CEF**®**

**CEF® Core Multicast**

**Interface Specification**

Version: 1.0

Release: July 2013

Last update: 10-Jul-13

Revision History

This section refers to the major changes of this version in comparison to the previous version.

|  |  |  |
| --- | --- | --- |
| Version | Revision Highlight | Document Reference |
| 1.0 | Initial document |  |

Table of contents

[1 Introduction 6](#_Toc361179589)

[1.1 How to use this specification guide 6](#_Toc361179590)

[1.2 Relevant documents 6](#_Toc361179591)

[2 Communication Interface 7](#_Toc361179592)

[2.1 IP Multicast dissemination graphic 7](#_Toc361179593)

[2.2 IP Multicast processing 8](#_Toc361179594)

[2.2.1 IP Multicast process 8](#_Toc361179595)

[2.2.2 Sender ID handling 8](#_Toc361179596)

[2.3 Multicast Products 9](#_Toc361179597)

[3 Message Description 10](#_Toc361179598)

[3.1 Contents of a Datagram 10](#_Toc361179599)

[3.2 General Message Structure 11](#_Toc361179600)

[3.2.1 Message Length Handling 11](#_Toc361179601)

[3.2.2 Folder/Field Organisation 13](#_Toc361179602)

[3.3 CEF® Core Multicast Message Structure 18](#_Toc361179603)

[3.4 Time resolution 19](#_Toc361179604)

[4 Message processing and Message structure 20](#_Toc361179605)

[4.1 Start-up 20](#_Toc361179606)

[4.2 Synchronisation between Service A and B 20](#_Toc361179607)

[4.3 Synchronisation between Snapshot and Delta Channel 20](#_Toc361179608)

[4.4 Host Failover processing 20](#_Toc361179609)

[4.5 Folder types and Message types 21](#_Toc361179610)

[4.5.1 Folder Types 21](#_Toc361179611)

[4.5.2 Header Message 21](#_Toc361179612)

[4.5.3 Refresh Quote Listing Data Message 22](#_Toc361179613)

[4.5.4 Tick-By-Tick Listing Data Message 23](#_Toc361179614)

[4.5.5 Heartbeat Data Message 24](#_Toc361179615)

[4.6 Compression 24](#_Toc361179616)

[5 Appendices 25](#_Toc361179617)

[5.1 Data Messages Summary 25](#_Toc361179618)

[5.2 System Folder List 25](#_Toc361179619)

[5.3 System Field Definition 26](#_Toc361179620)

[5.4 Data Value Presentation 27](#_Toc361179621)

[5.4.1 Boolean 27](#_Toc361179622)

[5.4.2 Byte stream 27](#_Toc361179623)

[5.4.3 BCD Date and Time 27](#_Toc361179624)

[5.4.3.1 BCD\_DATE\_TIME 27](#_Toc361179625)

[5.4.3.2 BCD\_TIME 28](#_Toc361179626)

[5.4.4 CHAR 29](#_Toc361179627)

[5.4.5 DNUM 29](#_Toc361179628)

[5.4.6 Integer 31](#_Toc361179629)

[5.4.7 Network Byte Order 31](#_Toc361179630)

[5.4.8 String 31](#_Toc361179631)

[5.5 Message Folders - constant values table 32](#_Toc361179632)

[5.6 Instrument Type 33](#_Toc361179633)

[5.7 Compression 34](#_Toc361179634)

Figures

[**Figure 2‑1 IP Multicast Network** 7](#_Toc361071318)

[**Figure 3‑1 Contents of a Datagram** 10](#_Toc361071319)

[**Figure 3‑2 Contents of a Heartbeat Message** 10](#_Toc361071320)

[Figure 3‑3 Basic Message Structure 11](#_Toc361071321)

[Figure 3‑4 Message Length Structure 11](#_Toc361071322)

[Figure 3‑5 MLB-1 Structure 11](#_Toc361071323)

[Figure 3‑6 Folder/Field Organization 13](#_Toc361071324)

[Figure 3‑7 Field/Folder Structure 13](#_Toc361071325)

[Figure 3‑8 Field Header Structure 14](#_Toc361071326)

[Figure 3‑9 Example of a Block Length Field 17](#_Toc361071327)

[Figure 3‑10 General Structure of a CEF® Core Multicast Message 18](#_Toc361071328)

[Figure 5‑1 Decimal Number Structure 29](#_Toc361071329)

Tables

[Table 3‑1 MLB Bits Value 12](#_Toc361071330)

[Table 3‑2 Field GRP Definition 15](#_Toc361071331)

[Table 3‑3 Data Type Definition for Fixed Length Field 15](#_Toc361071332)

[Table 3‑4 Data Type Definition for Variable Length Field 16](#_Toc361071333)

[Table 3‑5 Layout of a Message ID Folder 18](#_Toc361071334)

[Table 3‑6 All available Message IDs supported by the CEF® Core Multicast System 19](#_Toc361071335)

[Table 5‑1 Data Messages Summary 25](#_Toc361071336)

[Table 5‑2 System Folder List 25](#_Toc361071337)

[Table 5‑3 Field ID Definition 26](#_Toc361071338)

[Table 5‑4 Boolean Value Definition 27](#_Toc361071339)

[Table 5‑5 Decimal Number Data Types 30](#_Toc361071340)

[Table 5‑6 Examples of DNUMs 30](#_Toc361071341)

[Table 5‑7 Integer Data Types 31](#_Toc361071342)

[Table 5‑8 Code Set Definition 31](#_Toc361071343)

# Introduction

CEF®[[1]](#footnote-1) Core Multicast is the real-time data feed of Deutsche Boerse - Market Data + Services. This documentation provides all details which are necessary to use the CEF® Core Multicast service.

