Cboe Application Programming Interface

Version 1.0.4

CBOE Streaming Market Level 2 (Book Depth)

CBOE API for CSM Level 2 Book Depth Market Data

Cboe PROPRIETARY INFORMATION

July 25, 2014 (updated: 5/8/2018)

Disclaimer

The information contained in this document constitutes confidential and/or trade secret information belonging to Cboe. This document is made available to Cboe members and member firms for use with the CBOE Streaming Market Level 2 (Book Depth) Application Programming Interface. This document is provided "As Is" with all faults and without warranty of any kind, either express or implied

Change Notices

The following change notices are provided to assist users of the CSM Level 2 Book Depth feed in determining the impact of changes to their applications.

Date	Version	Description of Change
5/8/18	1.0.4	Removed references to C2. C2 transitioned to new technology.
2/27/18	1.0.4	Removed references to CFE. CFE transitioned to new technology.
10/12/17	1.0.4	Documented security type, Commodity (CMDTY)
9/6/16	1.0.3	Corrected CFE multi-cast IP addresses
3/23/15	1.0.2	Removed references to CBSX and One Chicago
12/16/14	1.0.1	Updated the open and close session times for CFE Futures
7/25/2014	1.0.1	Added Cboe_EXT multicast groups
11/6/2013	1.0.1	Sequence number reset at start of session (see SeqNoUpdate1, SeqNoUpdate2).
1/31/13	1.0	Production Release
1/03/13	0.8	Corrected Template Ids for MDSnapshotFullRefresh and MDIncRefresh which were reversed.
12/31/12	0.7.2	Corrected incorrect port numbers for CBSX feed.
12/26/12	0.7.1	Corrected Source Networks for Cboe, CFLEX, CFE, and ONE Chicago feeds
12/13/12	0.7	Added SecurityStatus message to prevent having to send SnapshotFullRefresh at market open just to update product state for products whose book structure does not change. Clarified usage of SecurityTradingStatus in IncRefresh and SnapshotFullRefresh messages. Documented Multicast groups and ports
11/13/12	0.6	Draft for release
11/08/12	0.5.2	Correct miscellaneous typos.
11/05/12	0.5.1	Correct MDEntrySize to be uInt32 vs int32
10/22/12	0.5	Initial Draft publication

Support and Questions Regarding This Document

Questions regarding this document can be directed to the Cboe Exchange at 312.786.7300 or via e-mail: api@cboe.com. The latest version of this document can be found at the Cboe web site http://systems.cboe.com.

Table of Contents

1 Introduction 7
1.1 System Overview 7
1.2 CSM Level 2 Feed Offerings 7
1.3 Hours of Operation 8
2 Data Feed and Message Overview 9
2.1 Data Feed Overview 9
2.2 Message Overview 9
2.3 Message Routing 11
3 Message Templates, Field Data Types and Data Encoding 12
3.1 Message Templates 12
3.2 Template IDs 12
3.3 Field Data Types and Data Encoding 13
4 Multicast Data Format 16
4.1 Packet Header 16
4.2 Message Header 17
5 Messages 19
5.1 Security Definition Message - Template ID 13 19
5.2 Snapshot Full Refresh Message - Template ID 17 23
5.3 Incremental Refresh Message - Template ID 18 26
5.4 Security Status Message – Template ID 19 29
5.5 Heartbeat Message (Line Integrity Message) - Template ID 16 30
6 Examples 31
6.1 Example Snapshot Full Refresh With Two-Sided Book 31
6.2 IncRefresh Insert Level – Customer Order 33
6.3 IncRefresh Delete + Change– Trade Top Of Book and Part of Level 2 34
6.4 IncRefresh Change + Overlay Level – Replace Top Of Book 35
6.5 IncRefresh Change – Trade Customer and Part Professional Volume 36
6.6 IncRefresh Insert – Contingent Volume at Top of Book 37
7 Processing For Start-Up and Data Recovery 38
7.1 Channel Level Processing 38
7.2 Product Level IncRefresh and Security Status Recovery Processing 38
7.3 Product Level Snapshot Recovery Processing 40
8 Appendix A – Multicast Group and Port Information 42
9 Appendix B – Hex Dump Wire Format Examples 48
9.1 Understanding Hex Dump Diagrams 48
9.2 Packet Header Example 48
9.3 Security Definition Message 49
9.4 Heartbeat Message 49

Reference Tables

1 - Example of an XML based template 12
2 - Templates and their IDs 12
3 - Packet Format 16
4 - Packet Header 16
5 - Message Format 17
6 - Message Header 17
7 – Multicast Message Types 17
8 - Security Definition Message Structure 20
9 - Security Definition Template 21
10 - Security Types 22
11 - Security Exchanges 22
12 - Security Price Types 22
13 - Security Put / Call 22
14 - Leg Side 22
15 - Exercise Styles 22
16 – Snapshot Full Refresh Message Structure 24
17 – Snapshot Full Refresh Template 25
18 - Security Trading Status 25
19 – Refresh Indicator 26
20 – MD Entry Type 26
21 – MD Volume Types 26
22 – Incremental Refresh Message Structure 28
23 – Incremental Refresh Template 29
24 – MD UpdateActions 29
25 – Security Status Message Structure 30
26 – Security Status Template 30
27 - Heartbeat Template 31

Examples of Data Transmissions

1 - Packet Header Hex Dump 48
2 - Packet Header Decoded 48
3 - Security Definition Hex Dump 49
4 - Security Definition Decoded 49
5 - Heartbeat Hex Dump 49
6 - Heartbeat Decoded 49

1 Introduction

CSM Level 2 publishes depth of book information for various Cboe markets / exchanges over the Cboe Financial Network (CFN) using the message format defined in this document. Data is transmitted using the IP Multicast network protocol. To connect to the CFN network, refer to the CFN Network Specification document on the Cboe API website at https://systems.cboe.com/Auth/CFN.aspx.

1.1 System Overview

CSM Level 2 distributes security definition data and depth of market data for the top N price levels of a market with individual quantity information for each price level (N is defined below for each feed). A **feed** is a set of multicast groups or **channels.** Several feeds in the CSM Level 2 format are available, separated by Cboe exchange and by product category or complexity. Each feed's channels are duplicated in a primary/secondary architecture with book depth data spread over one or more pairs of data channels. Communication is one way only with no mechanism for retransmission, but there are mechanisms for recovery. Messages are encoded using a mix of ASCII characters and binary data in the format defined in this document. Cboe also distributes templates that describe the message structure of CSM Level 2 feeds via the API website at https://systems.cboe.com/Auth/CFN.aspx. These templates may be used for encoding and decoding messages. The templates and message structures defined in this document will be static, and are not expected to change over the course of the trading day, nor even over a software release. Clients can expect sufficient advance notice about any changes to these templates or message structures.

1.2 CSM Level 2 Feed Offerings

A CSM Level 2 feed is a set of related multicast groups over which book depth and security definition information is transmitted for a particular Cboe exchange and product category or complexity. For Cboe exchanges that have strategy (multi-leg, complex) products, a separate feed is offered for strategy products versus non-strategy products for that exchange. This is done so recipients may choose to receive data only for products from the exchange they are interested in and for strategy / non-strategy products within that exchange.

For each feed, the number of book levels published *may* differ, but is generally the same across most feeds. Throughout the remainder of this document, a reference to "N levels" means you must consult the table below to find the value of N for a particular feed.

The feeds offered and number of book levels published is:

Exchange Feed And Product Category	**Num Levels**
Cboe Non-Strategy Options	5
Cboe Strategy Options (complex multi-leg products)	5
Cboe Flex (CFLEX) Non-Strategy Options	5
Cboe Flex (CFLEX) Strategy Options	5

1.2.1 For Customers of CSM Current Market: Security Definition is Identical

Both the *CSM Current Market* feed (a top of book feed offering separate from the Book Depth feed) and the *CSM Level 2* feed each publish data to their own dedicated security definition channels. This is done so the security definition multicast group for a feed is logically grouped with its data channels to make network multicast masking and routing less difficult to manage. In addition, the two kinds of feeds are purchased separately and operate from different systems.

If you are an existing customer of the CSM Current Market feed, and you also subscribe to this Book Depth feed, you are not required to read the security definition channel from both feeds. The format of the security definition message is identical on both feeds, and will publish the same data. The timing and sequencing of messages and the sequence numbers on the security definition channels will differ because the systems operate independently, but the content will be the same set of products on both.

Customers are free to leverage re-use of the same code to parse and process the security definition from either feed and they may choose to read only one of the security definition feeds. However, there is no penalty for reading both copies of the feed, should you choose to do so.

In the event of a future change to the format of the security definition message, the format will change on both CSM Current Market and CSM Level 2 Book Depth at the same time and will be rolled out to customers concurrently so the format is consistent on both types of feeds.

1.3 Hours of Operation

The system is expected to be available from 6:00 AM Central time (CT) to 4:00 PM CT. Securities are expected to transition to Pre-open around 6:00 AM. Security Definition Messages, Snapshot messages and IncRefresh messages will be published during these hours.

Normal market hours for exchanges and products within those exchanges are as follows:

Exchange	**Product**	**Open (CT)**	**Close (CT)**
Cboe	Interest Rate Options	7:20 AM	2:00 PM
Cboe	Equity Options	8:30 AM	3:00 PM
Cboe	Index Options	8:30 AM	3:15 PM

2 Data Feed and Message Overview

2.1 Data Feed Overview

A feed consists of one or more **data** multicast groups or channels and a **security definition** multicast group / channel. The number of data channels differs for various feeds based on capacity requirements. Low volume feeds may have only one data channel, while high volume feeds have multiple data channels. Each feed has a single security definition channel.

