To: Wilbert Shin wshin@veloxtradingsystems.com
Cc: Dora Zimbra tziambra@veloxtradingsystems.com

From: Art Larson dssinc@voicenet.com

TECHNICAL ANALYSIS FOR DEVELOPMENT OF RECEIVING APPLICATION FOR PROCESSING FOR PROCESSING Eurex Enhanced Broadcast System (EBS 10.0) MESSAGE DATA STREAMS DRAFT 1.3 (Un-Edited) Work in Progress July 28, 2015

Table of Contents

List of Figures

1 Overview

The purpose of this technical memo is to provide information for the design and development of the capability for Velox to support processing of the Eurex Enhanced BroadCast messages. The Eurex Exchange has established a facility known as the Enhanced BroadCast Solution (EBS). The EBS Eurex Server(s) will disseminate market data in via UDP/IP messages to Eurex Clients [Members].

The Fast Protocol Components Diagram in Figure 12 presents to communication heirarchcy interfaces between the Sender [Eurex EBS] and the Receiver [Velox Receiving Application(s)] . The FAST Protocol Reference documents are available at the following location : http://www.fixprotocol.org/fast Some of the applicable reference(s)are included in the Appendices of this document.

At the Velox location :

- The market data presented is in the form of complete order book information broadcasts (snapshot broadcasts) provided at a periodic interval and continuous order book updates (delta broadcasts) in parallel. A synchronization of the broadcasts in the individual streams in order to display the complete book order needs to be performed by the receiving application.
- The Velox EBS Receiving Application(s) will Operate on the Sun Solaris Platform. The program will execute on a Velox Server and be written in C/C++. The for the purposes of this documentation the Velox EBS Receiving Application will be called (EBS_VLXRCV) [Enhanced BroadCast System Velox Receiver]. EBS_VLXRCV will provide default startup parameters. When the program executes the program will provide a set of options to allow the program to statically configure startup parameters.
- An additional feature of EBS_VLXRCV will be to allow the program to read an mailbox. The messages received will be defined to allow the program to be dynamically reconfigured as during execution.
- A separate program will be provided that will provide the user to generate mailbox messages to be constructed and transmitted to EBS_VLXRCV. This program will provide the user and interface dialogue to control the desired operations of EBS_VLXRCV. This configuration program will be called EBS_VLXCMD.
- EBS_VLXRCV will provide mailbox(s) as identified here:
    - EKEBS_CMD_IN – Mailbox to Receive Configuration Commands from EBS_VLXCONFIG
    - EKEBK_MRKT_OUT – Mailbox that contains output Market Feed Data obtained from the Eurex EBS UDP Multicast message traffic.
- An Interface Message Description will be defined in this document to provide details of the message types that can be received from the command input mailbox and the Market Data Feed Output. **[TBD]**

NOTE: An additional consideration is what is the desired output and format for the results produced. Another issue is how to manage and control the program performing all these operations. Most likely one need a separate control program that interface with the Velox Workstation to allow a trader to modify parameters of the program in a dynamic mode of execution.

2 Appendices

2.1 Eurex EBS Broadcast Streams

Broadcast Streams

The EBS interface will disseminate market information in three streams in a live-live concept meaning that the data will be disseminated in three separate streams over two services called service A and service B. The multicast addresses for both these services are disseminated in product reference information. Due to the inherent unreliable nature of the UDP protocol data packets may be lost in the transmission network. Members are advised to join to both the services to reduce the probability of data loss. Every stream will consist of one or more channels and will follow certain rules for dissemination of data.