The feed is produced by Deutsche Boerse (hereinafter referred to as DBAG) data dissemination system (hereinafter referred to as the CEF® Core Multicast System) and covers real-time price and index data.

## How to use this specification guide

Chapter 1 contains a short introduction, the specification guide and references to related documents.

Chapter 2 describes the Multicast Communication Interface with which clients are connected to the dissemination system.

Chapter 3 describes concepts of IP Multicast message descriptions.

Chapter 4 describes the Structure and Message processing logic.

Chapter 5 contains appendices. Those appendices at the end of the document provide a reference for additional information.

## Relevant documents

|  |  |
| --- | --- |
| Item | Document |
| 1 | CEF® Core Multicast – Connection Parameter |
| 2 | CEF® Core Multicast – Fields & Products |

# Communication Interface

This section shows the IP multicast communication between clients and the CEF® Core Multicast System.

## IP Multicast dissemination graphic



**Figure 2‑1 IP Multicast Network**

Several Hosts (servers publishing Multicast messages) are running for reasons of scalability and load balancing. Updates for one product are always disseminated by the same host.

Besides the online hosts there are standby hosts in order to provide fail over functionality.

## IP Multicast processing

### IP Multicast process

The following types of IP Multicast messages are provided by CEF® Core Multicast System:

* Snapshot channel for Listing information messages (static listing attributes as well as the   
  current image of a listing, published periodically throughout the day)  
  including heartbeats
* Delta channel for tick-by-tick real-time messages  
  including heartbeats

Each Multicast host sends each delta datagram and the snapshot datagram twice to two multicast channels, providing Service A and Service B. Both Service A and Service B consist of the same content, they are redundant. This is valid for delta datagrams as well as for snapshot datagrams. Users have to connect to Service A or Service B.

**Recommendation:   
Connect to both services for reasons of gap detection or failover handling.**

### Sender ID handling

Each host adds a SENDER\_ID to each message, i.e. different SENDER\_IDs for different message types are used:

SENDER\_ID 1 for data messages on the delta channel (both multicast streams)

SENDER\_ID 2 for data messages on the snapshot channel (both multicast streams)

SENDER\_ID 3 for heartbeat messages on the delta channel (both multicast streams)

SENDER\_ID 4 for heartbeat messages on the snapshot channel (both multicast streams)

Thus the recipient of CEF® Core Multicast may detect host failover situations; in this case the SENDER\_ID will change (cf. chapter 4).

## Multicast Products

The following products are available via CEF® Core Multicast:

* ASIA\_DBAG\_INDEX
* ASIA\_EUREX
* ASIA\_SPOT
* ASIA\_STOXX\_INDEX

Each of these products is available via individual combination of IP address and port number. The valid IP addresses with Port numbers are available within the CEF® Core Multicast Connection Parameter document (see chapter 1.2 ‘relevant documents’).

# Message Description

## Contents of a Datagram

A message consists of two segments. The first segment contains the length information for the datagram. The second segment contains the message content.

A multicast datagram sent by the gateway can appear in two variants: Datagrams containing compressed or uncompressed data:

Length of following block with compressed data

Data either compressed or uncompressed

Data Message 1

Header Message

Data Message 2

…

Data Message n

**Figure 3‑1 Contents of a Datagram**

If the most significant bit of the first byte of the datagram (first byte in the Length Block) is set, the datagram contains compressed data. In this case, the first two bytes of the datagram are containing the length of the following block with compressed data. This length encoding is the same as the length encoding at the beginning of CEF® Core Multicast messages. (cf. chapter 3.2)

In this case the content of the data has to be de-compressed (to be considered as one block). The content itself, i.e. the individual messages as indicated in the graphic, will only be available after de-compression.

Heartbeat datagrams are containing only a header message and the CEF® Core Multicast heartbeat telegram:

Heartbeat Message

Header Message

**Figure 3‑2 Contents of a Heartbeat Message**

## General Message Structure

### Message Length Handling

A message consists of two segments:

****

Figure 3‑3 Basic Message Structure

Length Block

This Block contains the information about the length of the whole message (details below)

Data Block

This segment contains the actual contents of the message. Its size is determined by the *Length Block*.

Message Length (information in the Length Block)  
The Length Block contains 1 to 3 bytes. They are MLB-1, MLB-2 and MLB-3 as shown in the diagram below. The encoded message length is NOT including the size of the length field itself.

Byte MLB-1 is mandatory whilst the other two are optional, depending on the data in MLB-1.



Figure 3‑4 Message Length Structure

Byte MLB-1 has the following structure:



Figure 3‑5 MLB-1 Structure

*CF Bit* determines whether compressed data is stored in the *Message Content* segment.