General characteristics of a feed include the following:

- Each feed contains book depth data only for products from a particular exchange and product category or complexity as defined in *CSM Level 2 Feed Offerings*.
- Each channel of a feed is duplicated and sent to 2 different multicast groups and ports over 2 networks in a primary/secondary configuration.
- There are no retransmissions. If a recipient is late to join, or if packets are dropped, a complete cycle of Snapshot messages must be processed to insure accuracy of all book data. See section *7 Processing For Start-Up and Data Recovery* for more details on how to implement recovery processing.
- A sequence number is sent for each message. This can be used to identify missed messages over a particular channel. Each channel has its own sequence number, so for example, channel 1 of a feed has sequence numbers 1, 2, 3, etc. and channel 2 has its own sequence numbers of 1, 2, 3, etc. Sequence numbers will be reset at the start of each trading day's session.
- Messages are placed into blocks (packets) before delivery which allows for multiple messages per block. The maximum block size is 1000 bytes.
- The message structures, field names and field values are based as much as possible on the FIX 5.0 SP2 standard. However, messages are encoded using a proprietary hybrid ASCII + binary format, and FIX tags are not transmitted in the data stream. The FIX format was used for the convenience of those familiar with the FIX standard, so messages are defined in terms of FIX field names and FIX tags. Some user-defined fields were necessary for those fields not in the FIX specification, and some modifications to standard FIX fields are implemented for efficiency reasons.

2.2 Message Overview

Several types of messages are transmitted over a feed.

2.2.1 Security Definitions

Security definitions describe an exchange's products by name and trading parameters and associates those products with a product security ID. To conserve bandwidth, book depth update messages are transmitted with only security ID information, so the security definition message is used to relate the security ID from those messages to detailed product information.

Security definition messages are sent for all products on a dedicated channel in a cycle that repeats approximately every two minutes. They are used to establish the initial set of products traded on the feed. For new products added intraday, a security definition is sent to the data channel for that product preceding the first update for that product. Thereafter, newly added products are added to the next cycle of the security definition channel.

2.2.2 Book Depth Snapshot Full Refresh (Snapshot)

Snapshot messages contain the current SecurityTradingStatus for a product and the complete state of the top N levels of the book where N is defined in *CSM Level 2 Feed Offerings*. They are sent to the data channels for a feed.

Snapshot messages are sent for all products to the data channel associated with a product in a cycle that repeats approximately every two minutes (similar to security definitions). The Snapshot message is used to establish the initial state of the book at start-up, and to recover the state of the book when messages are dropped. The Snapshot is also used to establish the initial state of new products added intraday, and to re-establish the state of the book when data has been interrupted on a feed, as with a system failure.

2.2.3 Book Depth Incremental Refresh (IncRefresh)

IncRefresh messages contain the current SecurityTradingStatus for a product and changes to bid and offer prices and volumes for the top N levels of the book where N is defined in *CSM Level 2 Feed Offerings*. They are sent to the data channels for a feed.

IncRefresh messages are incremental instructions that indicate how the book structure has changed, the level that changed, which side (Bid or Ask) changed, corresponding volumes, and the current SecurityTradingStatus. IncRefresh messages may insert a new level, delete an existing level, update the volumes for a level, or overlay the price and volumes for a level. IncRefresh messages may also change the SecurityTradingStatus while simultaneously modifying levels of the book.

IncRefresh messages can be processed only if the current state of the book is accurate. In the event of dropped messages or at start-up before the state of the book is established, a recovery procedure must be implemented before they can be processed which requires waiting for a Snapshot message.

2.2.4 Security Status

Security Status messages contain the SecurityTradingStatus for a product. They are sent when the trading state of a product changes, but there is no structural book change.

In situations when a product state changes and there is a structural book change, an Incremental Refresh message will be sent instead along with a changed SecurityTradingStatus. The Security Status message is sent only when the trading state for a product changes but there is no structural book change.

2.2.5 Heartbeats

Heartbeat / line integrity messages are transmitted every five seconds to every channel. These messages may be used to determine that a channel is working during times when data is not transmitted on the feed (such as pre-market or post-market times).

2.3 Message Routing

Messages are routed to specific channels of a feed based on the type of message and and it's content.

2.3.1 Security Definitions

- Security definitions are sent to a dedicated security definition channel for each feed, with one exception, (see below).
- Security definition messages for all products are sent to the security definition channel in a repeating cycle of approximately 2 minutes duration. When one cycle completes the next begins immediately. A security definition for a particular product should therefore repeat approximately every 2 minutes.
- Security definitions for *existing* products are sent only to the security definition channel for the feed.
- When a *new* product is added intraday, a security definition will be sent once to the *data* channel for the newly added security, just before the first update for that security is sent to the data channel.

  This is done allow recipients to stop reading the security definition channel after they initially synchronize with a complete cycle and to insure that the security definition for a new product is seen before the first update for the product.
- After a new product is added intraday and its initial security definition is sent once to the data channel for the product, the product is added to the security definition repeating cycle. Subsequent security definitions for the product will appear only on the security definition channel.
- A field in the security definition, *TargetLocationID* indicates over which data channel the book updates for product will be transmitted. This is discussed in more detail in the definition section of this document.

2.3.2 IncRefresh, Snapshot, and Security Status Messages

- Messages are routed to a single data channel within a feed by **product class key**. A product class **is a numeric identifier that is related to, but not the same as the underlying security** for a product. One channel is chosen for a particular product class key which will not change intra-day.
  For example, all IBM products (options for options feeds, futures for futures feeds, etc.) will be sent to one channel. Corporate actions for IBM such as IBM1 share the same product class, so its products will be sent to the same channel.
  By contrast, SPX (regular) and SPXW (weekly) options share a common underlying (the S&P 500), but are different product classes and may be on different data channels.
- Across days (I.E. overnight), the data channel chosen for a product class may change due to load-balancing considerations or system improvements or enhancements to the distribution method.
- IncRefresh, Snapshot and Security Status messages for a particular product are sent to the same data channel to preserve correct sequencing.

2.3.3 Heartbeats

Heartbeat / line integrity messages are transmitted every five seconds to every channel.

3 Message Templates, Field Data Types and Data Encoding

3.1 Message Templates

Messages for CSM Level 2 feeds are described in this document in tabular text format and as an XML template. Templates define the content and characteristics of the messages to be encoded or decoded.

XML templates that describe the structure of messages are available to recipients on the Cboe web site at https://systems.cboe.com/Auth/CFN.aspx. Firms are encouraged to write software capable of using the XML templates to decode data from a CSM Level 2 feed.

XML templates are used to specify the structure, data types, field names, and FIX tags of a message:

1 - Example of an XML based template

`<template name="MDIncRefresh" id = "0">`	Start of new template
`<string name="MessageType" id="35" byteLength="1" value="X" />`	Defines a String Data Type Field, The id which represents the fix Tag is not transferred on the wire.
`<uInt32 name="MsgSeqNum" id="34" />`	
`<sequence name="MDEntries">`	Defines the start of a repeating group
`<length name="NoMDEntries" id ="268"/>`	Length of repeating group
`<string name="Symbol" id="55"/>`	
`<iUnt32 name="Quantity" id="53 />`	
`</sequence>`	End repeating group
`</template>`	End template

3.2 Template IDs

Each message structure is defined with a unique template ID. A template ID is a binary integer value stored as the first byte of every message that identifies the structure of the message.

The template ids for CSM Level 2 feeds are as follows:

2 - Templates and their IDs

Template Name	Template ID
Security Definition	13
Incremental Refresh	18
Snapshot Full Refresh	17
Security Status	19
Heartbeat	16

3.3 Field Data Types and Data Encoding

Fields defined in messages for CSM Level 2 feeds will have one of the following data types and methods of encoding:

3.3.1 STRING Field

Strings are ASCII character arrays or single-byte characters. There are two types of string fields which are encoded differently:

Single Byte String

If the *byteLength* attribute of a string field is defined as "1", for example:
<string name="MessageType" id="35" byteLength="1"/>
it is a single byte string, which is encoded with a single ASCII character.

Character Array String

If the *byteLength* attribute is not defined for the field, for example:
<string name="Symbol" id="55"/>
The string is variable length and is encoded with an unsigned binary byte indicating the length of string, followed by the string's characters:

Length of String (1 byte)	String Characters

For example, the string "IBM" would be encoded as:
Binary value 3, then the characters "IBM".

3.3.2 INTEGER Fields

Integer fields are big endian binary encodings of numeric values. The *byteLength* attribute in a template field definition can act as a modifier to restrict the number of bytes used to encode the integer value. Unsigned integers are encoded as zero or positive-only values. The top-most bit is part of the magnitude of the value. Signed integers are encoded as twos-complement binary values with the top-most bit as the sign bit.

There are several types of integer values:

Integer Field Type	**Byte Length**	**Encoding**	**Example**
uInt32	1	8 bit unsigned integer	<uInt32 name="MDPriceLevel" id="1023" byteLength="1"/>
uInt32	4 or omitted	32 bit unsigned integer	<uInt32 name="MDEntrySize" id="271" byteLength="4"/>
uInt64	8 or omitted	64 bit unsigned integer	<uInt64 name="SendingTime" id="52"/>
int32	1	8 bit signed integer	<int32 name="MDPriceLevel" id="1023" byteLength="1"/>
int32	4 or omitted	32 bit signed integer	<int32 name="MDEntrySize" id="271"/>
int64	8 or omitted	64 bit signed integer	<int64 name="SendingTime" id="52" byteLength="8">

The table below shows the min and max values for different integer data types.

