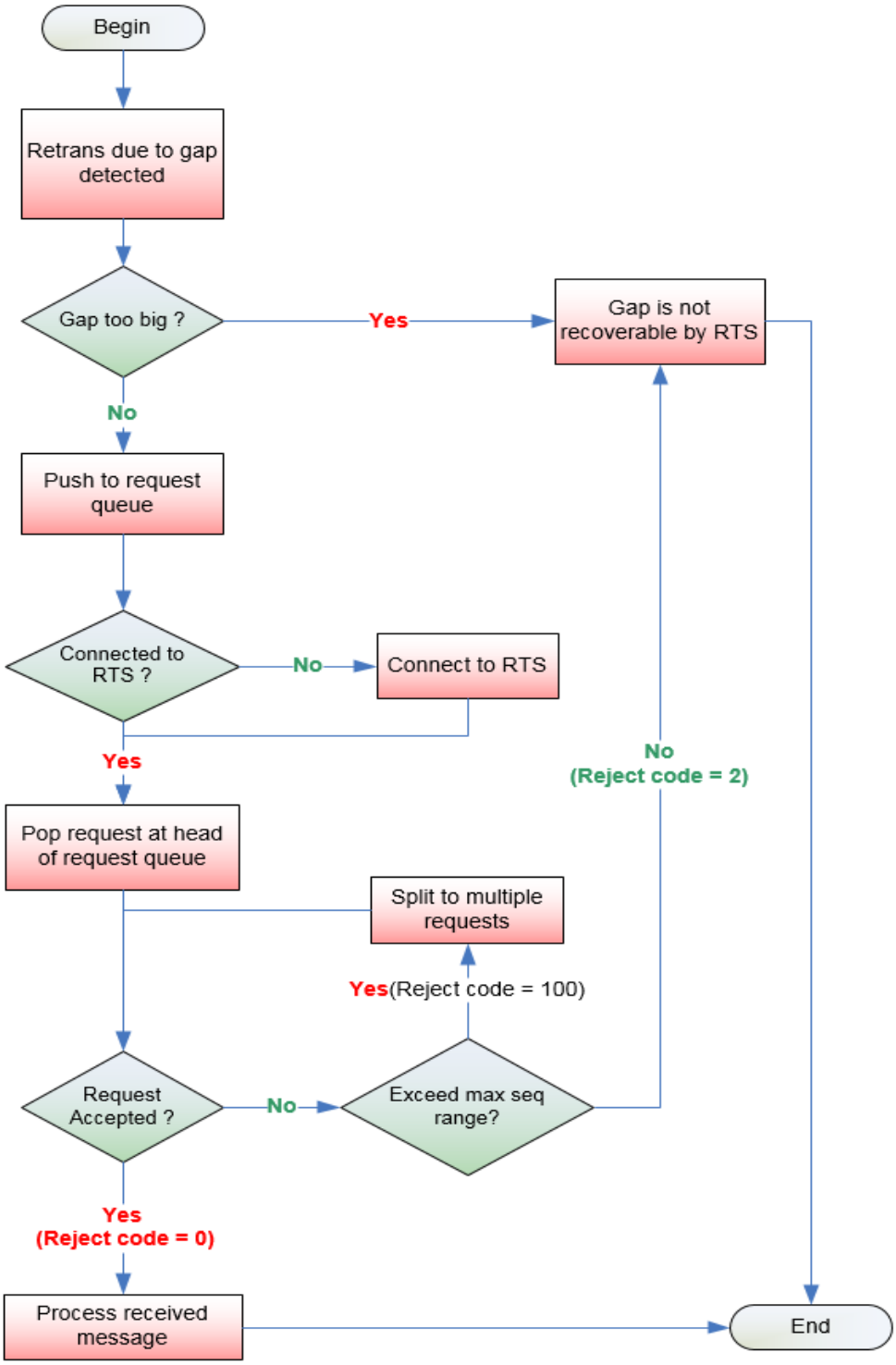
Retransmission Response Status	Meaning
0	Request accepted
1	Unknown/Unauthorised Channel ID
2	Messages not available
100	Exceeds maximum sequence range
101	Exceeds maximum requests in a day

It is very important to stop sending retransmission requests for the current day after being rejected with reason 101. Clients may contact the LME for assistance.

6.3.1.7 RTS Message

Upon receiving a 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.

6.3.1.8 Processing of RTS retransmission data



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

Please refer to APPENDIX B – Pseudo code for processing retransmission data for an example on handling data from LMEsource Retransmission server.