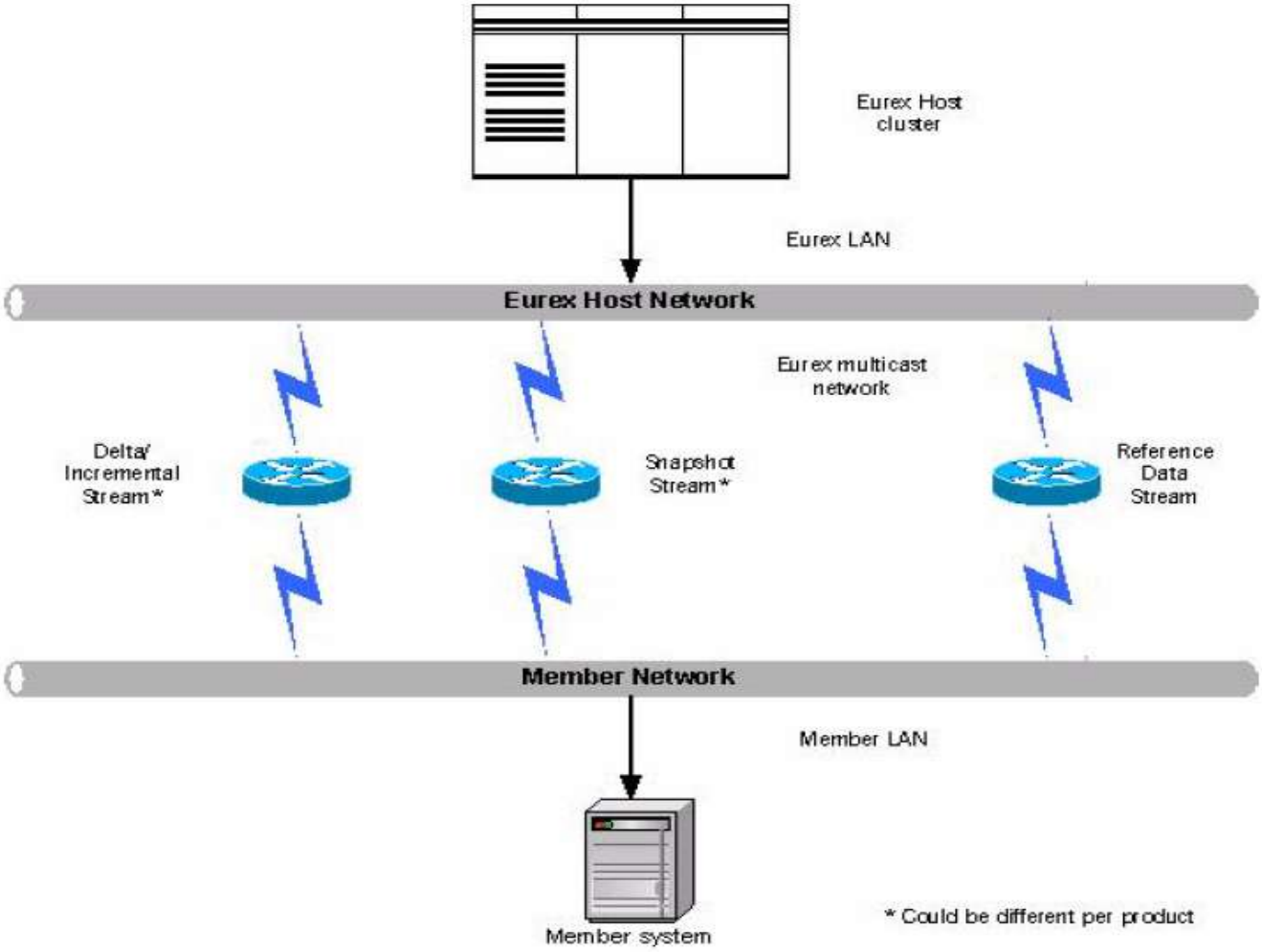


Figure 1 Eurex Broadcast Streams

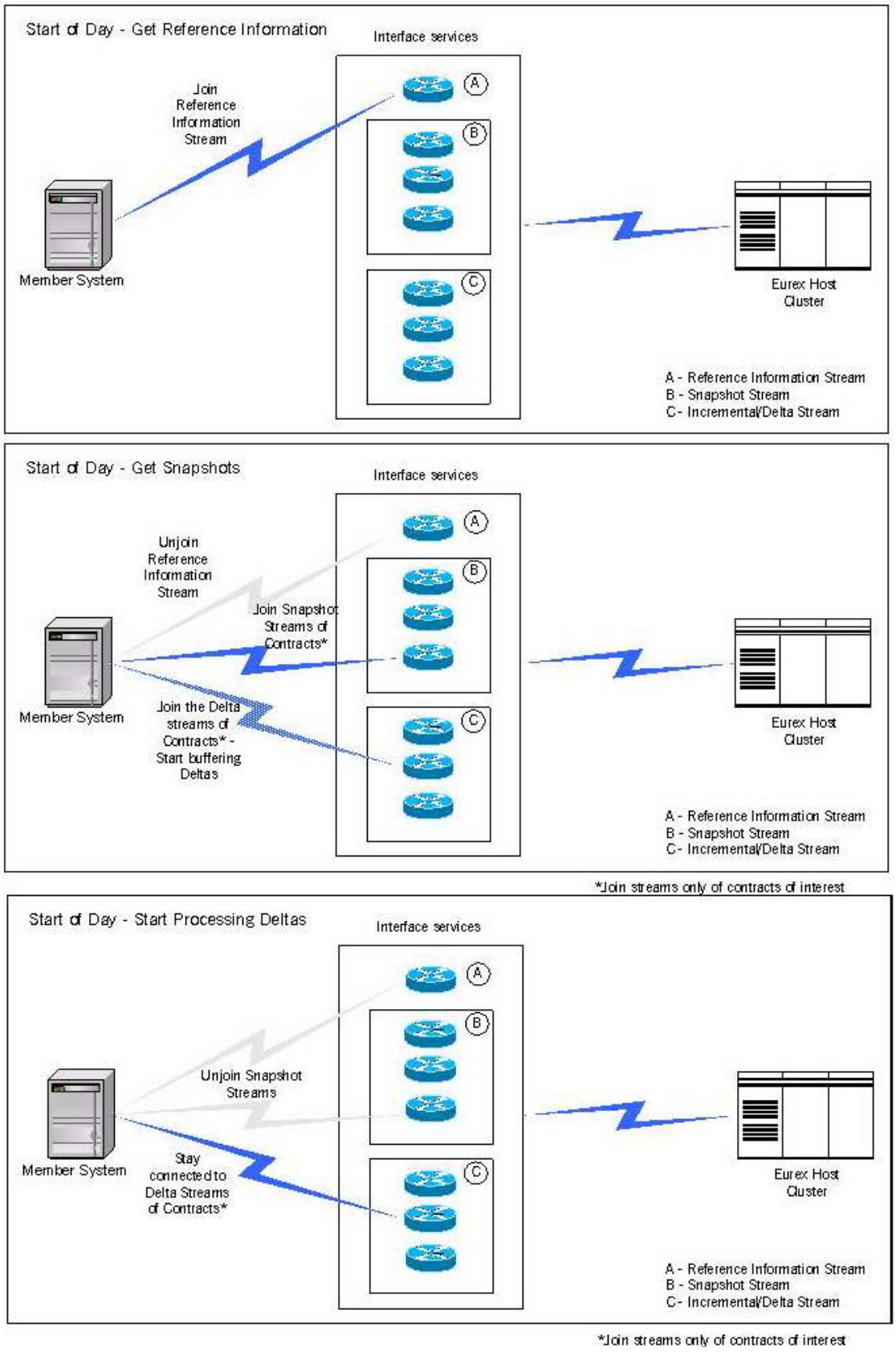


Figure 2 EBS Startup Processing Static - Start of Day

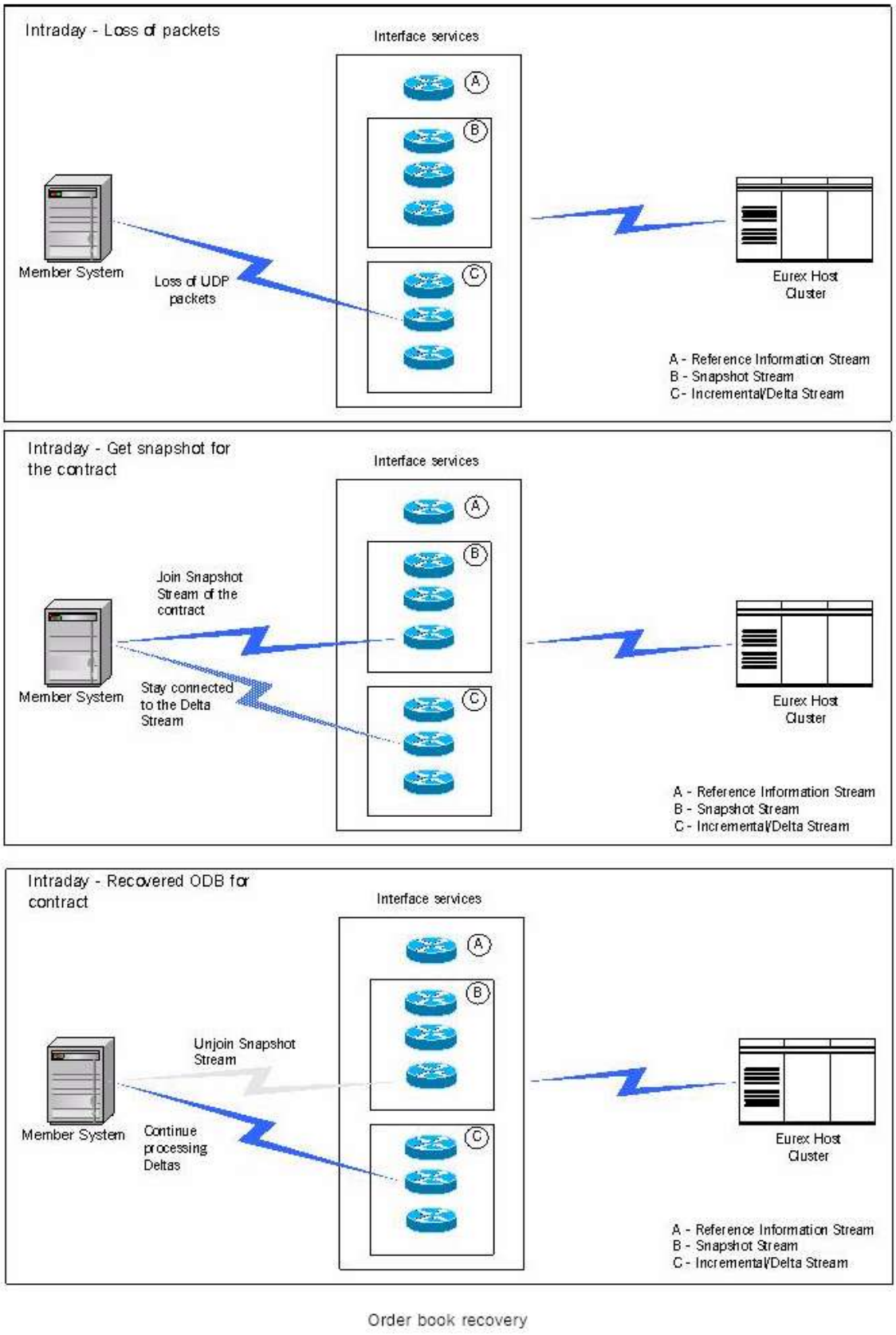


Figure 3 EBS Dynamic Operations (Update Member Copy of Order Book(s))

2.2 Eurex EBS Background Notes

The Enhanced Broadcast Solution will disseminate market data over an IP Multicast based network set up by Eurex. A router capable of forwarding IP Multicast datagrams will be required as the communication equipment for accessing the new interface.