If set to OFF, the *Message Content* segment contains uncompressed data.

If set to ON, the *Message Content* segment contains compressed data.

**Remark:**

**Compression within CEF**® **Core Multicast is always applied on datagram level. Thus, the CF bit of an individual message will always be 0.**

*MLB Bits* determine the length of the *Message Length* segment.

|  |  |  |
| --- | --- | --- |
| MLB Bits | | Description |
| 0 | 0 | *Message Content* segment size is stored in MLB-1 only.  Length bits: size of Message Content segment  MLB-2: not required  MLB-3: not required |
| 0 | 1 | *Message Content* segment size is stored in MLB-1 and MLB-2.  Length bits: most significant bits (bit 9-13)  MLB-2: remaining least significant bits (bit 1-8)  MLB-3: not required |
| 1 | 0 | *Message Content* segment size is stored in MLB-1, MLB-2 and MLB-3.  Length bits: most significant bits (bit 17-21)  MLB-2: remaining least significant bits (bit 9-16)  MLB-3: remaining least significant bits (bit 1-8) |

Table 3‑1 MLB Bits Value

### Folder/Field Organisation

The content of a message is stored inside the Data Block segment which is organized based on the folder and field concept. The diagram below is an example of a message with the content organized in folders and fields:



Figure 3‑6 Folder/Field Organization

A folder is a logical collection of related fields. If necessary, it may also contain folders as shown in the diagram above.

A field is the basic unit that carries the actual data. Depending on the type of data, the data in the field is encoded.

Folder/Field Structure

Folder and field share the same structure which contains two segments:



Figure 3‑7 Field/Folder Structure

**Field Content**

If it is a field the Field Content segment stores the actual field data.

If it is a folder the Field Content segment stores the definition of fields/folders inside this folder.

To determine whether the Field Content segment contains data for a field or for a folder, refer to the Field Header segment. The whole Field Header segment is also called 16-bit FID.

**Field Header**

The field header representing the 16-bit FID is the unique Field Identifier in CEF® Core Multicast and has the following structure:



Figure 3‑8 Field Header Structure

**Field GRP** designates whether it is a field or a folder.

|  |  |  |
| --- | --- | --- |
| Group Bits | | Description |
| 0 | 0 | The structure is a zero length field. |
| 0 | 1 | This structure is a fixed size field. |
| 1 | 0 | This structure is a variable size field. |
| 1 | 1 | This structure is a folder. |

Table 3‑2 Field GRP Definition

Zero length fields denote a field without data. The size of the *Field Content* segment is 0.

Fixed size field denotes a field whose size is implied by its data type. Refer to Field TYPE for more details.

Variable size field denotes a field with variable length data. Both the size and the actual data are stored together in the *Field Content* segment. Refer to section ‘Size of Variable Length Field’ for more details on processing such fields.

A folder is treated as a variable size field. Its definition is stored in the *Field Content* segment whose size is determined by the same method applied to the variable size field. Refer to section ‘Size of Variable Length Field’ for more details.

**Field TYPE** indicates the type of data contained in a field.

If field GRP indicates that it is a zero length field, field TYPE is ignored.

If field GRP indicates that it is a fixed length field, then field TYPE is used to check on the type of data contained in this field in the following table.

| Type Bits | Data Type | Size of Field Content |
| --- | --- | --- |
| 0x00 | CHAR | 1 |
| 0x01 | Integer – INT16 | 2 |
| 0x02 | Integer – INT32 | 4 |
| 0x03 | Boolean | 1 |
| 0x04 | BCD Date and Time (microseconds) | 10 |
| 0x05 | BCD Date | 4 |
| 0x06 | BCD Time (microseconds) | 6 |
| 0x07 | DNUM16 | 3 |
| 0x08 | DNUM32 | 5 |
| 0x09 | DNUM64 | 9 |
| 0x0D | INT64 | 8 |

Table 3‑3 Data Type Definition for Fixed Length Field

If field GRP indicates it is a variable length field, field TYPE is used to check the type of data in the field in the following table. For details on size, please refer to the section ‘Size of Variable Length Field’.

|  |  |  |
| --- | --- | --- |
| Type Bit | Data Type | Size of Field Content |
| 0x00 | Byte Stream | Variable |
| 0x01 | String | Variable |

Table 3‑4 Data Type Definition for Variable Length Field

If field GRP indicates it is a folder, field TYPE is set to 0x00.

More information on the data types mentioned in the above tables can be found in the chapter “Data Value Presentation”.