6.3.2 Refresh Service

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

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

The refresh packets are disseminated via dedicated multicast streams.

Similar to real time data transmission, the refresh data is published over two redundant lines, A and B. Clients can apply the Line Arbitration mechanism described in section 6.2.2, 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

6.3.2.1 RFS Snapshot

Please refer to the Refresh Service in the LMEsource Client Interface Specification for the coverage of snapshot data.

6.3.2.2 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

6.3.2.2.1 Connectivity

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

1. Live feed multicast
2. Refresh feed multicast

6.3.2.2.2 Synchronization

1. Subscribe to the real time multicast 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. Unsubscribe from the refresh multicast channel.
6. 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 8.4 for details.
7. Process the remaining cached real-time messages and resume normal processing.

6.3.2.2.3 Determine a full refresh snapshot

When subscribing to LMEsource 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 before the next fresh cycle begins, LMEsource sends Heartbeat messages at a regular time interval in the interim period. Clients wait for the full refresh snapshot starting with a message other than Heartbeat and ending with the arrival of Refresh Complete message.

2. The first message received is a Refresh Complete message

   Clients ignore the first Refresh Complete message and the subsequent Heartbeat message(s) in a refresh multicast channel. The next refresh snapshot starts with a message other than Heartbeat 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 messages at the moment. Simply skip message(s) until a Refresh Complete message is received. Usually, LMEsource sends a refresh snapshot at regular intervals. A 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 the 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 Clients listen to the refresh channels for the latest images, they may receive a 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 activity.

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”), LMEsource is running normally and the Clients have not missed any packets in the real-time channel.

6.3.2.2.4 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 to the refresh of channels that serve reference data**. Only after the refresh of the reference data is received, should clients 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.

6.3.2.3 Exception Handling

6.3.2.3.1 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 whilst processing Refresh messages:

1. Discard the Refresh messages received during the current Refresh cycle
2. Reset the next expected sequence number to 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


Begin
Cache messages of realtime multicast channel(s)
Subscribe to corresponding refresh channel(s)
Ready for reading new batch of snapshot messages
RFS had already started sending snapshot messages at current publishing interval
Yes
Wait for Refresh complete message
No
Read the refresh messages
Discard all snapshot messages
Store last seq number (LastSeqNum)
Message gap detected ?
Yes
Process the refresh messages
No
No
Refresh Complete message ?
Yes
Unsubscribe from Multicast channel
Apply the cached live messages with seq number > LastSeqNum
End

Please refer to APPENDIX C – Pseudo code for processing a refresh snapshot packet for an example on handling data from the LMEsource RFS.

7 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.

8 Exception Handling and Special Trading Condition

The following sections cover some common exceptions handling procedures that clients must be capable of when subscribing to LMEsource:

In order to facilitate clients' verification of their exception handling ability, some of the exceptional scenarios are included in the Readiness Test.

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. LME will issue a client notice to announce the schedule and coverage of a regular rehearsal in advance.

8.1 Late Connection / Start-up 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 6.3.2.2 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.

8.2 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.

8.2.1 Sequencing of events

1. Clear all cached data on that channel.
2. Cache real time messages in the multicast channel that previously experienced packet loss.
3. Listen to the corresponding refresh multicast channel and wait for the next snapshot (refer to section 6.3.2.2).
4. Process all refresh messages until the arrival of a Refresh Complete message.
5. Store the LastSeqNum sequence number provided in the Refresh Complete message.
6. Disconnect from the refresh multicast channel.
7. Process the cached real time messages with sequence numbers greater than the LastSeqNum.

The client will now maintain the current market image.

8.3 Client Application Restarts

The process for managing client application restarts is similar to that for “Late Connection / Intraday Refresh” as described above.

8.4 Sequence Reset Message

8.4.1 Sequence Reset Message from real time channels

8.4.1.1 Sequencing of events

1. Receipt of a “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 channels of all subscribed real time multicast channels to receive the current state of the market (refer to section 6.3.2.2).
5. Resume to process real time messages

8.4.1.2 Packet loss detection when processing Sequence Reset Message

After a Sequence Reset, the first UDP packet should have a sequence number of 1. If however this packet is lost and clients start receiving packets with a sequence number of 2 and onwards clients can try to recover the lost packet 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 have finished processing the retransmitted messages from RTS, clients can maintain the latest market image by handling the buffered data and then the live feed.

8.5 LMEsource Restarts before Market Open

In this scenario LMEsource performs a start-of-day twice due to errors encountered during first start-of-day. The second start-of-day should trigger a 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.