This Quick Start Guide contains additional information not already mentioned in the document “Enhanced Broadcast Solution - Interface Specification”. The purpose of this guide is to familiarise you with the use of the new interface as quickly as possible without focusing on a specific programming language or environment.

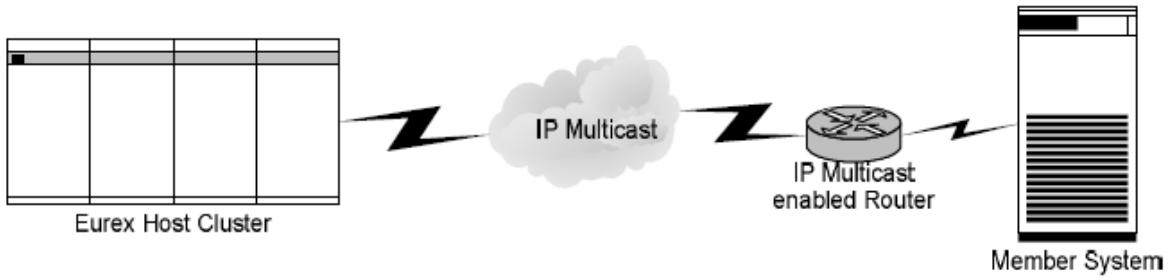


Figure 4 Eurex EBS Interface Diagram

The ‘Enhanced Broadcast Solution’ is a flexible, transparent, UDP based interface which disseminates market information to members over a multicast network. The information disseminated by the EBS interface includes prices, quantities and status information of single and multi-leg (strategies, including combinations) orders as well as quotes. The messaging protocol used by the EBS interface will resemble the Financial Information eXchange (FIX) protocol version 5.0 and will have message formats customized to fit the Eurex business requirements. The EBS interface will conform to the FIX Adapted for Streaming (FAST) protocol version 1.1 principles for efficient bandwidth utilization.

The interface will provide members with the information in form of broadcast streams and members will be able receive information that meets their requirements. The interface is designed so that members will only receive information from the streams they have joined.

The EBS interface will be supported in parallel to the existing VALUES and New Socket Datafeed interfaces. In future, support for the existing New Socket Datafeed interface will be discontinued and all un-netted public market information will be provided only through the ‘Enhanced Broadcast Solution’.

The IP addresses for the respective channels are also disseminated in the reference data stream. The Multicast group / port for the reference data are as follows (also described in the document "Network Access to Exchange Applications", chapter 6.1.6). The IP addresses of the reference data streams are not planned to be changed::

Service / Environment	Service A	Service B
Production	224.0.29.255:50099	224.0.30.255:50099
Simulation	233.49.81.127:50199	233.49.81.255:50199

For verification purposes members will be provided once with a list of initial Multicast addresses of all Products in the simulation environment (please see the attached file). Please note that some of these Multicast addresses will be changed in the course of the Eurex Release 10.0 Member simulation to give the members the opportunity to verify if their applications could handle these changes.

Figure 5 EBS Live-Live Production/Simulation Communication Channels

1.3 General Flow Overview

In order to receive the market data of interest, your application should perform the following steps:

1. Subscribe to well known addresses for reference information streams and receive/store the reference data for the products/contracts and strategies of interest

1. 233.49.81.127:5@@99 static} Live A[1]
2. 233.49.81.255:5@@99 static} Live B

2. Leave the static reference information streams as soon as *all* reference data *of interest* has been received.

3. Select addresses for the channels of interest. There are two different ways in which delta streams are disseminated

- Overlapping Mode
- Non-Overlapping Mode

Currently Eurex will only operate in one of those two modes, but your application should be prepared to support both modes appropriately as specified by the product reference data.

4.

1. Subscribe to the selected set of channels from delta streams
2. Subscribe to the selected set of channels from snapshot streams and receive/store data as applicable for your application

5. Leave a snapshot stream channel as soon as *all* snapshots for *all* contracts, combinations and strategies *of interest* have been received from that channel.

3. ⟲ **4.**[2]

Return to Step 3. as appropriate Return to step 4.2 for out-of-sequnce

Figure 6 EBS Receiving Application General Flow Overview

1.4 Basic Programming Concepts

In general the new enhanced broadcast interface can be accessed with any modern programming language that supports at least the concept of UDP[2] datagram sockets with IP multicast support.
Beyond that, it is recommended to use a programming language that also supports the following two concepts:

1.4.1 Synchronous I/O multiplexing

This principle is based on the `select` function and the `FD_SET`, `FD_CLR`, `FD_ISSET`, `FD_ZERO` macros from the Standard C Library to support synchronous I/O multiplexing. In Java there is a similar mechanism available provided by the `java.nio.channels.Selector` class.

Typically `select` is called with one or more socket sets to determine if at least one of the sockets is in a readable, writable or error state. If there is no timeout mechanism used, the select function will block the calling thread until at least one of the sockets is in one of the requested states. For simplicity we will use the pseudocode notation

$$\texttt{select}(\{s_1, \ldots, s_n\})$$

to determine the next datagram socket from s_1, ..., s_n that is in a readable state.

1.4.2 Multithreading with appropriate synchronisation mechanisms

Multiple threads provide a way of executing one or more tasks concurrently or in parallel within one process.
Because the new enhanced broadcast interface is targeted at high bandwidth and the UDP protocol by its nature does not guarantee reliability or ordering it is highly recommended to read available datagrams with the highest possible priority from sockets and to decouple any further more time consuming processing like decoding etc. from this as much as possible.

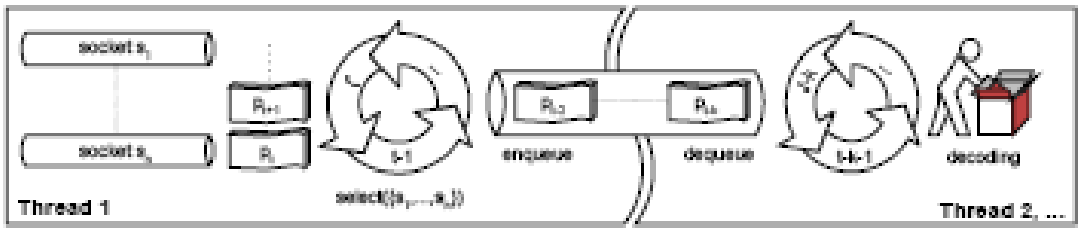


Figure 2: Basic application design

Using an application design as depicted in Figure 2 also provides a simple strategy to scale the application by using one or more threads for the further processing like decoding etc. with respect to the system the application is deployed on.

Note: For simplicity we will assume that we use a single thread for reading data from a common set of sockets as depicted in Figure 2 for the rest of the description.