**Field NUMBER** indicates the content in a field e.g.

* 16-bit FID 0x6002: Field of Data Type DNUM32 BEST\_ASK
* 16-bit FID 0x4010: Field of Data Type CHAR INSTRUMENT\_TYPE
* 16-bit FID 0x6008: Field of Data Type DNUM32 CLOSE
* 16-bit FID 0x582A: Field of Data Type BCD Time HIGH\_TIME

**Size of Variable Length Field**

* Size and contents are stored in the Field Content segment.
* Size information is stored in the Block Length Field.
* Block Length Field is always at the beginning of the Field Content segment.
* Block Length Field is a multi-bit integer presented as a series of bytes.
* The first byte of Block Length Field contains the most significant bits.
* The last byte of a Block Length Field contains the least significant bits.
* Each byte in the Block Length Field uses 7 bits only. The high order bit is reserved to indicate the end of a Block Length Field. If it is set to 1, the next following byte is part of the Block Length Field. If it is set to 0, this byte is the last byte of a Block Length Field.
* Size is determined by combining all bits in the Block Length Field to form a long integer.   
  The integer value represents the data’s actual size following the Block Length Field.
* The diagram below shows an example of Block Length Field.
* In the example, the first byte of the Block Length Field indicates that a second byte is to follow (high bit set to 1). The second byte indicates that this is the last byte of the Block Length Field (high bit set to 0). The precise length of the data following the Block Length Field is indicated by the 14-bit values stored across two bytes. The most significant bits (denoted here by ‘X’) are in the first byte and the least significant bits (denoted here by ‘Y’) are in the second byte.

1

X

X

X

X

X

X

X

Block Length

Field (2 bytes)

Bit 15 - 9

0

Y

Y

Y

Y

Y

Y

Y

Bit 7 - 1

**First byte**

**Second byte**

Figure 3‑9 Example of a Block Length Field

## CEF® Core Multicast Message Structure

A message is defined as a collection of folders where the Message ID Folder is always the leading folder.



Figure 3‑10 General Structure of a CEF® Core Multicast Message

The layout of the Message ID folder is presented below:

|  |  |
| --- | --- |
| Message ID Folder | |
| Field (Field ID) | Description |
| \* MESSAGE\_ID (4331) | Message ID |
| … |  |
| … |  |

Table 3‑5 Layout of a Message ID Folder

Remarks:

\* – Mandatory Field

Message ID is always the first field in the folder. Depending on the nature of the message there may be other fields that follow it. It is stored as a CHAR in the Message ID Field. Please refer to the Appendix for the actual values.

The Message ID Folder carries information that defines the nature of this message. In particular, it carries a Message ID field that provides the following indications:

* Message class to which it belongs
* Message format
* Message handling

The Message ID defines the layout of a message and the possible folder for a message. The table below illustrates all available Message IDs which are currently supported by the CEF® Core Multicast System.

|  |  |
| --- | --- |
| Message ID | Description |
| **Messages Generated by the CEF**® **Core Multicast System** | |
| HEADER\_MESSAGE | Header message |
| MID\_HEARTBEAT | Heartbeat for system activity |
| MID\_LISTING\_DATA | Market data of listing |

Table 3‑6 All available Message IDs supported by the CEF® Core Multicast System

## Time resolution

Within the field BCD\_TIME a time resolution for one microsecond is available. This means, the BCD\_TIME field can contain 1/1000000 of a second.

This time resolution appears all BCD\_TIME and BCD\_DATE\_TIME fields.

# Message processing and Message structure

## Start-up

Each customer application should join the Snapshot multicast channel and Delta multicast channel on service A and B for each ordered product within the CEF® Core Multicast Service. After joining the multicast channels the application starts with processing the data from Delta and Snapshot channels.

## Synchronisation between Service A and B

The datagrams on Snapshot and Delta channel will be sent on Service A and Service B. Each datagram message begins with a Header message. This message includes the field DATAGRAM\_SEQUENCE, which can be used together with the field SENDER\_ID to identify an individual datagram on service A and B. So if a datagram is lost on service A it may be taken from Service B.

## Synchronisation between Snapshot and Delta Channel

The Refresh Quote Listing Data and the Tick-By-Tick Listing Data message include the sequence field UPDATE\_SEQUENCE. This field can be used for synchronisation between Snapshot and Delta messages. This field will be incremented with each update (each delta message) per Listing.

The recipient’s software should check the consecutive order of messages received. If there is a message gap detected, the snapshot messages may be used in order to re-synchronize the image of a listing.   
The content of UPDATE\_SEQUENCE in each snapshot represents the message counter of the latest delta message that has been taken into account for this snapshot.   
Thus it is possible to re-synchronize the image of a listing by taking a snapshot message as new basis and further on process the delta messages with higher values (starting with the next one) in the field UPDATE\_SEQUENCE.

## Host Failover processing

After Host failover all SENDER\_IDs are changed and the sequence fields (DATAGRAM\_SEQUENCE, UPDATE\_SEQUENCE) are reset to zero. So if the SENDER\_ID is changed on a given message type a host failover has occurred. The new host will start sending data on Snapshot and Delta channel with a new SENDER\_ID. The Refresh Quote Listing Data messages from Snapshot channel are including the current price information.

## Folder types and Message types

### Folder Types

The following Folder types are used:

* Message ID Folder
* Listing Identification Folder
* Quote Folder

### Header Message

A Header Message contains a Folder type ‘Message Id Folder’.