Type	Min	Max
uInt32 with byteLength=”1”	0	255
uInt32	0	4,294,967,295
uInt64	0	18,446,744,073,709,551,615
int32 with byteLength=”1”	-128	127
int32	-2,147,483,648	2,147,483,647
int64	- 9,223,372,036,854,775,808	9,223,372,036,854,775,807

3.3.3 DECIMAL Field

A decimal field is used to represent a floating point number as exponent and mantissa. The exponent is a signed 8 bit integer used to express precision and the mantissa is a signed 32 bit integer used to express the value. The numerical value is obtained by multiplying the mantissa with the base-10 power of the exponent expressed as: number = mantissa * 10^{exp}. The exponent and mantissa is decoded as a single, composite field.

Decimal fields are 5 bytes in length. The first byte is the exponent and the remaining 4 bytes represent the mantissa. For example, the number 0.90 is encoded as FE0000005A.
FE (exponent) == -2, 0000005A (mantissa) == 90, value == 90 * 10^{-2} == 90 * 0.01 == 0.90

3.3.4 SEQUENCE Field

A sequence is a repeating group of fields. The group of fields contained in a sequence can be a simple type as described above, or another nested sequence. A length field encoded as an unsigned int immediately precedes the fields contained in the sequence. The length field is defined in a template with a special attribute of “<length”, and it can be modified with *byteLength* attribute. If byteLength=”1”, the encoding length field is a single 8-bit unsigned byte. All sequences transmitted to CSM Level 2 feeds use a single 8-bit unsigned length and can be no longer than 255 entries.

Sequences are encoded as follows:

Length field	Group#1 Field #1	Group#1 Field#2	…	Group#1 Field#N	Group#2 Field#1	Group#2 Field#2	…	Group#2 Field#N	…

Note that a sequence may contain a nested sequence, as in the following template example:

```
<sequence   name="MDEntries">
    <length     name="NoMDEntries"          id="268"    byteLength="1"/>
    <string     name="MDEntryType"          id="269"    byteLength="1"/>
    <decimal    name="MDEntryPx"            id="270"    byteLength="5"/>
    <sequence   name="MDVolumeEntries">
        <length     name="NoMDVolumeEntries"    id="21000"  byteLength="1"/>
        <uInt32     name="MDVolumeType"         id="21001"  byteLength="1"/>
        <uInt32     name="MDEntrySize"          id="271"    byteLength="4"/>
    </sequence>
</sequence>
```

Here is an example of the above sequence with 2 MDEntries elements each with different lengths of nested MDVolumeEntries.

Field	Length in Bytes	Value	Comments
NoMDEntries	1	2	Length of MDEntries Sequence
MDEntryType	1	'0'	Bid entry
MDEntryPx	5	FE0000000A	FE == -2 exponent, 0000000A == 10 mantissa, value == 0.10
NoMDVolume Entries	1	2	Bid entry has 2 volume entries
MDVolumeType	1	1	Customer limit volume type
MDEntrySize	4	00000004	Customer limit volume == 4
MDVolumeType	1	0	Total Limit volume type
MDEntrySize	4	00000014	Total Limit volume == 20
MDEntryType	1	'1'	Ask entry
MDEntryPx	4	FE00000010	FE == -2 exponent, 00000010 == 16 mantissa, value == 0.16
NoMDVolume Entries	1	1	Ask entry has 1 volume entry.
MDVolumeType	1	0	Total limit volume type (no customer volume for Ask)
MDEntrySize	4	0000000A	Total Limit Volume == 10

4 Multicast Data Format

All messages are sent in Multicast packets. Each packet consists of a packet header and one or more messages.

3 - Packet Format

Packet (a.k.a. Block)					
Packet Header					Contents
Version	**Length**	**Sending Time**	**Number of messages**	**First Msg Seq #**	**Messages...**

4.1 Packet Header

Each packet has a packet header that appears once at the beginning of the packet. The packet header has the following structure:

4 - Packet Header

Field Name	Type	Length (Bytes)	Comments
Version	uInt32	1	The version associated with the contents and format of this header. Currently, this will be a constant value of 1.
Packet Length	uInt32	2	Length of the packet including this length field and the version. Note that this is a 2 byte length.
Sending Time	uInt64	8	The time that this packet was sent. It applies to all messages in this packet.
Number of Messages	uInt32	1	The number of messages in this packet.
First Msg Seq #	uInt32	4	The sequence number on the first message in this packet.

The Version of a packet indicates the format of the packet. This may be incremented in future releases to indicate a change in the format of the packet. Initially, it is set to the number 1.

The Packet Length is encoded as a 2 byte (16 bit) unsigned integer that includes the length of the version, the 2 byte Packet Length itself, and the remainder of the packet.

The Sending Time is the time that the CSM Level 2 application published the packet on the feed. The sending time is the millisecond timestamp from midnight, January 1, 1970 UTC.

The "First Msg Seq #" is the sequence number of the first message of this packet, and the "Number of Messages" indicates the total number of messages contained in the packet.

For verification of data at the channel level, one could compute the expected "first msg seq #" of the next packet by adding the number of messages to the current packet's "first msg seq #".

4.2 Message Header

A packet contains one or more messages. Each message is preceded by a message header common to all messages.

5 - Message Format

Message						
Message Header				Contents		
		Template defined fields				
Message Length	**Template ID**	**Msg Type**	**Msg Seq #**	**Field #1**	**…**	**Field #N**

6 - Message Header

Field ID	Field Name	Type	Length (Bytes)	Comments
	Message Length	uInt32	2	The length of this message including the 2 bytes for this length field.
	Template ID	uInt32	1	The Template ID is for decoding the message. See table: *2 - Templates and their IDs*
35	MessageType	String	1	See table: *7 – Multicast Message Types*
34	MsgSeqNum	uInt32	4	Sequence Number

The Message Length is encoded as a 2 byte (16 bit) unsigned integer that includes itself and the remainder of the message including all Message Header fields.

The Template ID defines the specific Structure of the message.

The Msg Type defines the market data message type compliant to the FIX standard.

The message sequence number is a consecutively increasing number from the previous message. The first message in a packet will start with 1 number greater than the last message in the previous packet.

7 – Multicast Message Types

d	Security Definition
X	Incremental Refresh
W	Snapshot Full Refresh
f	Security Status
0	Heartbeat

4.2.1 Message Sequence Numbers

Every packet has a "first" sequence number in the header. This is the number associated with the first message in the packet. Each subsequent message in the packet has a sequence number that is one greater than the previous message. The next packet will have a starting sequence number that is one more than the last message in the previous packet, except in the event of a Cboe system failure. When

this occurs, the sequence number can reset to a lower value than was previously seen prior to the failure. Thereafter, the sequence number will again increase by one for each message.

Each multicast channel on a feed has its own sequence number associated with it starting with sequence number 1. This sequence number will be reset to 1 at the start of each trading day's session. Verification of message sequence numbering must be done for each individual multicast channel.

Firms must ensure that the sequence numbers maintain continuity. Any deviation from an expected sequence number must be considered as an error condition. Firms are required to take appropriate recovery action any time that an unexpected sequence number is detected.

See the section *Processing For Start-Up and Data Recovery* for details on how the MsgSeqNumber can be used to handle various recovery scenarios.

5 Messages

5.1 Security Definition Message - Template ID 13

Security definitions describe an exchange's products by name and trading parameters and associates those products with a product security ID.

Security definition messages for all products are sent to the security definition channel in a repeating cycle of approximately 2 minutes duration. When one cycle completes the next begins immediately

When a *new* product is added intraday, a security definition will be sent once to the *data* channel for the newly added security, just before the first update for that security is sent to the data channel.

At startup, a complete cycle of the security definition channel must be read to build the initial set of products. To detect a cycle, watch for the same securityID to appear twice.

After a complete security definition cycle is read, it is not necessary to continue reading the channel unless a message on one of the data channels is dropped. The data that was dropped could have been a new security, so another cycle of the security definition channel should be read to re-establish the complete set of products. As long as there is no missing data on any data channel, however, it is not necessary to read the security definition channel after the first complete cycle.

For feeds that contain strategies (multi-leg products), the leg security definitions will be included on the security definition channel of the strategy feed. Book updates for the legs will not be sent on the strategy feed's data channels, however.

The Security Definition message fields are as follows:

8 - Security Definition Message Structure

Field ID	Field Name	Type	Length (Bytes)	Comments
	Message Header			See table: *6 - Message Header*, MessageType = "d"
167	SecurityType	string		See table: *10 - Security Types*
207	SecurityExchange	single byte string	1	See table: *11 - Security Exchanges*
55	Symbol	string		Symbol of the class
143	TargetLocationID	string		See *Target Location ID*
21004	ClassKey	uInt32	4	Product Class key
48	SecurityID	uInt32	4	Product key
541	MaturityDate	uInt64	8	Expiration date. Format is "YYYYMMDD" This field is required for options and futures
423	PriceType	uInt32	1	Specifies how to interpret the value in the StrikePrice field. See table: 12 - Security Price Types.
202	StrikePrice	decimal	5	First byte represents the exponent and the last 4 bytes represent the mantissa This field is required for options
201	PutOrCall	uInt32	1	This field is required for options. See table: 13 - Security Put / Call
21005	MinimumStrike PriceFraction	decimal	5	The minimum interval between strike prices.
21006	MaxStrikePrice	decimal	5	The maximum allowable strike price.
21007	PremiumBreak Point	decimal	5	The premium price where above and below fractions take effect.
21008	MinimumAbove PremiumFraction	decimal	5	The multiple that premium can be when above the break point.
21009	MinimumBelow PremiumFraction	decimal	5	The multiple that premium can be when below the break point.
21010	ExerciseStyle	uInt32	1	See table: 15 - Exercise Styles
996	CurrencyCode	string		Not used
311	UnderlyingSymbol	string		The underlying symbol
310	UnderlyingType	string		See table: 10 - Security Types
231	ContractSize	uInt32	4	The number of contracts per unit of size
	Legs	Sequence Field of legs		A sequence Field contains one or more legs information, See below.