Figure 7 EBS Receiver Basic Programming Concepts

1.5 Subscribing a Channel

A channel for a set of products and a specific range of order book depths is represented by a multicast address m_i:p_i where m_i is the multicast IP address and p_i is the UDP port (e.g. `239.0.0.16:59000`). Subscribing to such a channel includes the following steps:

1. Try to find an already allocated socket s_i that is bound to *anylocal*: p_i where *anylocal* represents the unspecified local address (`0.0.0.0` in IPv4 or `INADDR_ANY` for BSD-Sockets typically used in the C programming language), or

2. If no such socket was found in step 1 then allocate a new socket s_i, bind it to *anylocal*: p_i[3] and remember it for future use

3. Join the multicast address m_i on socket s_i. In the C programming language with BSD socket support this is typically done as in the following code fragment:

```
struct ip_mreq optval;

memset(&optval, 0, sizeof(optval));
optval.imr_multiaddr.s_addr = inet_addr(m_i);
optval.imr_interface.s_addr = htonl(INADDR_ANY);

setsockopt(s_i, IPPROTO_IP, IP_ADD_MEMBERSHIP,
           (const char*) &optval, sizeof(optval));
```

Note: We will use the following pseudocode notation to summarize the three steps above:

```
s_i = subscribe(m_i:p_i)
```

Figure 8 EBS Receiver Subscribing A Channel

1.6 Decoding UDP datagrams

The payload of a UDP datagram is a sequence of one or more FAST encoded messages. As a detailed description of how to build a FAST decoder does not fit into the scope of this document, the following two documents which cover all aspects of FAST encoding should be consulted and used within the scope of the enhanced broadcast solution :

- *FAST Specification Version 1.1*
  from http://www.fixprotocol.org/fastspec, 2006-12-20,
  written by David Rosenborg
- *Enhanced Broadcast Solution – Interface Specification*
  from Eurex Frankfurt AG, 2007

This document will cover only aspects after the FAST decoding has been applied to a UDP datagram.

Each datagram sent by the new interface will start with the following two messages:

Version Information Message	The value for the versNo field from this message is different for each exchange, each release, and each application and production environment. Each application must be configurable with respect to this field and must discard all datagrams that do not carry the same value for versNo which was configured for the environment for which the application is deployed.
FAST Reset Message	Because of the unreliability of the UDP protocol each message *must* not contain any FAST related references to messages from previous datagrams. The purpose of the FAST Reset message at the beginning of a datagram is to use a standardized way of telling the decoding engine to reset all internal dictionaries and to start a new FAST decoding frame. An implicit effect resulting from this is that the next message following the FAST Reset message must have all presence map bits set to 1 for all fields that allocate a presence map bit.

Note: If an application uses several threads for FAST decoding in parallel as depicted in Figure 2 each thread should have its own decoder instance each sequentially decoding complete datagrams.

Note: For simplicity we will not consider the Version Information and FAST Reset messages anymore explicitly and assume that datagrams are filtered by versNo and FAST decoders are reset accordingly.

Figure 9 EBS Receiver Decoding UDP Datagrams

1.7 Subscription of Reference Information Streams

Reference data is sent cyclically throughout a Eurex trading day in the following way:

⋮
Start*ReferenceInformation*
ReferenceInformation(seqNo=1, ...)
⋮
ReferenceInformation(seqNo=*n*, ...)
End*ReferenceInformation*(noOfMsgs=*n*, ...)
⋮

where *ReferenceInformation* may be one of Product Reference Information or Single Leg Reference Information.

Several single messages might be packed together into a single datagram. The sequence of single messages as shown above will only be available after applying the FAST decoding to each of the datagrams.

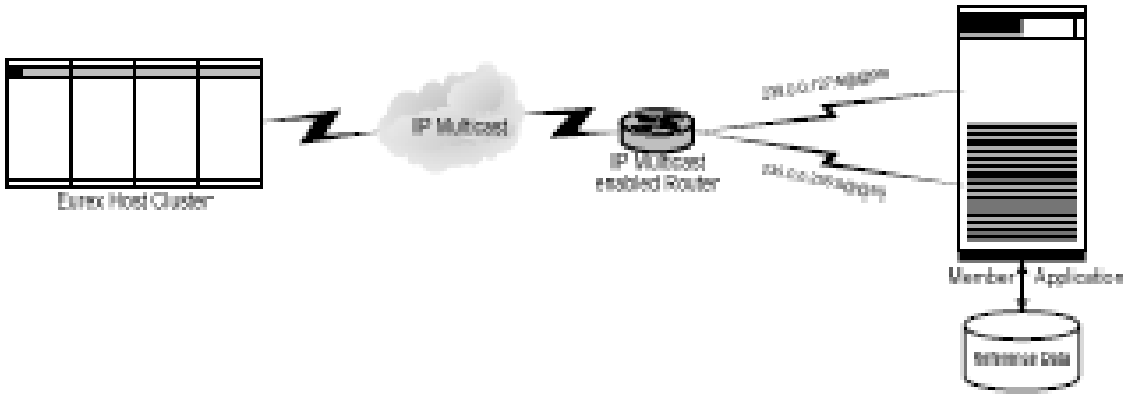


Figure 3: Subscription of reference information streams

Currently there is only a static reference information stream specified:

static reference streams	Product Reference Information and Single Leg Reference Information messages are disseminated over the static reference streams. This information is static and will not change throughout a trading day.

Figure 10 EBS Receiver Obtain Reference Data from Information Stream(s)

1.7.1 Static reference data

Because UDP does not guarantee the delivery or ordering of datagrams, the following simple pattern might be used to receive a complete set of static reference data:

1. s_1 = subscribe(239.0.0.127:50099)
s_1 = subscribe(239.0.0.255:50099)
Add the socket s_1 to the set of sockets as described above (see Figure 2)

2. For each received datagram do ...

3. Decode the sequence of single messages msg_i, i = 1, ..., r.
For each message msg_i, i = 1, ..., r do ...

4.
- If msg_i is a Start*ReferenceInformation* message then
  - optionally store the message for future use
- If msg_i is a *ReferenceInformation*(seqNo=k), $k \in \{1, ..., n\}$ then
  - store the message in a message Set specifically used for *ReferenceInformation* with access key k for future use
- If msg_i is a End*ReferenceInformation* message then
  - remember $noOfMsgs_{ReferenceInformation}$ and store the message for future use
- If $sizeof(Set_{ReferenceInformation}) = noOfMsgs_{ReferenceInformation}$ then
  - goto 5.

5.
- If $sizeof(Set_{Product\ Reference\ Information}) = noOfMsgs_{Product\ Reference\ Information}$ and If $sizeof(Set_{Single\ Leg\ Reference\ Information}) = noOfMsgs_{Single\ Leg\ Reference\ Information}$ then
  - leave(s_1, 239.0.0.127)
  - leave(s_1, 239.0.0.255)
- else
  - goto 2.

Note: Leaving a multicast address m_i on socket s_i in the C programming language with BSD socket support is done in the same way as joining but using option `IP_DROP_MEMBERSHIP` instead of `IP_ADD_MEMBERSHIP` for which we will use the following pseudocode notation:

leave(s_i, m_i)

Figure 11 EBS Receiver - Obtaining Static Reference Data

1.8 Subscription of Snapshot and Delta Streams

Messages on snapshot streams are sent cyclically throughout a trading day delivering snapshots for all currently active contracts, combinations or strategies (the field `cntrExpDate` from the Single Leg Reference Information messages need to be considered):

```
        ⋮
OrderBookSnapshot(cntrId=60001, ...)
OrderBookSnapshot(cntrId=60002, ...)
        ⋮
        ⋮
OrderBookSnapshot(cntrId=60001, ...)
        ⋮
```

Note: Applications should not rely on any specific sequence when listening to snapshot streams.

For delta streams there is no specific protocol defined. Messages are sent as events occur due to the trading activity within the Eurex system.