Message ID folder contains the fields MESSAGE\_ID, SENDER\_ID and DATAGRAM\_SEQUENCE.

|  |  |  |  |
| --- | --- | --- | --- |
| **Purpose:** | This message contains information about the datagram | | |
| **Layout:** | **Field Name** | | **Value** |
| \* | **Message ID Folder** | | |
| \* | MESSAGE\_ID | | MID\_DATAGRAM\_HEADER |
| \* | SENDER\_ID | | 1, 5, 9, … for update datagrams  2, 6, 10, … for snapshot datagrams  3, 7, 11, … for heartbeat datagrams on the update channel  4, 8, 12, … for heartbeat datagrams on the snapshot channel |
| \* | DATAGRAM\_SEQUENCE | | 1, 2, …  Incremented for each datagram per SENDER\_ID |
|  |  |  | |
| **Remarks:** | 1. Mandatory fields and folders are labelled with an asterisk. 2. This message is always sent as the first message within a datagram. | | |

### Refresh Quote Listing Data Message

|  |  |  |
| --- | --- | --- |
| **Purpose:** | This message contains quote refreshes, is published via snapshot channel. | |
|  |  | |
| **Layout:** | **Field Name** | **Value** |
| \* | **Message ID Folder** | |
| \* | MESSAGE\_ID | MID\_LISTING\_DATA |
| \* | OUTBOUND\_MESSAGE\_KEY | Delta channel:  Original value from CEF® Core Multicast  Snapshot channel: *OUTBOUND\_MESSAGE\_KEY of last integrated delta message of the listing.* |
| \* | UPDATE\_SEQUENCE | Delta channel:  Incremented for each update per listing per SENDER\_ID  Snapshot channel: *UPDATE\_SEQUENCE of last integrated delta message of the listing* |
| \* | **Listing Identification Folder** | |
| \* | LISTING\_DATA\_SUBJECT | LIST\_DATA\_QUOTE\_REFRESH |
| \* | INSTRUMENT\_TYPE |  |
| \* | SYMBOL | ISIN |
|  | EXPIRY | For futures and options |
|  | OPTION\_CATEGORY | ‘C’ for Call  ‘P’ for Put |
|  | ORIGINAL\_STRIKE\_PRICE | For options |
|  | GENERATION\_NUMBER | For options |
| \* | SOURCE\_NAME |  |
| \* | **Quote Folder** | |
| \* | <any source-specific quote fields> |  |
|  |  |  |
| **Remarks:** | 1. Mandatory fields and folders are labelled with an asterisk. | |

### Tick-By-Tick Listing Data Message

|  |  |  |
| --- | --- | --- |
| **Purpose:** | This message contains quote refreshes, is published via delta channel. | |
|  |  | |
| **Layout:** | **Field Name** | **Value** |
| \* | **Message ID Folder** | |
| \* | MESSAGE\_ID | MID\_LISTING\_DATA |
| \* | OUTBOUND\_MESSAGE\_KEY | Original value from CEF® Core Multicast |
| \* | UPDATE\_SEQUENCE | Incremented for each update per listing per SENDER\_ID |
| \* | **Listing Identification Folder** | |
| \* | LISTING\_DATA\_SUBJECT | LIST\_DATA\_TICK-BY-TICK |
| \* | INSTRUMENT\_TYPE |  |
| \* | SYMBOL | ISIN |
|  | EXPIRY | For futures and options |
|  | OPTION\_CATEGORY | ‘C’ for Call  ‘P’ for Put |
|  | ORIGINAL\_STRIKE\_PRICE | For options |
|  | GENERATION\_NUMBER | For options |
| \* | SOURCE\_NAME |  |
| \* | **Quote Folder** | |
| \* | <any source-specific quote fields> |  |
|  |  |  |
| **Remarks:** | 1. Mandatory fields and folders are labelled with an asterisk. | |

### Heartbeat Data Message

|  |  |
| --- | --- |
| Purpose: | This message informs clients that the CEF® Core Multicast Service is up and running. |

|  |  |  |
| --- | --- | --- |
| Layout: | Field Name | Value |
| \* | Message ID Folder | |
| \* | MESSAGE\_ID | MID\_HEARTBEAT |
| \* | Heartbeat Folder | |
| \* | SYSTEM\_TIME |  |

|  |  |
| --- | --- |
| Remarks: | 1. Mandatory folders and fields are labelled with a leading asterisk. 2. This message is transmitted every 5 seconds. |

## Compression

Data compression is possible. Therefore the ‘Zlib’ algorithm will be used. Compressed message datagrams appear when the message length exceeds the length limit. Thus it is necessary to check the header, first 2 bytes, of each message for compression information.  
In contrast compressions in TCP/IP environments, where the ‘compression dictionary’ is valid for the whole session here in multicast dissemination the ‘compression dictionary’ needs to be reset after every datagram!

The CEF® Core Multicast System (each individual host) decides whether data shall be compressed or not. This will be done based on overall load and data amount. Compressed data can follow uncompressed data and vice versa. In case of compression one or more CEF® Core Multicast messages are taken to one single block of data. This block is subjected to compression. This refers to the CEF® Core Multicast General Message Structure. The result has to be processed like one Message Content Segment. A Message Length Segment is provided in front of the Content Segment. In this Message Length Segment the CF Bit is set. This indicates that the Message content segment contains compressed data.