8.6 LMEsource Restarts after Market Open (Intraday Restart)

When LMEsource fails during trading hours, besides failing over to the backup site to resume service, LMEsource may be recovered by Intraday Restart where LMEsource will be shut down and then restarted. When LMEsource is shut down, retransmission services in both primary and backup sites will be disconnected and no multicast traffic, including heartbeats, will appear on any real-time or refresh channels.

LME will inform clients if an Intraday Restart needs to take place. The intraday restart will be announced by a news message on LMEselect, email or on the LME.com website.

LME will notify clients when LMEsource is subsequently restarted. Upon receipt of the notification from LME all clients should clear all their internal caches and reconnect to LMEsource afresh in the same way as if their systems start up late and connect to LMEsource 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 section 6.3.2.2) to establish the latest market snapshot.

The duration between LMEsource's shutdown and the subsequent restart could be a timespan of an hour and clients need to observe announcements made by LME.

8.7 LMEsource Component Failover

Electronic Channels

Component failover on Electronic channels would only impact one of LMEsource's lines. If no live data can be received on one line (line A) there is no impact to clients as LMEsource would continue publishing data via the alternative line (line B). Clients can reapply line arbitration as usual when LMEsource resumes the service and publishes the data from line A. Data published during the interruption would not be resent after line A has resumed the service.

Non-Electronic Channels

Component failover could cause non-electronic channels to experience an outage for a few seconds. When the service resumes recovery messages will be published to update clients with the latest market image. Duplicated messages resent will carry the same data content as the original but with new packet time and sequence number.

Refresh Channels

If a "Sequence Reset Message" is received from refresh channels, the client should clear the cached refresh messages and reset the sequence number of corresponding channel. (Refer to section 6.3.2.2 for handling sequence reset message from Refresh channels).

~~8.7~~8.8 Site Failover

In the event of a severe technical problem that requires LMEsource to failover to the backup site to allow for resumption of the market data service, a Disaster Recovery Signal (105) (referred to as "DR Signal" hereafter) message will indicate the status of LMEsource on the backup site for client actions.

A dedicated multicast channel is assigned to transmit DR Signal. During normal days, LMEsource starts up on the primary site and 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 RTS servers for Retransmission Service.

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

Clients are advised to clear all LMEsource data previously cached when DR Status "1" is detected in the DR Signal and prepare to execute their failover procedure. 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 6.3.2 on how to rebuild the market image from the refresh service). Clients are advised to build an automatic recovery mechanism so as to enable timely recovery on their side.

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

- Multicast data including heartbeats from all real-time & refresh channels is unavailable.
- Clients are unable to connect and log on to the RTS or, in the case of an already established RTS session, the TCP connection is disconnected or appears stale with no response from the RTS servers.

- LMEsource 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.
- LMEsource starts broadcasting the DR Status “2” (DR completed) status repeatedly once LMEsource 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 LMEsource 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 start-up when LMEsource 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 LMEsource site failover solution.

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.

~~8.8~~8.9 Special Trading Condition

The clients’ systems should have the flexibility to handle the variations of LMEsource data transmission patterns due to the extraordinary trading conditions which may necessitate a late start to LMEselect trading.

9 APPENDIX A – Pseudo Code of Line Arbitration

This example shows how clients can process a packet received from LMEsource. 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 a short period
            //Gap may be filled from the next few incoming packets,
            //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++;
                        }
        }
}
```

This example shows how 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);
            }
        }
}
```

10 APPENDIX B – Pseudo code for processing retransmission data

This example shows how clients can process incoming data from the LMEsource Retransmission server. It handles Heartbeat, RTS Logon Response, RetransRequest Response and Retrans data.

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

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

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

           switch (msgType)
           {
              case LOGON_RESPONSE_TYPE:
              {
                    LogonResponse *logonResponse
                                = (LogonResponse *)(mRtsBufPos + sizeof(PacketHeader));
                    processLogonResponse(logonResponse);
                    break;
              }
              case RETRANS_RESPONSE_TYPE:
              {
                    RetransResponse* resp = (RetransResponse*) (mRtsBufPos + sizeof(PacketHeader));
                    processRetransResponse(resp);
                    break;
              }
```

```
                default:
                        processPacket(packet);
            }
        }
    }
    // Wait for the rest of the data to come from the socket
    else
    {
        break;
    }
  }
}
```

11 APPENDIX C – Pseudo code for processing a refresh snapshot packet

This example shows how clients can process refresh snapshot data from the LMEsource RFS server and merge with real-time 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);
        }
}
```