The *Legs* sequence field has following sub fields (For detailed format information, refer to the section *Field Data Types and Data Encoding* in this document):

Field ID	Field Name	Type	Length (Bytes)	Comments
555	NoLegs	length (unsigned int)	1	Num. of Legs in a strategy product. Only used for products that contain legs such as strategies. The length of the sequence will not be more than 256
The Following Fields Repeat NoLegs times for MLEG – Strategies				
623	LegRatioQty	uInt32	4	Leg ratio
602	LegSecurityID	uInt32	4	SecurityID of the leg
624	LegSide	string	1	See table: 14 - Leg Side

9 - Security Definition Template

```
<template   name="SecurityDefinition"           id="13">
    <string     name="MessageType"                  id="35"     byteLength="1"
                                                                value="d"/>
    <uInt32     name="MsgSeqNum"                    id="34"     byteLength="4"/>
    <string     name="SecurityType"                 id="167"/>
    <string     name="SecurityExchange"             id="207"    byteLength="1"/>
    <string     name="Symbol"                       id="55"/>
    <string     name="TargetLocationID"             id="143"/>
    <uInt32     name="ClassKey"                     id="21004"  byteLength="4"/>
    <uInt32     name="SecurityID"                   id="48"/>
    <uInt64     name="MaturityDate"                 id="541"    byteLength="8"/>
    <uInt32     name="PriceType"                    id="423"    byteLength="1"/>
    <decimal    name="StrikePrice"                  id="202"    byteLength="5"/>
    <uInt32     name="PutOrCall"                    id="201"    byteLength="1"/>
    <decimal    name="MinimumStrikePriceFraction"   id="21005"  byteLength="5"/>
    <decimal    name="MaxStrikePrice"               id="21006"  byteLength="5"/>
    <decimal    name="PremiumBreakPoint"            id="21007"  byteLength="5"/>
    <decimal    name="MinimumAbovePremiumFraction"  id="21008"  byteLength="5"/>
    <decimal    name="MinimumBelowPremiumFraction"  id="21009"  byteLength="5"/>
    <uInt32     name="ExerciseStyle"                id="21010"  byteLength="1"/>
    <string     name="CurrencyCode"                 id="996"/>
    <string     name="UnderlyingSymbol"             id="311"/>
    <string     name="UnderlyingType"               id="310"/>
    <uInt32     name="ContractSize"                 id="231"    byteLength="4"/>
    <sequence   name="Legs">
        <length     name="NoLegs"                       id="555"    byteLength="1"/>
        <uInt32     name="LegRatioQty"                  id="623"    byteLength="4"/>
        <uInt32     name="LegSecurityID"                id="602"    byteLength="4"/>
        <string     name="LegSide"                      id="624"    byteLength="1"/>
    </sequence>
</template>
```

10 - Security Types

OPT	Options
FUT	Futures
CS	Common Stock
INDX	Indexes
MLEG	Strategies
CMDTY	Commodity

11 - Security Exchanges

C	Cboe
O	One Chicago – Not Supported
W	CBSX – Not Supported
F	CFE/COF – Not Supported
2	Cboe2 Options – Not Supported

12 - Security Price Types

1	Percentage
3	Fixed Amount

13 - Security Put / Call

0	PUT
1	CALL

14 - Leg Side

B	Buy (Bid)
S	Sell (Ask)

15 - Exercise Styles

0	American
1	European

5.1.1 Target Location ID

Target Location ID is an ASCII-character encoded string containing a numerical index that indicates over which data channel the book updates for a given product will be delivered. It is a zero-based index of the data channel number for a feed. This can be used if desired to figure out over which data channel a product will be transmitted.

The relationship between the TargetLocationID and the actual multicast group can be found in Appendix A – Multicast Group and Port Information

The TargetLocationID is unique only within a feed and not across feeds. For example, Cboe Non Strategy Options, Cboe Strategy, each have channel index ‘0’. There is no cross-feed reference.

The TargetLocationID for a leg security definition message sent to a strategy feed is undefined, and will be set to '0'. It is not a reference to a channel index in the non-strategy feed.

5.2 Snapshot Full Refresh Message - Template ID 17

Snapshot messages contain the current SecurityTradingStatus for a product and the complete state of the top N levels of the book where N is defined in *CSM Level 2 Feed Offerings*. Each Snapshot message contains information for only one product.

Snapshot messages are used by recipients to establish the initial state of the book, and to recover the state of the book if messages are dropped. Under *normal* processing, once the book state has been established by a recipient application, Snapshot messages can be ignored except as noted below.

In situations such as a system failure or when a new product is added intraday, a Snapshot message will be sent with a *RefreshIndicator* value of 'Y This is used to indicate that recipients MUST process the Snapshot, even if they have completed a cycle and think they are initialized properly.

See the section *Processing For Start-Up and Data Recovery* for detailed information on how Snapshot recovery processing can be implemented.

Snapshot messages are sent continuously to the data channel associated with a product and are co-mingled with IncRefresh and SecurityStatus messages, possibly in the same packet. They are transmitted in a repeating cycle of approximately two minutes.

Snapshot messages indicate the position of each price in the book up to N levels, which side of the book the price refers to, and the corresponding volumes and volume types for that price.

The types of volumes that are sent if present in the top N levels of the book include:

- Total limit volume for each price level
- Customer Limit for each price level
- Total Contingent volume for each price level
- Customer Contingent volume for each price level.

To process a Snapshot, the book should be cleared and re-built entirely from only the contents of the Snapshot message.

Only the volume types that are non-zero for each price level are transmitted in the Snapshot. All other volume types not transmitted in the Snapshot are implied to be zero.

For products with an empty book, the *NoMdEntries* in the Snapshot message will be zero, and no *MDEntries* are present in the Snapshot message.

For products with only bid-side or only ask-side book entries, the *MDEntries* sequence will contain only entries with an *MDEntryType* for the side of the book that is present. No *MDEntry* will be present in the Snapshot message for the side of the book that is empty. In other words, the absence of an *MDEntry* for a side of the book implies that side of the book is empty.

For products where the book contains less than N price levels, only the number of levels that exist will be sent in the Snapshot. For example, if N is 5 levels and a product has 2 bid-side levels and 3 ask-side levels, only 5 *MDEntry* items will be sent. Two *MDEntry* items will be sent with *MDEntryType* of bid, and 3 *MDEntry* items will be sent with an *MDEntryType* of ask. There will be no bid-side entries sent for levels 3-5 and no ask-side entries sent for levels 4-5. The absence of an *MDEntry* for a side and level implies that level does not exist for that side of the book.

The Snapshot Full Refresh message fields are as follows:

16 – Snapshot Full Refresh Message Structure

Field ID	Field Name	Type	Length (Bytes)	Comments
	MessageHeader			See *6 - Message Header*, MessageType = "W"
21004	ClassKey	uInt32	4	Product Class key
48	SecurityID	uInt32	4	Product ID
83	RptSeq	uInt32	4	Per-product seq number. See the section *Processing For Start-Up and Data Recovery* for how this can be used.
326	SecurityTrading Status	uInt32	1	See table: *18 - Security Trading* Status
423	PriceType	uInt32	1	Specifies how to interpret the values in the MDEntryPx field. See table: *12 - Security Price Types*
1187	Refresh Indicator	single byte string	1	See table: *19 – Refresh Indicator*
	MDEntries	Sequence of MDEntries		A Sequence of MDEntry indicating price, side, level, and volumes of the book.

The *MDEntries* sequence field has the following sub fields (For detailed format information, refer to the section *Field Data Types and Data Encoding* in this document):

Field ID	Field Name	Type	Length (Bytes)	Comments
268	NoMDEntries	length (unsigned int)	1	Number of MDEntries in this message. The length of the sequence will not be more than 255
The Following Fields Repeat NoMDEntries times				
269	MDEntryType	single byte string	1	Side of book for this entry. See table: *20 – MD Entry Type*
1023	MDPriceLevel	uInt32	1	Level number of book for this *MDEntry*.
270	MDEntryPx	decimal	5	Price at this level and side of book.
	MDVolume Entries			Sequence of Volume Entries for this price level.

In each *MDEntry* contains a nested sequence of *MDVolumeEntries*. Each *MDVolumeEntry* has the following sub-fields:

Field ID	Field Name	Type	Length (Bytes)	Comments
21000	NoMDVolume Entries	length (unsigned int)	1	Number of *MDVolumeEntries* for this *MDEntry*. The length of this sequence cannot be more than 4 (the number of valid volume types)
The Following Fields Repeat NoMDVolumeEntries times				
21001	MDVolume Type	uInt32	1	See table: *21 – MD Volume* Types
271	MDEntrySize	uInt32	4	Quantity for this type of volume.