As soon as the essential parts of information from the reference data that are needed to subscribe to the products, contracts or strategies of your interest are available, the relevant channels or streams can be subscribed:

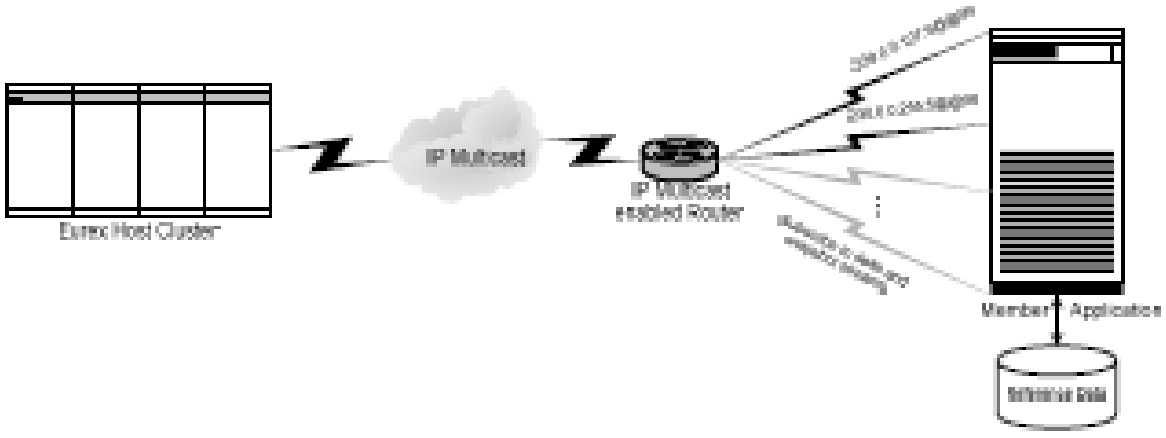


Figure 12 EBS Receiver - Subscription of SnapShot and Delta Streams

If e.g. the Product Reference Information contains the following channel setup for ALV and BMW[4]

ALV	Live A	Live B
Snapshot	239.0.0.16:59033	239.0.0.144:59033
Delta ≤ 0	239.0.0.17:59032	239.0.0.145:59032
Delta ≤ 10	239.0.0.18:59032	239.0.0.146:59032

⋮

BMW	Live A	Live B
Snapshot	239.0.0.16:59033	239.0.0.144:59033
Delta ≤ 0	239.0.0.17:59032	239.0.0.145:59032
Delta ≤ 10	239.0.0.18:59032	239.0.0.146:59032

⋮

and BMW up to price level 10 is required,

the following initial subscriptions depending on the overlapping mode that was sent in the corresponding Service Message (e.g. EndProductReferenceInformation) should be made:

Non-Overlapping Mode	Overlapping Mode
239.0.0.16:59033	239.0.0.16:59033
239.0.0.17:59032	239.0.0.18:59032
239.0.0.18:59032	239.0.0.144:59033
239.0.0.144:59033	239.0.0.146:59032
239.0.0.145:59032	
239.0.0.146:59032	

As can be seed from the example above, there are typically more than one product sent over the same channel (typically products are set up into logical product groups by Eurex).

Note: Applications *can* silently discard snapshot or delta messages for contracts, combinations or strategies that are of no interest, but applications *should not* leave a channel before the snapshot messages for *all* contracts, combinations or strategies of interest have been received.

As soon as all snapshots for contracts, combinations or strategies of interest have been received, the associated channel should be left.

Figure 13 EBS Receiver – Sample Subscription Overlapped and Non-Overlapped Modes

1.9 Out-Of-Sequence Scenarios

Snapshot messages must be correctly positioned between delta messages and vice versa to be able to maintain the order book from an initial snapshot and successive delta messages. Therefore each OrderBookDeltaInformation message has a `seqNum` field that positions it within a chain of successive sequence numbers. There is a chain of sequence numbers for each combination of

- `contractDescription`, channel and `srcId`
- `strategyDescription`, channel and `srcId`

A `contractDescription` identifies a contract by its `cntrId`. A `strategyDescription` identifies a combination or strategy by its attributes `prodId`, `stratType`, `undrPrc` and the group of `strategyLegs`. The channel is identified by its IP multicast address. Each combination of `contractDescription` and channel must have only one associated srcId.

The following scenarios could occur:

1. There is a gap detected from the seqNum of one message to another message with the same `contractDescription`, channel and `srcId` or `strategyDescription`, channel and `srcId`.
2. There is a message with a linked broadcast identifier missing.
3. There is a change in srcId for a combination of `contractDescription` and channel or `strategyDescription` and channel. In this case the `seqNum` will also abruptly restart from 1.

Note: In the first case the client application should consider, that it might always be possible due to the unreliability of the UDP protocol that datagrams might not always be delivered in the same order in that they have been sent, though the client application should maintain a small time window where it waits for outstanding datagrams, before it signals an out-of-sequence situation.

If the client application detects either one of the scenarios above and if a copy of the order book cannot be generated, it should restart subscribing the appropriate snapshot channels as shown in Step 5. from section 1.3 General Flow Overview.

For further details on how to maintain the order book correctly and how to recover from out-of-sequence please refer to the document Enhanced Broadcast Solution - Interface Specification published on the Eurex website (e.g. General Guidelines for Order Book Information or Appendix - How to Use)

Figure 14 EBS Receiver – UDP Messages Out of Sequence Processing Scenarios

Applications that join a multicast group and receive multicast packets should be able to tolerate duplication and out of sequence packets. The Eurex EBS uses transmission of duplicate data on separate physical channels (multicast group | UDP port). By using separate physical network infrastructure configurations, the redundancy is used to mitigate loss of multicast communications. The receiver application should capable listening to both channels , arbitrate the data between the channels and then produce a single data set reconciled from both data channels.

2.3 FIX Protocol Basic User's Guide Implementation Notes

The principal of content-awareness not only allows FAST to achieve high levels of data compression but also allows data be compressed with extremely low processing overhead and latency. Data can be encoded and decoded very rapidly causing negligible levels of processing latency. Additionally, compression decreases the amount of data that has to be transferred, resulting in decreased transfer latency.

In general, the reduction in transfer latency greatly outweighs the increase in processing latency and produces a significant improvement in overall latency. FAST provides performance gains that are markedly better than what can be achieved with an uncompressed feed or compression using other methods. The impact can be defined as:

Increase in processing latency minus decrease in transfer latency

The diagram below illustrates the layers of the FAST Protocol in a simple configuration from a sending application to a receiving application.