The client Application has to inspect the CF-Bit when parsing the Message Length Segments of a CEF® Core Multicast datagram. If this Bit is not set it not compressed CEF® Core Multicast datagram. If the CF-Bit is set, the message content segment is a block of compressed data that has to be expanded before further processing. Therefore the chosen decompression method has to be implemented. This will result in a series of one or more CEF® Core Multicast messages each consisting of Message Length Segment and Message Content Segment. Before the application uses the decompression algorithm the internal dictionary must be reset.

CEF® Core Multicast uses inflate algorithm of zlib-implementation. A free implementation as C-Source is given at <http://www.gzip.org/zlib/> .  
(refer to chapter 5.7 for an example for decompression as C-Source implementation)

# Appendices

## Data Messages Summary

| Data Messages |
| --- |
|
| Header Messages (MESSAGE\_ID = MID\_DATAGRAM\_HEADER) |
| SENDER\_ID |
| DATAGRAM\_SEQUENCE |
| Listing Data Messages (MESSAGE\_ID = MID\_LISTING\_DATA) |
| OUTBOUND\_MESSAGE\_KEY |
| UPDATE\_SEQUENCE |
| Listing Data Messages (MESSAGE\_ID = LIST\_DATA\_QUOTE\_REFRESH |
| INSTRUMENT\_TYPE |
| SYMBOL |
| EXPIRY |
| OPTION\_CATEGORY |
| ORIGINAL\_STRIKE\_PRICE |
| GENERATION\_NUMBER |
| SOURCE\_NAME |
| Heartbeat Message (MESSAGE\_ID = MID\_HEARTBEAT) |
| SYSTEM\_TIME |

Table 5‑1 Data Messages Summary

## System Folder List

|  |  |
| --- | --- |
| Folder | 16bit FID |
| LISTING\_IDENTIFICATION\_FOLDER | C2C9 |
| HEARTBEAT\_FOLDER | C2C2 |
| MESSAGE\_ID\_FOLDER | C2CC |
| QUOTE\_FOLDER | C2D2 |

Table 5‑2 System Folder List

## System Field Definition

| Field | Data Type | Description | 16bit FID |
| --- | --- | --- | --- |
| INSTRUMENT\_TYPE | CHAR | A value representing the instrument type of the listing or instrument. | 4010 |
| OUTBOUND\_MESSAGE\_KEY | INT64 | Identification key of a CEF® Core Multicast outbound message (OMK, OMID) | 7762 |
| SOURCE\_NAME | STRING | CEF® Core Multicast source name. Source is a logical grouping of data in the CEF® Core Multicast System. It usually represents the origin or supplier of the data. For example, the market data of a listing is usually supplied by an exchange. | 874E |
| SOURCE\_TIME | BCD Date Time | Time at which the event that was reported by the data source occurred. | 534F |
| SYMBOL | STRING | Symbol of the instrument.  International Security Identification Number (ISIN). | 8751 |
| SYSTEM\_TIME | BCD Time | System time at which the message is generated. | 5B54 |
| TICK\_ID | INT32 | Tick ID – unique to a listing within a day. | 4B55 |
| TICK\_TIME | BCD Date Time | Time of the tick. | 5356 |
| MESSAGE\_ID | CHAR | Message ID   * MID\_LISTING\_DATA * MID\_HEARTBEAT * MID\_DATAGRAM\_HEADER | 4331 |
| SENDER\_ID | INT16 | 1, 5, 9, … for update datagrams  2, 6, 10, … for snapshot datagrams  3, 7, 11, … for heartbeat datagrams on the update channel  4, 8, 12, … for heartbeat datagrams on the snapshot channel | 442F |
| DATAGRAM\_SEQUENCE | INT64 | Incremented for each datagram per SENDER\_ID | 742F |
| UPDATE\_SEQUENCE | INT32 | Incremented for each update per listing per SENDER\_ID | 482F |

Table 5‑3 Field ID Definition

## Data Value Presentation

This chapter summarizes all possible data types supported by the CEF® Core Multicast Service.

### Boolean

Boolean data field contains ONE byte that may have the following possible values:

|  |  |
| --- | --- |
| Value | Meaning |
| 0 | FALSE/NO/OFF |
| 1 | TRUE/YES/ON |

Table 5‑4 Boolean Value Definition

### Byte stream

The byte stream data field contains binary data. Its length is variable. Refer to section ‘Size of Variable Length Data Type’ for details on how to determine the size of a byte stream data field.

### BCD Date and Time

BCD\_TIME will internally be 8 Bytes and BCD\_DATE\_TIME will internally be 12 Bytes long. In order to find all affected fields please cf. CEF® Core Multicast - Fields & Products, filter column ‘Format’ for those fields of data types BCD\_Date\_Time or BCD\_Time.

#### BCD\_DATE\_TIME

This data type encodes both date and time information inside a single field and has the following characteristics:

* 10-byte CEF® Core Multicast-internal to represent both DATE and TIME
* First 4-byte integer is the value for DATE
* Last 6-byte integer is the value for TIME
* Both DATE and TIME are in BCD format.
* DATE offers precision up to the year 9999. BCD format is in “YYYYMMDD”
* TIME is represented in the format “HHMMSSNNNNNN” where

HH is in 24-hour format.

MM is minutes

SS is seconds

NNNNNN values represents microseconds.

* A value of 0xFFFFFFFF in DATE implies NO DATE

A value of 0xFFFFFFFF in TIME implies NO TIME.