17 – Snapshot Full Refresh Template

```
<template   name="MDSnapshotFullRefresh"    id="17">
    <string     name="MessageType"              id="35"     byteLength="1"
                                                            value="W"/>
    <uInt32     name="MsgSeqNum"                id="34"     byteLength="4"/>
    <uInt32     name="ClassKey"                 id="21004"  byteLength="4"/>
    <uInt32     name="SecurityID"               id="48"     byteLength="4"/>
    <uInt32     name="RptSeq"                   id="83"     byteLength="4"/>
    <uInt32     name="SecurityTradingStatus"    id="326"    byteLength="1"/>
    <uInt32     name="PriceType"                id="423"    byteLength="1"/>
    <string     name="RefreshIndicator"         id="1187"   byteLength="1"/>
    <sequence   name="MDEntries">
        <length   name="NoMDEntries"               id="268"    byteLength="1"/>
        <string   name="MDEntryType"               id="269"    byteLength="1"/>
        <uInt32   name="MDPriceLevel"              id="1023"   byteLength="1"/>
        <decimal  name="MDEntryPx"                 id="270"    byteLength="5"/>
        <sequence   name="MDVolumeEntries">
            <length     name="NoMDVolumeEntries"        id="21000"  byteLength="1"/>
            <uInt32     name="MDVolumeType"             id="21001"  byteLength="1"/>
            <uInt32     name="MDEntrySize"              id="271"    byteLength="4"/>
        </sequence>
    </sequence>
</template>
```

18 - Security Trading Status

2	Market Halted
17	Market Open
18	Market Closed
21	Pre Open
22	Market in Opening Rotation
23	Fast Market
24	Strategy Market in Opening Rotation
25	Strategy Market Quotes Non-Firm
26	Market Suspended (Quotes are not firm)

19 – Refresh Indicator

Y	Mandatory refresh by all recipients. Snapshot must be processed.
N	Process if necessary

20 – MD Entry Type

0	Buy (Bid)
1	Sell (Ask)

21 – MD Volume Types

0	Total Limit
1	Customer Limit
2	Total Contingency (All or None)
3	Customer Contingency (All or None)

5.2.1 About Volumes and Volume Types

Volume types indicate what kind of volume is contained in the MDEntrySize of the MDVolumeEntry. Different volume types have different meanings which are described below.

Total Limit Volume
Total limit volume includes professional and customer volume. To find the professional limit volume, you can subtract customer limit volume from total limit volume.

Customer Limit Volume
Customer limit volume contains customer orders resting in the book.

Total Contingent Volume
Total contingent volume contains professional and customer contingent volume (such as all-or-none orders). To find the professional contingent volume, subtract customer contingent volume from total contingent volume.

Customer Contingent Volume
Customer contingent volume contains volume from customer contingency orders.

5.3 Incremental Refresh Message - Template ID 18

IncRefresh messages contain the current SecurityTradingStatus for a product and one or more instructions that indicate how the book structure has incrementally changed since the last book update. Each IncRefresh message contains information for only one product.

IncRefresh messages indicate the level of the book that changed, which side (Bid or Ask) changed, and corresponding volumes and volume types for a price level and side.

The types of volumes that are sent if present in the top N levels of the book include:

- Total limit volume for each price level
- Customer Limit for each price level
- Total Contingent volume for each price level

- Customer Contingent volume for each price level.

IncRefresh messages may insert a new level, delete an existing level, update the volumes for a level, or overlay the price and volumes for a level. Update and overlay are similar except that updates modify only volumes, while overlay modifies volumes and prices. IncRefresh messages may also modify the SecurityTradingStatus while simultaneously identifying the structural changes to the book.

IncRefresh messages can be processed only if the current state of the book is accurate. In the event of dropped messages or at start-up before the state of the book is established, a recovery procedure must be implemented that requires waiting for a Snapshot for the products whose data is missing. See the section *Processing For Start-Up and Data Recovery* for more information on how IncRefresh messages can be processed for recovery purposes.

The volume types and volumes transmitted on an IncRefresh message are complete replacements for all possible volumes for a particular price level of the book. Only non-zero volumes for each price level are transmitted, and these represent the current and complete state of the volumes for a price and side. The absence of a volume type implies that volume is zero. Volume updates are not incremental.

Insert instructions should push down levels of the book that are >= to the level to be inserted, then the new entry is inserted at the indicated level. If N levels of the book exist before the insert is applied, level N should be deleted or "rolled off" from the bottom of the book as part of processing the insert instruction. No separate instruction will be sent to remove the bottom of the book.

Delete instructions should remove a level of the book and pull up levels >= the level to be deleted. Note that doing this causes the depth of the book to reduce by one temporarily. If more levels are available to fill in the bottom of the book, delete instructions will be accompanied by insert instructions to fill in the bottom of the book.

Delete instructions will not contain volumes. The *NoMDVolumeEntries* will be zero for all delete instructions.

Change instructions will change only the volumes of a book entry, and not the prices. The volumes and volume types sent with the change instruction are complete replacements for all volumes associated with the price, level and side of the book. Only non-zero volumes for each volume types is sent; absent volume types are implied to be zero.
Overlay instructions are used to indicate a delete/insert combination where the price of a level of the book changes, as well as its volumes. For example, if the top of the book (level 1) changes price, but the new price is still the top of the book, this is like a delete/insert at the top of the book. It would be inefficient to send 2 instructions, however, so an overlay instruction will be sent instead. Overlay is used to differentiate a price + volume change for a level from a volume-only change for a level.

Note A single IncRefresh message can contain multiple instructions and can update both sides of the book. In a changing market it is possible for bid and ask prices to cross temporarily before all instructions are processed. The book should not be used until all IncRefresh entries in a message are processed.

22 – Incremental Refresh Message Structure

Field ID	Field Name	Type	Length (Bytes)	Comments
	MessageHeader			See *6 - Message Header*, MessageType = "X"
21004	ClassKey	uInt32	4	Product Class key
48	SecurityID	uInt32	4	Product ID
83	RptSeq	uInt32	4	Per-product seq number. See the section *Processing For Start-Up and Data Recovery* for how this can be used.
326	SecurityTrading Status	uInt32	1	See table: *18 - Security Trading* Status
423	PriceType	uInt32	1	Specifies how to interpret the values in the MDEntryPx field. See table: *12 - Security Price Types*
	MDEntries	Sequence of MDEntries		A Sequence of MDEntry indicating how price, side, level, and volumes of the book changed.

The *MDEntries* sequence field has the following sub fields (For detailed format information, refer to the section *Field Data Types and Data Encoding* in this document):

Field ID	Field Name	Type	Length (Bytes)	Comments
268	NoMDEntries	length (unsigned int)	1	Number of MDEntries in this message. The length of the sequence will not be more than 255
The Following Fields Repeat NoMDEntries times				
279	MDUpdate Action	uInt32	1	Indicates how book structure changed. See table: *24 – MD UpdateActions*
269	MDEntryType	single byte string	1	Side of book for this entry. See table: *20 – MD Entry Type*
1023	MDPriceLevel	uInt32	1	Level number of book for this *MDEntry*.
270	MDEntryPx	decimal	5	Price at this level and side of book.
	MDVolume Entries			Sequence of Volume Entries for this price level.

Each *MDEntry* contains a nested sequence of *MDVolumeEntries*. Each *MDVolumeEntry* has the following sub-fields:

Field ID	Field Name	Type	Length (Bytes)	Comments
21000	NoMDVolume Entries	length (unsigned int)	1	Number of *MDVolumeEntries* for this *MDEntry*. The length of this sequence cannot be more than 4 (the number of valid volume types)
The Following Fields Repeat NoMDVolumeEntries times				
21001	MDVolume Type	uInt32	1	See table: *21 – MD Volume* Types
271	MDEntrySize	uInt32	4	Quantity for this type of volume.

23 – Incremental Refresh Template

```
<template   name="MDIncRefresh"             id="18">
    <string     name="MessageType"              id="35"     byteLength="1"
                                                            value="X"/>
    <uInt32     name="MsgSeqNum"                id="34"     byteLength="4"/>
    <uInt32     name="ClassKey"                 id="21004"  byteLength="4"/>
    <uInt32     name="SecurityID"               id="48"     byteLength="4"/>
    <uInt32     name="RptSeq"                   id="83"     byteLength="4"/>
    <uInt32     name="SecurityTradingStatus"    id="326"    byteLength="1"/>
    <uInt32     name="PriceType"                id="423"    byteLength="1"/>
    <sequence   name="MDEntries">
        <length     name="NoMDEntries"              id="268"    byteLength="1"/>
        <uInt32     name="MDUpdateAction"           id="279"    byteLength="1"/>
        <string     name="MDEntryType"              id="269"    byteLength="1"/>
        <uInt32     name="MDPriceLevel"             id="1023"   byteLength="1"/>
        <decimal    name="MDEntryPx"                id="270"    byteLength="5"/>
        <sequence   name="MDVolumeEntries">
            <length     name="NoMDVolumeEntries"        id="21000"  byteLength="1"/>
            <uInt32     name="MDVolumeType"             id="21001"  byteLength="1"/>
            <uInt32      name="MDEntrySize"              id="271"    byteLength="4"/>
        </sequence>
    </sequence>
</template>
```

24 – MD UpdateActions

0	Insert (New level)
1	Change (volume change of existing price)
2	Delete
5	Overlay (price and volume change of existing level)

5.4 Security Status Message – Template ID 19

Security Status messages contain the SecurityTradingStatus for a product. They are sent when the trading state of a product changes, but there is no structural book change.

In situations when a product state changes and there is a structural book change, an Incremental Refresh message will be sent instead of a Security Status message along with a changed SecurityTradingStatus. The Security Status message is sent only when the trading state for a product changes but there is no structural book change that occurs simultaneously.

Security Status messages may be used at any time to indicate a product's trading state change, but are most likely to be used at market open for products where there are no orders in the book that can be traded at the market open. In such cases, the Security Status message indicates the product is open for trading, but the book remains structurally the same as it was prior to the market open.

Note: The RptSeq is incremented when a SecurityStatus message is sent (as with an IncRefresh message) so that missing messages can be detected on a per-product basis and used for recovery purposes. See the section *Processing For Start-Up and Data Recovery* for details on how recovery process can be implemented.