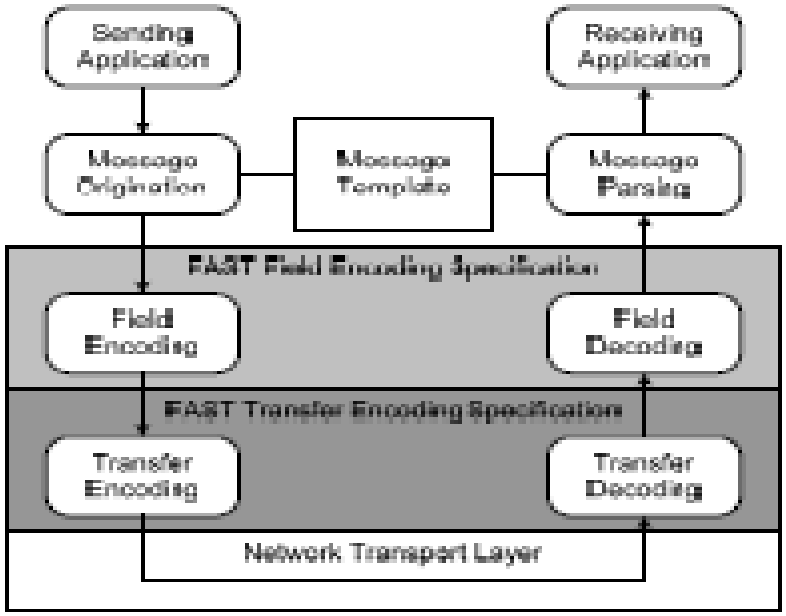


Figure 1 - FAST Protocol Components

Templates

The message template is a fundamental component of the FAST Protocol message exchange that was established and developed as part of the FIX Implicit Tagging concept. A template uniquely identifies an ordered collection of fields and optionally includes notation corresponding to the field encoding and transfer encoding rules. A template can be represented in machine readable forms as in compact notation or xml, or simply as a bilateral agreement between the parties and supported in text documentation as many of the exchange data feeds provide today. Templates will be discussed in greater detail later in this document.

Figure 15 FIX FAST Protocol Sender/Receiver Communications Architecture

2.4 FIX Adapted for Streaming – FAST Protocol Technical Overview

FAST℠ Protocol Technical Overview

FAST℠ – a technical solution

Broadly the design of the final FAST℠ Protocol is a series of layers as shown in the diagram below:

Application messages are the market data that need to be transmitted. Market data messages are handed into a Field Encoding layer that removes redundant data before they are handed off to the Transfer Encoding layer. The Transfer Encoding layer applies a number of techniques to compress the data into a minimum number of bytes. Once compressed the data is transmitted either using UDP in a multicast environment or TCP in a peer to peer session based protocol. Session Control governs such aspects of the communication as initiating and terminating FAST sessions, and error reporting.

The Field Encoding layer removes redundant information by assigning multiple messages to the same logical frame, called a data stream in FAST℠, and using a template that describes the data format and optimisations. This allows FAST℠ to exploit similarities between consecutive messages to remove the need to transmit all the data content of a message.

The relationships are "Copy" when the value is a copy of the data transmitted in the previous message, "Increment" when you simply increment the value from the previous message, "Delta" when only the difference is sent, "Default where a default value is described in the template and the value is only sent if different" and "None" when there is no relationship and all the data on each message is sent. Message Templates describe the layout of each message type and define which field encoding technique is to be used on each field. A final type of encoding supported by the message template is "Constant" and this represents a field that always takes a particular value. These constant values are always simply added back into each message on decoding.

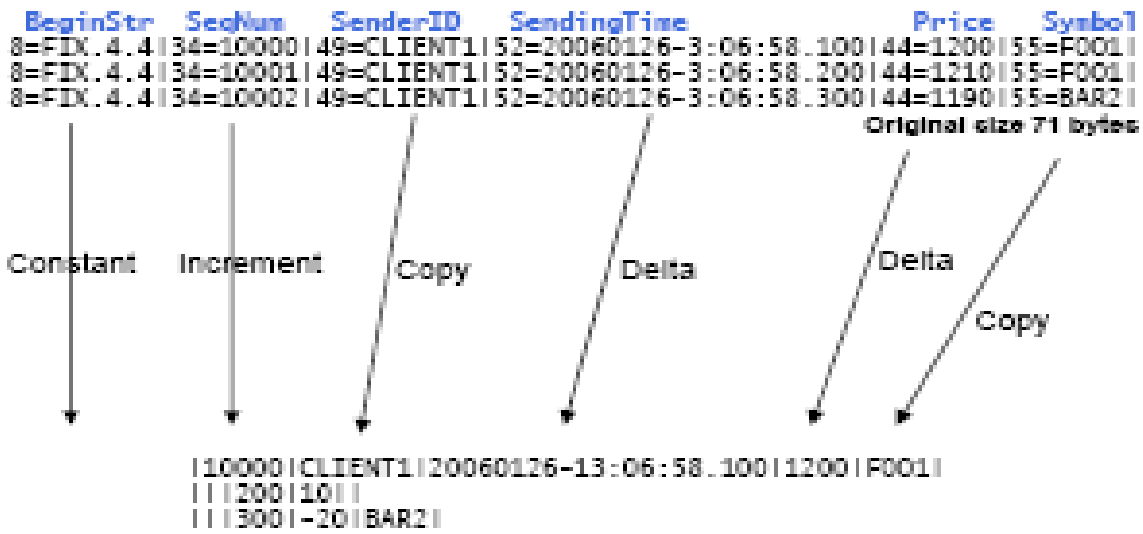
This message template is described in an XML template

```
<template name="ExampleOrder">
   <messageRef name="NewOrderSingle"/>
   <string name="BeginStr"> <constant value="FIX.4.4"/> </string>
   <u32 name="SeqNum"> <increment/> </u32>
   <string name="SenderID"> <copy/> </string>
   <string name="SendingTime"> <delta/> </string>
   <decimal name="Price"> <delta/> </decimal>
   <string name="Symbol"> <copy/> </string>
</template>
```

The effect of coding a message according to the rules described in the message template can be seen on the extract of three messages. The diagram below illustrate this

Figure 16 FAST Protocol Messaging Layers

FAST℠ Protocol Technical Overview



The Transfer Encoding layer compresses the data into the minimum physical space required on the wire. This compression is achieved utilizing a number of techniques.

In traditional FIX messages each field takes the form "Tag=Value<SOH>" where the tag is a number representing which field is being transmitted and the value is the actual data content. The ascii <SOH> character is used as a byte delimiter to terminate the field.

The first area of redundancy is the opening of every field with "Tag=". These are mostly 2, 3 or 4 digit tags so they represent 3, 4 or 5 bytes of data that can be eliminated by exchanging a template that describes the message structure. This technique is known as implicit tagging as the tags become implicit in the data.

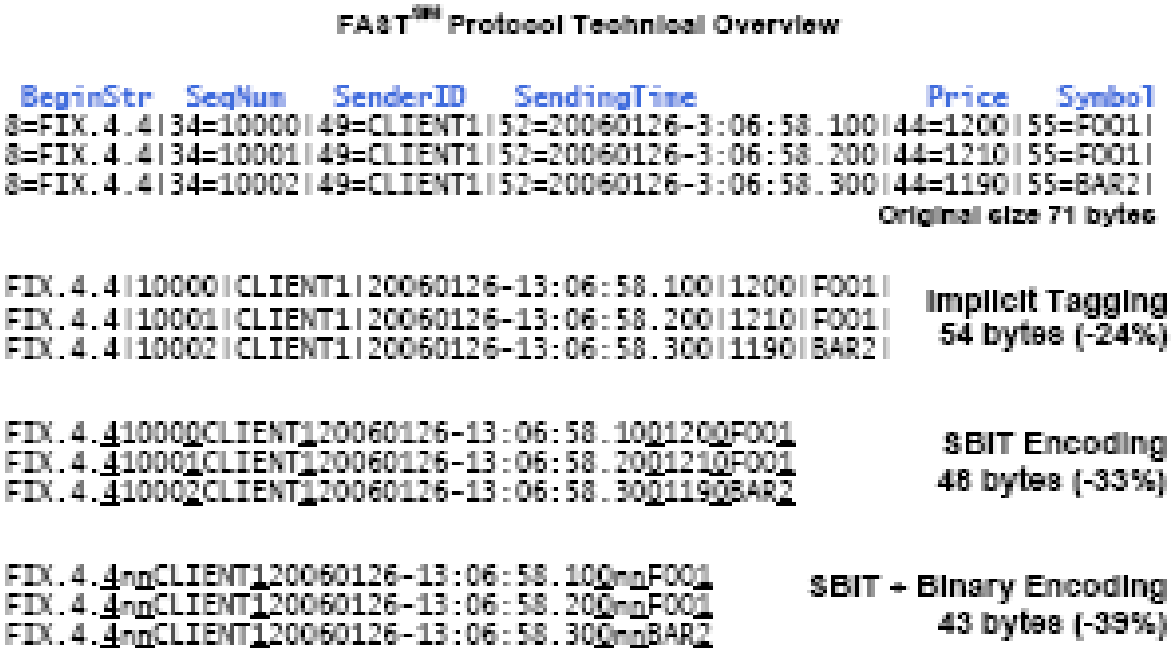
At the data field level a stop bit is used instead of FIX's traditional <SOH> separator byte. Thus 7 bits of each byte are used to transmit data and the eighth bit is used to indicate if the end of the field has been reached. For market data where the average field length is approximately 2 bytes this saves approximate 33% of the bandwidth.