Example for BCD\_DATE\_TIME

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date and Time with Microseconds resolution**  09.03.2010 12:34:08.139930 | | | | | | | | | | | | |
| Byte | b1 | b2 | b3 | b4 | b5 | b6 | b7 | b8 | b9 | b10 |  |  |
| Coding | YY | YY | MM | DD | hh | mm | ss | nn | nn | nn |  |  |
| Example | 20 | 10 | 03 | 09 | 12 | 34 | 08 | 13 | 99 | 30 |  |  |

#### BCD\_TIME

* 8-byte CEF® Core Multicast-internal to represent extended TIME down to microseconds
* TIME is in BCD format.

BCD Time format for microseconds

BCD\_TIME format 6 Bytes Overview: Formats and content of time fields

|  |  |  |
| --- | --- | --- |
| Resolution  Context | Microseconds | |
| Readable Display | HH:MM:SS.nnnnnn  15:17:38.123456 | |
| BCD Time Format also corresponds to the time part of BCD Date Time | 1. Byte  2. Byte  3. Byte  4. Byte 5. Byte 6. Byte | HH MM SS nn nn nn |

The table below assumes that only the TIME in BCD format is stored inside the field. Please refer to the section BCD Date Time for details on how time information is encoded.

Example for BCD\_TIME

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BCD\_TIME:  Microseconds representation 12:34:08.139930 | | | | | | | | |
| Byte | b1 | b2 | b3 | b4 | b5 | b6 |  |  |
| Coding | hh | mm | ss | nn | nn | nn |  |  |
| Example | 12 | 34 | 08 | 13 | 99 | 30 |  |  |

### CHAR

CHAR data type field contains single-byte characters only. There are two ways to express its value:

* As a number: value is between 0 – 255.
* As a presentable ASCII character: for example, ‘A’

### DNUM

Decimal Number (DNUM) is another encoding methodology for representing numeric values. This encoding method has the following characteristics:

* It stores a value without precision loss
* It has a fixed size storage
* It supports decimal values only

The diagram below shows the structure of a Decimal Number.



Figure 5‑1 Decimal Number Structure

A DNUM value is composed of two parts:

* Exponent
* Mantissa

Exponent is a single-byte signed integer that defines the order of magnitude of base 10 of the number. It ranges from between -128 and 127. Mantissa is a 2, 4, or 8-byte signed integer that contains the value of the number.

Actual Value = Mantissa \* 10Exponent The CEF® Core Multicast supports three kinds of DNUMs:

|  |  |  |
| --- | --- | --- |
| Type | BYTE | Mantissa Range |
| DNUM16 | 3 | (-215) – (215 - 1) |
| DNUM32 | 5 | (-231 ) – (231 - 1) |
| DNUM64 | 9 | (-263 ) – (263 - 1) |

Table 5‑5 Decimal Number Data Types

**Examples**

The table below shows some examples of DNUMs.

|  |  |  |  |
| --- | --- | --- | --- |
| Actual Value | Type | Representation | |
| Exponent | Mantissa |
| 123.45 | DNUM16 | FE | 30 39 |
| 123450000 | DNUM16 | 04 | 30 39 |
| 123.45 | DNUM32 | FE | 00 00 30 39 |
| -12345.67 | DNUM32 | FE | FF ED 29 79 |
| -123450000 | DNUM32 | 04 | FF FF CF C7 |
| 123.45 | DNUM64 | FE | 00 00 00 00 00 00 30 39 |
| -12345.67 | DNUM64 | FE | FF FF FF FF FF ED 29 79 |
| Actual Value | Type | Representation | |
| Exponent | Mantissa |
| 1234567890.123 | DNUM64 | FD | 00 00 01 1F 71 FB 04 CB |
| -123456789012300 | DNUM64 | 02 | FF FF FE E0 8E 04 FB 35 |

Table 5‑6 Examples of DNUMs

### Integer

The System supports the following kinds of signed integers:

|  |  |  |
| --- | --- | --- |
| Type | BYTE | Range |
| INT16 | 2 | (-215 ) – (215- 1) |
| INT32 | 4 | (-231 ) – (231- 1) |
| INT64 | 8 | (-263 ) – (263- 1) |

Table 5‑7 Integer Data Types

The byte order of the signed integer type follows the standard described in the section Network Byte Order.

### Network Byte Order

When transmitting a numeric data type, the CEF® Core Multicast System adopts the following Network Byte Order standard: **High order byte of high order bits is transmitted first**.

For example, for an INT16 data field with value 0x0F0A, byte 0x0F is transmitted first, followed by byte 0x0A.

### String

The string data type is used to store a text string.