25 – Security Status Message Structure

Field ID	Field Name	Type	Length (Bytes)	Comments
	MessageHeader			See *6 - Message Header*, MessageType = "f"
21004	ClassKey	uInt32	4	Product Class key
48	SecurityID	uInt32	4	Product ID
83	RptSeq	uInt32	4	Per-product seq number. See the section *Processing For Start-Up and Data Recovery* for how this can be used.
326	SecurityTrading Status	uInt32	1	See table: *18 - Security Trading* Status

26 – Security Status Template

```
<template   name="MDSecurityStatus"          id="19">
    <string     name="MessageType"               id="35"     byteLength="1"  value="f"/>
    <uInt32     name="MsgSeqNum"                 id="34"     byteLength="4"/>
    <uInt32     name="ClassKey"                  id="21004"  byteLength="4"/>
    <uInt32     name="SecurityID"                id="48"     byteLength="4"/>
    <uInt32     name="RptSeq"                    id="83"     byteLength="4"/>
    <uInt32     name="SecurityTradingStatus"     id="326"    byteLength="1"/>
</template>
```

5.5 Heartbeat Message (Line Integrity Message) - Template ID 16

This message contains only a Message Header. The heartbeat will repeat at a regular interval on all data channels. It may be used to confirm operation of a channel during times when no normal market activity exists for a channel (such as pre-opening).

Field ID	Field Name	Type	Length (Bytes)	Comments
	Message Header			**See** 6 - Message Header, MessageType = "0" (zero)

27 - Heartbeat Template

```
<template   name="Heartbeat"        id="16">
    <string     name="MessageType"      id="35"     byteLength="1"  value="0"/>
    <uInt32     name="MsgSeqNum"        id="34"     byteLength="4"/>
</template>
```

6 Examples

This section contains examples of various Snapshot and Incremental Refresh scenarios and how to process the book using those examples. For all examples, the max number of levels of the book transmitted is 5.

6.1 Example Snapshot Full Refresh With Two-Sided Book

Data Type	Field	Value	Comment
sbSTRING:	MessageType	'W'	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1829	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
sbSTRING:	RefreshIndicator	N	
SEQUENCE:	NoMDEntries	Len=7	7 MDEntries
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.07	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	1	Volume = 1
UINT8:	MDVolumeType	1	Customer Limit Volume
UNIT32:	MDEntrySize	1	Volume = 1
STRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	2	Level 2
uDECIMAL:	MDEntryPx	0.05	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	341	
UINT8:	MDVolumeType	1	Customer Limit Volume
UNIT32:	MDEntrySize	244	
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.11	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	Total Limit Volume
UINT32:	MDEntrySize	41	
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	2	Level 2
DECIMAL:	MDEntryPx	0.12	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type

Data Type	Field	Value	Comment
UINT8:	MDVolumeType	0	Total Limit Volume
UINT32:	MDEntrySize	48	
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	3	Level 3
DECIMAL:	MDEntryPx	0.28	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	
UINT32:	MDEntrySize	11	
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	4	Level 4
DECIMAL:	MDEntryPx	0.38	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	Total Limit Volume
UINT32:	MDEntrySize	10	
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	5	Level 5
DECIMAL:	MDEntryPx	2.28	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	Total Limit Volume
UINT32:	MDEntrySize	10	

This snapshot defines a book with the following contents:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont-ingent Vol**	**Cust Cont-ingent Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont-ingent Vol**	**Cust Cont-ingent Vol**
1	0.07	1	1	0	0	1	0.11	41	0	0	0
2	0.05	341	244	0	0	2	0.12	48	0	0	0
						3	0.28	11	0	0	0
						4	0.38	10	0	0	0
						5	2.28	10	0	0	0

In this example, there are no bid entries beyond level 2.
Volumes that are not contained in the message are implied to be zero.

6.2 IncRefresh Insert Level – Customer Order

In this example, assume the current state of the book:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	0.07	1	1	0	0	1	0.11	41	0	0	0
2	0.05	341	244	0	0	2	0.12	48	0	0	0
						3	0.28	11	0	0	0
						4	0.38	10	0	0	0
						5	2.28	10	0	0	0

A **customer limit order to sell 10 at 0.13** is entered:

Data Type	**Field**	**Value**	**Comment**
sbSTRING:	MessageType	W	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1830	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
SEQUENCE:	NoMDEntries	Len=1	1 MDEntry
UINT8:	MDUpdateAction	0	Insert
sbSTRING:	MDEntryType	‘1’	Ask
UINT8:	MDPriceLevel	3	Level 3
DECIMAL:	MDEntryPx	0.13	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	10	Volume = 10
UINT8:	MDVolumeType	1	Customer Limit Volume
UNIT32:	MDEntrySize	10	Volume = 10

Two volume types are sent because total limit volume includes customer limit volume. Volumes not sent for a price level are implied to be zero.

Insert the price 0.13 at level 3 of the ask side of the book as indicated in the message, and roll off level 5 automatically. No instruction to roll off level 5 is sent. The resulting book is:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	0.07	1	1	0	0	1	0.11	41	0	0	0
2	0.05	341	244	0	0	2	0.12	48	0	0	0
						3	**0.13**	**10**	**10**	**0**	**0**
						4	0.28	11	0	0	0
						5	0.38	10	0	0	0

6.3 IncRefresh Delete + Change– Trade Top Of Book and Part of Level 2

A market order to sell for quantity 10 is entered. This trades with:

the 0.07 customer bid volume 1 and

part of the 0.05 volume 341 and 244.

Assume a market model that has customer priority.

Data Type	Field	Value	Comment
sbSTRING:	MessageType	'W'	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1830	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
SEQUENCE:	NoMDEntries	Len=2	2 MDEntries
UINT8:	MDUpdateAction	2	Delete
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.07	
SEQUENCE:	NoMDVolumeEntries	Len=0	0 Volume Types
UINT8:	MDUpdateAction	1	Change
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1 (after delete processed)
DECIMAL:	MDEntryPx	0.05	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	332	Volume = 332
UINT8:	MDVolumeType	1	Customer Limit Volume
UNIT32:	MDEntrySize	235	Volume = 235

The delete entry contains no volumes. Its NoMDVolumeEntries is zero.
Two volumes are sent for the change entry because total limit volume includes customer limit volume.
The market order of 10 trades with the 0.07 volume 1 entry, which eliminates that entry, then the remaining 9 trades with the (formerly level 2) entry which is level 1 after the delete instruction.
The resulting book is:

Bid						**Ask**					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	**0.05**	**332**	**235**	**0**	**0**	1	0.11	41	0	0	0
						2	0.12	48	0	0	0
						3	0.13	10	10	0	0
						4	0.28	11	0	0	0
						5	0.38	10	0	0	0

6.4 IncRefresh Change + Overlay Level – Replace Top Of Book

The **best quote is changed** from **0.05 x 97** – **0.11 x 41** to:**0.05 x 90** – **0.10 x 10**.
(Formerly the 0.05 bid price professional volume was 332 total – 235 customer == 97 professional).
Assume the ask price of 0.11 was quote volume for one market maker who changes their ask quote to 0.10, causing the 0.11 price to disappear.

Data Type	Field	Value	Comment
sbSTRING:	MessageType	'W'	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1830	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
SEQUENCE:	NoMDEntries	Len=2	2 MDEntries
UINT8:	MDUpdateAction	1	Change
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.05	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	325	Volume = 325
UINT8:	MDVolumeType	1	Customer Limit Volume
UNIT32:	MDEntrySize	235	Volume = 235
UINT8:	MDUpdateAction	5	Overlay (replace, like delete/insert)
sbSTRING:	MDEntryType	'1'	Ask
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.10	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	10	Volume = 10

Prior to update, bid volume was 332 total – 235 customer == 97 professional (quote in this case).
This changes to 90, reducing total limit volume by 7, resulting in 332 – 7 == 325

Prior to update, best ask was 0.11 x 41. Assume this was one market maker's quote whose ask changes to 0.10 x 10. This could use 2 instructions: (delete 0.11 + insert 0.10 x 10), but one instruction is more efficient, so *overlay* is sent, indicating to replace the entire level, including price.
The resulting book is:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	0.05	**325**	235	0	0	**1**	**0.10**	**10**	**0**	**0**	**0**
						2	0.12	48	0	0	0
						3	0.13	10	10	0	0
						4	0.28	11	0	0	0
						5	0.38	10	0	0	0

6.5 IncRefresh Change – Trade Customer and Part Professional Volume

A **limit order to sell 250 at 0.05** is entered. Assume customer priority.
This order trades out completely with the 235 customer volume, leaving 15, which trades with some of the remaining professional volume expressed in total limit volume. The result is the customer volume becomes zero, leaving only professional volume.

Data Type	Field	Value	Comment
sbSTRING:	MessageType	'W'	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1830	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
SEQUENCE:	NoMDEntries	Len=2	2 MDEntries
UINT8:	MDUpdateAction	1	Change
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.05	
SEQUENCE:	NoMDVolumeEntries	Len=1	1 Volume Type
UINT8:	MDVolumeType	0	Total Limit Volume
UNIT32:	MDEntrySize	75	Volume = 75

The customer volume is traded out completely, leaving only professional volume in the total limit volume.
Zero volumes are not transmitted and are implied to be zero. There is no separate MDVolumeEntry to indicate the customer volume. Its absence implies it is zero.
The resulting book is:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	0.05	**75**	**0**	0	0	**1**	**0.10**	**10**	**0**	**0**	**0**
						2	0.12	48	0	0	0
						3	0.13	10	10	0	0
						4	0.28	11	0	0	0
						5	0.38	10	0	0	0

6.6 IncRefresh Insert – Contingent Volume at Top of Book

A **customer all-or-none (AON) order to buy 50 at 0.07** is placed.