At the message level the first bytes of each message are a presence map indicating which fields are present in this particular message. If a field is shown as being transmitted in the presence map the transmitted field values then follow the map. If no fields are shown as present, all fields are derived from previous messages and the next byte is the start of the presence map for the next message.

Another technique, called binary encoding is used on number, this basically renders the number into binary across the 7 data bits in each byte. Thus a number less than 2^7-1, (127) will only occupy one byte, a number between 2^7 and 2^7^2 – 1 (16,383), will occupy two bytes etc. This saves space on many numbers, for example 16 represented as ASCII requires two bytes, binary encoded it only requires one.

The diagram below illustrates the effect of transfer encoding

Figure 17 FAST Protocol Messaging Format



The reference code

FPL had experience of launching a protocol before and felt that one of the problems with any protocol specification was that it was open to interpretation and that any ambiguity is both difficult to spot and interpreted in at least two different ways. Indeed a large part of the maturing process of FPL has been the clarification of those ambiguities. To avoid this pitfall in the FAST℠ Protocol FPL decided to publish a reference implementation. The prototype has been developed in C and it was anticipated that most implementations would be in C due to its speed.

Summary

FPL has launched a new protocol called FAST℠. The FAST℠ Protocol is designed to stream financial market data using as little bandwidth and introducing as little processing latency as possible.

References

This whitepaper is largely based on material presented at a series of FAST briefings by FIX Protocol Limited. The original material can be found on the http://www.fixprotocol.org/fast

Figure 18 FAST Protocol Message Format Cont'd

2.5 Session Control Protocol (SCP) Message Processing

The SCP 1.0/1.1 specifications provide a setup of predefined messages used to initiate and control the exchange of FAST encoded messages. The SCP is layer of software in the EBS Receiving Application, this software receives the UDP/IP packets received from the Enhanced Broadcast System (EBS) [Eurex Exchange]. The SCP is used transmit encoded messages to the EBS Receiving application. The information received conforms with the FAST protocol version 1.0/1.1

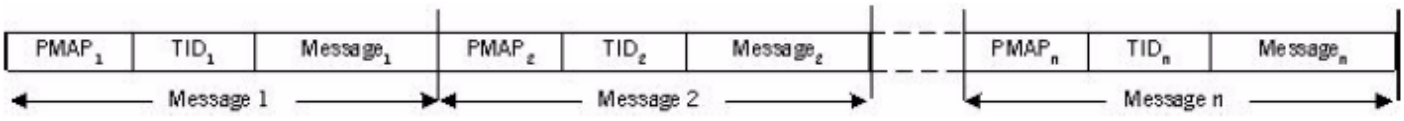
SCP is a communications layer of software that needs to reside above the received UDP messages. Here the EBS receiving program will first call the FAST decoder. And then Examine the messages based on the message ID (TID). And perform the desired operations.

The SCP is software that needs to be written and/or obtained as basic processing unit Within the Receiving application. Either we have to write this layer of software or obtain This from either Open Source Software and/or Vendor..

The SCP decodes and formats the inbound UDP messages into FAST Protocol messages. This can include control functions like Hello (sync between sending and receiver), Reset (Receiver to stop processing and change state to initial condition), Templates or other functions identified in the FIX Protocol Functionality Matrix.

Structure of Messages

The Enhanced Broadcast Solution disseminates data in UDP datagrams in a network byte order also known as big endian byte order. All messages disseminated by the EBS interface will conform to the FAST standard 1.1.



Enhanced Broadcast Solution message structure

Every UDP datagram sent through the Enhanced Broadcast Solution will have the above structure.

- Presence Map (PMAP)

The Presence map is a bit combination indicating the presence or absence of a field in the message body. The allocation of bit for a field in the presence map is governed by the FAST field encoding rules.

- Template Identifier (TID)

The Template Identifier is a uint32 field to inform the template to be used for decoding the message. The following table lists the message types and their corresponding template identifiers:

Figure 19 Structure of Eurex EBS Messages

The following table lists the message type and their corresponding template identifiers (TIDS).

TID	Message Type
1	Version Information Message
2	Beacon Message
3	Product Reference Information Message
4	Single Leg Reference Information Message
5	Strategy Reference Information Message
6	Order Book Snapshot Information Message
7	Order Book Delta/Incremental Information Message
8	Request Information Message
9	Trade Information Message
10	Product Status Message
11	Block Auction Status Message
12	Additional Contract Information
120	FAST Reset Message (SCP 1.0)
128	Start of Service Message
129	End of Service Message
130	Start of Product Reference Message
131	End of Product Reference Message
132	Start of Single Leg Reference Message
133	End of Single Leg Reference Message
134	Start of Strategy Reference Message
135	End of Strategy Reference Message

Figure 20 Summary of EBS TID Message Types

Note: The UDP Protocol adds at least a 28 byte header to every packet (20 byte IP header plus byte UDP protocol header).

Messages are logically divided into:

- **Service Data Messages** – These messages are sent to synchronize or to indicate status of service. Service Messages do not contain market information.
- **Market Data Messages** – Contain Market Information , UDP Datagram will be a complete message. No dependency across datagrams is present.

Refer to Eurex-Enhanced Broadcast Solution Interface Specification – (Final Version) EUREX® Release 10.0 Section 5.0 Structure of Messages

2.6 FIX Protocol - Recommended Practices for Book Management

The FIX Market Data Working Group has defined a set of messages and protocols with the objective of creating a standard that allows vendors and clients to connect quickly and reliably across a range of markets.
The Working Group recognizes two general categories of markets. The first category is the style of market where executable orders and quotes are maintained in a central limit order book and is typical of the automated equity markets. The other style is a Quote market where quotes are streamed from many dealers and cannot be consolidated. The latter is typical of the bond and currency markets.
Please note that all materials included herein are recommended guidelines with respect to implementing a book management paradigm, not hard and fast rules. The specification provides flexibility in applying different techniques to manage and organize book-related market data.
The use of a Central Orderbook is typical of electronic equity markets where executable orders are placed into a book, sorted by price, then time. A client can expect to execute at the price of the best order in the book or better. The markets are usually anonymous, where the displayed price is firm and available to all parties.