Basic facts:

* If the highest bit of the first byte is OFF, then the whole string is assumed to be an ASCII string.
* If the highest bit of the first byte is ON, then the first byte itself is a code set indicator which indicates the code set of that string (see table below):

|  |  |
| --- | --- |
| Code Set Indicator | Code Set |
| 0x85 | Unicode |
| 0x87 | ISO 8859-1 (Latin-1) |
| 0x88 | ISO 8859-15 (Latin-15) |

Table 5‑8 Code Set Definition

## Message Folders - constant values table

|  |  |
| --- | --- |
| MESSAGE\_ID | Value |
| MID\_HEARTBEAT | 2 |
| MID\_LISTING\_DATA | 9 |
| MID\_DATAGRAM\_HEADER | 16 |

|  |  |
| --- | --- |
| LISTING\_DATA\_SUBJECT | Value |
| LIST\_DATA\_TICK-BY-TICK | 1 |
| LIST\_DATA\_QUOTE\_REFRESH | 4 |

## Instrument Type

|  |  |
| --- | --- |
| **Description (in alphabetical order)** | **Value** |
| Bond | B (66) |
| Bond Basis Trade | Y (89) |
| Certificates | C (67) |
| Energy | V (86) |
| Equity | E (69) |
| Eurex®[[2]](#footnote-2) Double Leg | P (80) |
| Eurex® Strategy | Q (81) |
| Foreign Exchange | X (88) |
| Funds | D (68) |
| Future Contract | F (70) |
| Future Product | G (71) |
| Index | I (73) |
| Interest Rate | J (74) |
| Market State | A (65) |
| Option Contract | O (79) |
| Option Product | U (85) |
| Other Security | S (83) |
| Subscription Rights | R (82) |
| Warrant | W (87) |

**Notes:**

* The value ‘A’ (65) – Market State is the instrument type used for ‘Market Halt’ messages.

## Compression

Compression Example

#include <zlib.h>

static z\_stream stream;

void initialise()

{ int windowBits = -15; // raw inflate

memset(&stream, 0, sizeof(stream));

if (inflateInit2(&stream, windowBits) != Z\_OK)

{ fprintf(stderr, "error calling inflateInit2(); error [%s]\n", stream.msg);

exit(1);

}

}

/\*\*

\* \brief inflate a buffer in one step

\*

\* \param inBuf

\* [in] pointer to a buffer containing compressed bytes

\* \param inLength

\* [in] size of the buffer with compressed bytes

\* \param outBuf

\* [out] pointer to a buffer where uncompressed bytes should be written to

\* \param outLength

\* [in] size of buffer where uncompressed bytes should be written to

\* [out] number of result bytes in the output buffer

\*

\* \return <code>true</code>, if inflate was successful, <code>false</code> otherwise

\*/

bool inflateWrapper(uint8\_t\* inBuf, size\_t inLength, uint8\_t\* outBuf, size\_t\* outLength)

{ inflateReset(&stream); // reset the dictionary

stream.next\_in = (Bytef\*) inBuf;

stream.avail\_in = (uInt) inLength;

stream.next\_out = (Bytef\*) outBuf;

stream.avail\_out = (uInt) (\*outLength);

if (inflate(&stream, Z\_FINISH) != Z\_STREAM\_END)

{ fprintf(stderr, "error calling inflate(); error [%s]\n", stream.msg);

return false;

}

\*outLength = (size\_t) stream.total\_out;

return true;

}

/\*\*

\* \brief calculates length of contents of a CEF® Core Multicast message or a compressed datagram

\*

\* \param cp

\* [in] points to the beginning of the message

\* \param numberOfLengthBytes

\* [out] size of the message length block

\*

\* \return The requested length

\*/

size\_t getLength(uint8\_t\* cp, size\_t\* numberOfLengthBytes)

{ size\_t lengthOfContents = 0;

\*numberOfLengthBytes = ((\*cp >> 5) & 0x03) + 1;

switch ((int)(\*numberOfLengthBytes))

{case 1:

lengthOfContents = \*cp & 0x1f;

break;

case 2:

lengthOfContents = \*cp++ & 0x1f;

lengthOfContents <<= 8;

lengthOfContents += (uint8\_t)\*cp;

break;

case 3:

lengthOfContents = \*cp++ & 0x1f;

lengthOfContents <<= 8;

lengthOfContents += (uint8\_t)\*cp++;

lengthOfContents <<= 8;

lengthOfContents += (uint8\_t)\*cp;

break;

default:

// this is not expected to happen

fprintf(stderr, "found unexpected MLB length encoding\n");

break;

}

return lengthOfContents;

}

/\*

\* \brief processes a datagram

\*

\* \param datagram

\* pointer to a buffer containing a datagram

\* \param length

\* length of datagram

\*/

void process(uint8\_t\* datagram, ssize\_t length)

{

#define OUT\_BUFSIZE 64000

uint8\_t uncompressedDatagram[64000];

size\_t lengthOfContents = 0;

size\_t numberOfLengthBytes = 0;

size\_t outLength = OUT\_BUFSIZE;

uint8\_t\* cp = datagram; // points to next message

if (\*datagram & 0x80)

{

// compression bit is set, uncompress the datagram contents

lengthOfContents = getMessageLength(cp, &numberOfLengthBytes);

if (inflateWrapper(datagram + numberOfLengthBytes, lengthOfContents, uncompressedDatagram, &outLength) == false)

{

// error

...

}

cp = uncomressedDatagram;

}

...

}

1. CEF® is a registered trademark of Deutsche Boerse AG. [↑](#footnote-ref-1)
2. Eurex® is a registered trademark of Deutsche Boerse AG. [↑](#footnote-ref-2)