Data Type	Field	Value	Comment
sbSTRING:	MessageType	'W'	
UINT32:	MsgSeqNum	4209855	
UINT32:	ClassKey	69223595	
UINT32:	SecurityID	1426985904	
UINT32:	RptSeq	1830	
UINT8:	SecurityTradingStatus	17	Ready To Trade
UINT8:	PriceType	3	Fixed Amount
SEQUENCE:	NoMDEntries	Len=2	2 MDEntries
UINT8:	MDUpdateAction	0	Insert
sbSTRING:	MDEntryType	'0'	Bid
UINT8:	MDPriceLevel	1	Level 1
DECIMAL:	MDEntryPx	0.07	
SEQUENCE:	NoMDVolumeEntries	Len=2	2 Volume Types
UINT8:	MDVolumeType	2	Total Contingent Volume
UNIT32:	MDEntrySize	50	Volume = 50
UINT8:	MDVolumeType	3	Customer Contingent Volume
UNIT32:	MDEntrySize	50	Volume = 50

The 0.07 bid price is better than the top limit bid price, so this will be inserted at the bid-side top of book. The 0.05 price is pushed down.
The resulting book is:

Bid						Ask					
Level	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**	**Level**	**Price**	**Total Limit Vol**	**Cust Limit Vol**	**Total Cont Vol**	**Cust Cont Vol**
1	**0.07**	**0**	**0**	**50**	**50**	1	0.10	10	0	0	0
2	0.05	75	0	0	0	2	0.12	48	0	0	0
						3	0.13	10	10	0	0
						4	0.28	11	0	0	0
						5	0.38	10	0	0	0

7 Processing For Start-Up and Data Recovery

CSM Level 2 feeds transmit incremental book update messages, which cannot be processed accurately unless the current state of the book is valid and correct. Incremental messaging requires some special processing for start-up and recovery from dropped messages.

This section describes how recovery processing can be implemented.

7.1 Channel Level Processing

Each message sent on a channel causes its *MsgSeqNumber* to increment by one. To detect missing data at the channel-level, compare each incoming *MsgSeqNumber* with the last received *MsgSeqNumber* + 1. If the incoming *MsgSeqNumber* is not equal to the (last received *MsgSeqNumber* + 1), data is missing from the channel.
Regardless of whether missing data is detected or not, the *MsgSeqNumber* of the incoming message should be stored associated with the channel so subsequent missing data can be detected.

When missing data is detected at the channel-level, all products that were received over that channel should be treated as "suspect", meaning their book *may* be invalid. At the time missing data is detected on a channel, there is no way to know which product's data is missing, therefore, all products received from that channel should be treated as though the book for those products may be incorrect.

Products marked as suspect or possibly-incorrect should remain in that state until a Snapshot for that product is received, or until other means can be used to determine that the product is no longer potentially incorrect (see *Product Level IncRefresh and Security* Status Recovery Processing below).

7.1.1 Example Pseudo Code for Channel Level Processing

```
int expectedSeqNum = channel.lastSeqNum + 1;
if (expectedSeqNum != incomingMsg.msgSeqNum) {
    Product[] productList = productCache.getAllProductsForChannel(channel);
    for(int i = 0; i < productList.length; i++) {
        productList[i].suspect = true;
    }
}
channel.lastSeqNum = incomingMsg.msgSeqNum;
productCache.processMessage(incomingMsg);
```

7.2 Product Level IncRefresh and Security Status Recovery Processing

Each IncRefresh and Security Status message sent has a field, *RptSeq*, which is a per-product sequence number. *RptSeq* is incremented by 1 for each IncRefresh or Security Status message sent for a particular product. This can be used to determine which specific products are missing data and which products are not missing data after a channel-level data loss is discovered. To use this field, for each IncRefresh or Security Status message received compare the *RptSeq* of the incoming message with the last received *RptSeq* + 1 for that product.

If the incoming *RptSeq* is not equal to the product's (last received *RptSeq* + 1), it means that data is missing for this product. In this event you must wait for a Snapshot for the product to re-establish the correct state of the book and to obtain the current *RptSeq* for the product. The data for the product should treated as incorrect, and subsequent processing of IncRefresh and Security Status messages for this product should be suspended until the Snapshot message arrives.

If, however, the incoming IncRefresh's or Security Status's *RptSeq* is equal to the product's (last received *RptSeq* + 1), it means that this IncRefresh or Security Status is in the proper sequence and no data has been missed for the product. The IncRefresh or Security Status can be applied, and its *RptSeq* should be stored with the product. Also, if a prior channel-level data loss detection caused this product to be treated as "suspect" or as possibly incorrect, this state can be undone and data for the product can be considered to be correct.

7.2.1 Example Pseudo Code for Product Level IncRefresh Recovery Processing

```
Product product = productCache.findProduct(incRefresh.classKey,
                                           incRefresh.securityID);
if (product == null) {
    /*
     * Possible error condition. SecurityDef may have been dropped
     * or not yet received (are you just starting up?)
     * Or perhaps you do not care about this product?
     * Whatever the case, we cannot process this message.
     */
    return;
}

int expectedRptSeq = product.rptSeq + 1;

// Note: Do not use > comparison. rptSeq may be < prior values seen.
// If the Cboe sender fails over, rptSeq numbers are reset.

if (incRefresh.rptSeq != expectedRptSeq) {
    product.suspect = true;
} else {
    // Product is actually in sequence after all. Undo suspect if set
    if (product.suspect) {
        product.suspect = false;
    }
    product.applyIncrememtaUpdate(incRefresh);
    product.rptSeq = incRefresh.rptSeq;
}
```

7.2.2 Example Pseudo Code for Product Level Security Status Recovery Processing

```
Product product = productCache.findProduct(secStatus.classKey,
                                           secStatus.securityID);
if (product == null) {
    /*
     * Possible error condition. SecurityDef may have been dropped
     * or not yet received (are you just starting up?)
     * Or perhaps you do not care about this product?
     * Whatever the case, we cannot process this message.
     */
    return;
}

int expectedRptSeq = product.rptSeq + 1;

// Note: Do not use > comparison. rptSeq may be < prior values seen.
// If the Cboe sender fails over, rptSeq numbers are reset.

if (secStatus.rptSeq != expectedRptSeq) {
    product.suspect = true;
} else {
    // Product is actually in sequence after all. Undo suspect if set
    if (product.suspect) {
        product.suspect = false;
    }
    product.setSecurityStatus(secStatus);
    product.rptSeq =secStatus.rptSeq;
}
```

7.3 Product Level Snapshot Recovery Processing

Each Snapshot message also has the *RptSeq* field which is the per-product sequence number. However, the *RptSeq* for a Snapshot message is the *current state* of the *RptSeq*; it is NOT incremented when a Snapshot is sent. If no data was missed, the Snapshot's *RptSeq* will be equal to the *RptSeq* value sent in the latest IncRefresh or SecurityStatus message for a product. This can be used in certain conditions:

When a Snapshot message arrives, first examine the *RefreshIndicator*. If the *RefreshIndicator* is equal to 'Y', it means that the CSM Level 2 system has determined that a data disruption has occurred between it and all receivers (such as a system failure). The Snapshot cannot be processed normally; you **must** process the Snapshot by clearing the contents of the book and re-applying all data contained in the Snapshot.

If the Snapshot's *RefreshIndicator* is NOT equal to 'Y', it means the Snapshot can be processed normally. Compare the Snapshot *RptSeq* to the product's *RptSeq* that is cached. If equal, it means that you have received all prior IncRefresh and Security Status updates for the product and you do not need to process the Snapshot. If the Snapshot's *RptSeq* is NOT equal to the *RptSeq* value for the product, it means data for this product was missed and the Snapshot should be applied. To apply it, clear the contents of the book for the product, and re-build it using the contents of the Snapshot.

In either case when a Snapshot is applied, the *RptSeq* should be stored with the product so the IncRefresh processing works correctly, and the "suspect" state should be undone and the product considered correct.

7.3.1 Example Pseudo Code for Product Level IncRefresh Recovery Processing

```
Product product = productCache.findProduct(snapshot.classKey,
                                           snapshot.securityID);
if (product == null) {
    /*
     * Possible error condition. SecurityDef may have been dropped
     * or not yet received (are you just starting up?)
     * Or perhaps you do not care about this product?
     * Whatever the case, we cannot process this message.
     */
    return;
}

if (snapshot.refreshIndicator == 'Y') {
    // Must process snapshot, sender knows there was a data disruption.
    product.clearBook();
    product.applySnapshot(snapshot);
    product.rptSeq = snapshot.rptSeq;
    product.suspect = false;
} else {
    // Normal processing.
    // Avoid snapshot processing if we have all updates for this product

    if (snapshot.rptSeq == product.rptSeq) {
        return;
    }
    product.clearBook();
    product.applySnapshot(snapshot);
    product.rptSeq = snapshot.rptSeq;
    product.suspect = false;
}
```

8 Appendix A – Multicast Group and Port Information

Refer to the CFN Network Specifications document for complete multicast group and port information.

Cboe Non Strategy Options and Strategy Options Primary Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.32/28	Cboe CSM2 Non Strategy Prod A Groups		A			
224.4.7.32	Cboe Book Depth	'0'	A	63900	170.137.128.253	170.137.144.0/26
224.4.7.33	Cboe Book Depth	'1'	A	63901	170.137.128.253	170.137.144.0/26
224.4.7.34	Cboe Book Depth	'2'	A	63902	170.137.128.253	170.137.144.0/26
224.4.7.35	Cboe Book Depth	'3'	A	63903	170.137.128.253	170.137.144.0/26
224.4.7.36	Cboe Book Depth	'4'	A	63904	170.137.128.253	170.137.144.0/26
224.4.7.37	Cboe Book Depth	'5'	A	63905	170.137.128.253	170.137.144.0/26
224.4.7.38	Cboe Book Depth	'6'	A	63906	170.137.128.253	170.137.144.0/26
224.4.7.39	Cboe Book Depth	'7'	A	63907	170.137.128.253	170.137.144.0/26
224.4.7.40	Cboe Book Depth	'8'	A	63908	170.137.128.253	170.137.144.0/26
224.4.7.41	Cboe Book Depth	'9'	A	63909	170.137.128.253	170.137.144.0/26
224.4.7.42	*Reserved for future use*		A	63910	170.137.128.253	170.137.144.0/26
224.4.7.43	*Reserved for future use*		A	63911	170.137.128.253	170.137.144.0/26
224.4.7.44	*Reserved for future use*		A	63912	170.137.128.253	170.137.144.0/26
224.4.7.45	Cboe Securities Definition		A	63913	170.137.128.253	170.137.144.0/26
	Cboe CSM2 Strategies Prod A Groups					

224.4.7.46	Cboe Strategies Book Depth	'0'	A	63914	170.137.128.253	170.137.144.0/26
224.4.7.47	Cboe Strategies Securities Definition		A	63915	170.137.128.253	170.137.144.0/26

Cboe Non Strategy Options and Strategy Options Backup Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.160/28	Cboe CSM2 Non Strategy Prod B Groups		B			
224.4.7.160	Cboe Book Depth	'0'	B	63932	170.137.128.254	170.137.144.64/26
224.4.7.161	Cboe Book Depth	'1'	B	63933	170.137.128.254	170.137.144.64/26
224.4.7.162	Cboe Book Depth	'2'	B	63934	170.137.128.254	170.137.144.64/26
224.4.7.163	Cboe Book Depth	'3'	B	63935	170.137.128.254	170.137.144.64/26
224.4.7.164	Cboe Book Depth	'4'	B	63936	170.137.128.254	170.137.144.64/26
224.4.7.165	Cboe Book Depth	'5'	B	63937	170.137.128.254	170.137.144.64/26
224.4.7.166	Cboe Book Depth	'6'	B	63938	170.137.128.254	170.137.144.64/26
224.4.7.167	Cboe Book Depth	'7'	B	63939	170.137.128.254	170.137.144.64/26
224.4.7.168	Cboe Book Depth	'8'	B	63940	170.137.128.254	170.137.144.64/26
224.4.7.169	Cboe Book Depth	'9'	B	63941	170.137.128.254	170.137.144.64/26
224.4.7.170	*Reserved for future use*		B	63942	170.137.128.254	170.137.144.64/26
224.4.7.171	*Reserved for future use*		B	63943	170.137.128.254	170.137.144.64/26
224.4.7.172	*Reserved for future use*		B	63944	170.137.128.254	170.137.144.64/26
224.4.7.173	Cboe Securities Definition		B	63945	170.137.128.254	170.137.144.64/26
	Cboe CSM2 Strategies Prod B Groups		B			
224.4.7.174	Cboe Strategies Book Depth	'0'	B	63946	170.137.128.254	170.137.144.64/26

224.4.7.175	Cboe Strategies Securities Definition		B	63947	170.137.128.254	170.137.144.64/26

Cboe_EXT Non Strategy Options Primary Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.56/31	Cboe_EXT CSM Non Strategy Prod A Groups		A			
224.4.7.56	Cboe_EXT Book Depth	0	A	63990	170.137.128.253	170.137.144.0/26
224.4.7.57	Cboe_EXT Securities Definition		A	63991	170.137.128.253	170.137.144.0/26

Cboe_EXT Non Strategy Options Backup Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.184/31	Cboe_EXT CSM Non Strategy Prod B Groups		B			
224.4.7.184	Cboe_EXT Book Depth	0	B	63994	170.137.128.254	170.137.144.64/26
224.4.7.185	Cboe_EXT Securities Definition		B	63995	170.137.128.254	170.137.144.64/26

Cboe_EXT Strategy Options Primary Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.58/31	Cboe_EXT CSM Strategies Prod A Groups		A			
224.4.7.58	Cboe_EXT Strategies Book Depth	0	A	63992	170.137.128.253	170.137.144.0/26
224.4.7.59	Cboe_EXT Strategies Securities Definition		A	63993	170.137.128.253	170.137.144.0/26

Cboe_EXT Strategy Options Backup Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.186/31	Cboe_EXT CSM Strategies Prod B Groups		B			
224.4.7.186	Cboe_EXT Strategies Book Depth	0	B	63996	170.137.128.254	170.137.144.64/26
224.4.7.187	Cboe_EXT Strategies Securities Definition		B	63997	170.137.128.254	170.137.144.64/26

Cboe Non Strategy Flex Options and Flex Strategy Primary Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.48/30	Cboe CSM2 Flex Options Prod A Groups		A			
224.4.7.48	Cboe Strategies Book Depth	'0'	A	63980	170.137.128.253	170.137.144.0/26
224.4.7.49	Cboe Strategies Securities Definition		A	63981	170.137.128.253	170.137.144.0/26
	Cboe CSM2 Flex Strategy Prod A Groups					
224.4.7.50	Flex Strategies Book Depth	'0'	A	63982	170.137.128.253	170.137.144.0/26
224.4.7.51	Flex Strategies Securities Definition		A	63983	170.137.128.253	170.137.144.0/26

Cboe Non Strategy Flex Options and Flex Strategy Backup Groups

Group	Description	Target Location ID	A or B	Port	RP	Source Networks
224.4.7.176/30	Cboe CSM2 Flex Options Prod B Groups		B			
224.4.7.176	Cboe Strategies Book Depth	'0'	B	63984	170.137.128.254	170.137.144.64/26
224.4.7.177	Cboe Strategies Securities Definition		B	63985	170.137.128.254	170.137.144.64/26
	Cboe CSM2 Flex Strategy Prod B Groups					
224.4.7.178	Flex Strategies Book Depth	'0'	B	63986	170.137.128.254	170.137.144.64/26
224.4.7.179	Flex Strategies Securities Definition		B	63987	170.137.128.254	170.137.144.64/26

9 Appendix B – Hex Dump Wire Format Examples

Examples are provided here of a hex-dump of various messages and what they look like on the wire.

9.1 Understanding Hex Dump Diagrams

Through this section you will see hexadecimal dumps of packets from a multicast channel. Each of the examples will be in a similar format.

```
00000000  CD580000 00030000 0134C8E5 A8992B6C  |.X.......4....+l|
00000010  031FFE00 00006600 00006401 00        |......f...d..   |
```

Each line represents 16 bytes of data. The first column of numbers is the zero-based byte offset of the first hex byte from the packet. Following this is up to 4 groups of hexadecimal representation. Every 2 characters in this area represents one byte. Spaces are for formatting purposes only. Characters between pipes ("|") are ASCII representations of the hex data. Periods indicate non-printable values.

The first 16 bytes of each packet are the packet header.

9.2 Packet Header Example

1 - Packet Header Hex Dump

```
00000000  01038B00 00013A83 F905430B 00000015  |......:...C.....|
```

2 - Packet Header Decoded

```
BlkVersion(01) == 1
BlockSize(038B) == 907
SendTime(00013A83F90543)==1,350,833,866,051 ms since Jan 1, 1970 UTC ==
      2012-10-21 10:37:46.051 CDT
MsgsInBlock(0B) == 11
FirstMsgSeq(00000015) == 21
```

9.3 Security Definition Message

An example Options Security Definition for ADBE 49 Put, expiring 2012-10-20.

3 - Security Definition Hex Dump

```
00000000  01034E00 0001375B C7509B0A 0024419A  |..N...7[.P...$A.|
00000010  00530D64 0024419A 034F5054 43044144  |.S.d.$A..OPTC.AD|
00000020  42450134 1C1A88EA 207643D3 00000000  |BE.4.... vC.....|
00000030  013305BC 03FD0000 BF6800FC 000004E2  |.3.......h......|
00000040  FE000F42 36FE0000 012CFE00 000005FE  |...B6....,......|
00000050  00000001 00000441 44424502 43530000  |.......ADBE.CS..|
00000060  00640000 530D6400 24419B03 4F505443  |.d.             |
```

4 - Security Definition Decoded

```
Message#(1) MsgSize(83)
TemplateId(13)   AppMsg: MDSecurityDefinition Type: d
  FieldType        FieldName                          Id     Value
  sbSTRING:        MessageType                        35     d
  UINT32:          MsgSeqNum                          34     2376090
  STRING:          SecurityType                       167    OPT
  sbSTRING:        SecurityExchange                   207    C
  STRING:          Symbol                             55     ADBE
  STRING:          TargetLocationID                   143    4
  UINT32:          ClassKey                           21004  471501034
  UINT32:          SecurityID                         48     544621523
  UINT64:          MaturityDate                       541    20121020
  UINT8:           PriceType                          423    3
  uDECIMAL:        StrikePrice                        202    49.000
  UINT8:           PutOrCall                          201    0
  uDECIMAL:        MinimumStrikePriceFraction         21005  0.1250
  uDECIMAL:        MaxStrikePrice                     21006  9999.90
  uDECIMAL:        PremiumBreakPoint                  21007  3.00
  uDECIMAL:        MinimumAbovePremiumFraction        21008  0.05
  uDECIMAL:        MinimumBelowPremiumFraction        21009  0.01
  UINT8:           ExerciseStyle                      21010  0
  STRING:          CurrencyCode                       996
  STRING:          UnderlyingSymbol                   311    ADBE
  STRING:          UnderlyingType                     310    CS
  UINT32:          ContractSize                       231    100
  SEQUENCE:        NoLegs                             555    Len=0
```

9.4 Heartbeat Message

5 - Heartbeat Hex Dump

```
00000000  01001800 000135A7 00C6C901 00000F95  |......5.........|
00000010  00081030 00000F95                    |...0....        |
```

6 - Heartbeat Decoded

```
Message#(1) MsgSize(8)
TemplateId(16)   AppMsg: Heartbeat Type: 0
  FieldType        FieldName                          Id     Value
  sbSTRING:        MessageType                        35     0
  UINT32:          MsgSeqNum                          34     3989
```