Precision and Depth

A Central Orderbook is characterized by two fundamental concepts; ***Precision*** and ***Depth***. Precision refers to the level of detail at which a book is kept and may be summarized by price level or comprised of individual orders. These are referred to as ***Price-Depth (Aggregate Order Book)*** and ***Order-Depth (Non-aggregated Order Book Detail)*** respectively.
Price-Depth is used to refer to the case where the aggregated quantity is provided for each side of each price level.
Order-Depth is used to refer to the case where every order is described and the client can recreate a copy of the orderbook. There is limited information about each order. Details such as Identity, Account and FreeText are not available, except in a hit and take market where identity of the opposite party may be divulged.
The second aspect of book management is Depth which refers to the number of entries that will be kept in a book. Book Depth may range from ***Top-Of-Book*** which represents only the best price level to Full Book Depth which represents all orders currently open in the market.
Note that different markets refer to the views with different names. The working group has selected a name that is generally acceptable so that the different views are not confused. A common name for the aggregate view is Price-Depth while the common name for the orderbook Detail is Order-Depth. Alternate names for Top-Of-Book are Top Of File and Best Bid Offer or BBO.
The Working Group has defined a standard for these views that can be used across multiple markets. Given the fundamental characteristics of Aggregation and Depth, the following permutations are possible and are recommended as the most common book management practices:

- Top-Of-Book
- Partial Price-Depth
- Partial Order-Depth
- Full Price-Depth
- Full Order-Depth

Trades are sent as a separate instructions within a FIX Market Data Message (MDEntryType = “Trade”). Trades are only used to update the Ticker, or any display where Last, High, Low, Open, Close, Volume, and Cumulative Volume are displayed. **The Trade instruction is not used to update the view of the orderbook.** A trade may not have executed fully against the orderbook depending upon the priority rules of the market. When a trade executes against the book then a Change or Delete instruction is used to update the orderbook.

A FIFO market could require that the client use the Trade Instruction to update the Order Book so that corresponding Orderbook changes do not need to be sent; however, this is not the recommended practice. Additionally, when privately negotiated trades (Trade Capture Reports) are also published as market data, using trade instructions to update Order Book data is again not recommended.

2.6.1 FIX Message Structures

FIX provides two message formats that will be referred to within this document.
The Market Data Incremental Refresh message is used to apply instructions to a book in an incremental manner. This message is best suited for the maintenance of price-depth and order-depth books but may also be used for top-of-book.
The Market Data Snapshot Full Refresh is used to provide a full snapshot of the current book. For this reason it is appropriate for top-of-book updates in which both sides of the book are sent. The message may also be used in a recovery situation where a number of order books need to be re-established.

2.6.2 Top-Of-Book

The Vendor may choose to provide the current best bid and offer in the market. Trade details and Instrument Status are provided with separate Market Data Instructions.

Top-Of-Book (Best Bid Offer)

Bid	Ask
100 @ $2.90	400 @ $3.10

The vendor may distribute other information beside Top-Of-Book

Trade Information

Last	Qty	Open	High	Low	Chg	Tick	CumQty
$2.95	2000	$3.10	$3.10	$2.90	- 0.15	+	10,500,000

Trading Session Status

Status
Open

The Vendor maintains the Top-Of-Book view with the OVERLAY instruction.**Overlay**, to update the quantity and Price The incremental instruction approach assumes the use of the Market Data Incremental Refresh message. The Bid and Ask Sides are updated independently with separate instructions. Many existing BBO (Best Bid Offer) feeds have fixed formats that provide both the bid and offer sides in all updates. The practice of sending separate instructions can provide efficiencies by allowing only the bid or ask to be sent, based on which side has changed, rather than both sides. Top-Of-Book only has one price level so there is no need to use multiple instructions when the best price changes. An action of *overlay* is used to indicate a new price or a change in quantity at an existing price. When the best price changes the vendor does not send a delete of one price level followed by a New for the next price level.

A Trade instruction does not update the Top-Of-Book. A separate *overlay* instruction is sent to update the book as a result of the trade. The *overlay* instruction does not necessarily follow the trade. The Trade instruction and the instruction to update the book may be separated by a period of time and the change could be sent before or after the trade**.**

2.6.3 Trades

The Trade instruction that is sent within the Market Data message is used to provide information about two sided trades. One party could enter an order that executes against several counterparties over multiple prices. Some Vendors will send a consolidated trade for the total quantity at each price level. Other vendors will send one trade instruction for each counterparty pair. Either method is acceptable. One concept that is prohibited is to show the Bid and Ask sides as separate trades as this results in double counting. Individual Fill execution reports are sent to each party in the trade, but not through the Market Data feed.

A Trade can update the trade statistics (Last, Volume, Open, High, Low, Chg, Direction) for both the local exchange and the national market. The trade may have been an off-market trade in which case it does not update any of the trade statistics. There are two approaches commonly used to update the trade statistics. Some vendors send Open, High, Low and Last with each trade and some send them only if they are changed. Proper formatting of Trades, Volume, Open, High, and Low will be discussed further below.

2.6.3.1 A Basic Instruction of Top-of_Book

A FIX message sent to update the top of book will update one side only. The tags normally sent with Top Of Book are:

```
SecurityDesc/SecurityID/Symbol=security identifier  (note that the there
are several ways to identify the security)MDBookType=Top-Of-Book,
MDUpdateAction=Overlay
MDEntryType=Bid/Ask
MDEntrySize=quantity available at bid/ask MDEntryPx=price.
MDEntryPx=price
```

2.6.3.2 Indexing the Entry

Top-of-Book entries can be referenced for maintenance purposes using several techniques. The first is to create a composite index which consists of the Security Identifier (see above), MDEntryType (bid/ask) and

MDEntryPx. This set of fields acts as a composite key that allows the entry (bid or ask) to be accessed and subsequently deleted.
The other approach is to use MDEntryID and assign a unique identifier to act as a key. Every active entry carries a unique id which can be used for reference purposes. When the entry is no longer active, the ID can be re-used and assigned to the next active entry.
Both approaches have their advantages. The 'composite key' approach does not require that an external identifier to be carried on the initial Add or Delete instruction nor does it require the sending system to maintain a matrix of external identifiers. The MDEntryID approach provides a single field for referencing an entry which may improve system performance.
Note: that use of MDEntryID does not mean that MDEntryType, MDEntryPx, or MDPriceLevel can be dropped on a Delete instruction. These fields are necessary to ensure the integrity of the book when applying the instruction.

2.6.3.2.1.1 Price-Depth

The Vendor may choose to provide the aggregate quantity available at each price level. This view can be visualized as a number of rows in a table for each of the bid and offer sides. On each side there are a number of rows showing the quantity available at a number of price levels.
Example:

Bid	Ask
100 @ $2.90	400 @ $3.10
65 @ $2.85	5000 @ $3.20
450 @ $2.80	200 @ $3.30

The Price-Depth view is an expansion of the Top-Of-Book view. The trade and status windows are independent and can be shown beside the Price-Depth view in the same way as they are shown beside the Top-Of-Book view.A Price-Depth Book is sequenced by Price, descending for bid and ascending for ask. Another common representation is to show the Market in a Vertical fashion.

Bid	Ask
	200 @ $3.30
	5000 @ $3.20
	400 @ $3.10
100 @ $2.90	
65 @	

$2.85	
450 @ $2.80	

The Vendor must maintain the Price-Depth view with the following entries

- **New**, to create/insert a new price level
- **Delete**, to remove a price level
- **Change**, to update quantity at a price level
-

The Bid and Ask Sides are updated with separate instructions. Most markets do not show the Price-Depth view for the entire book. The Price-Depth view is limited to a number of levels, typically three to five. Equity Markets that trade in pennies tend to show at least ten price levels. The Recipient must know how many price levels are being supplied by the vendor and must delete the bottom row when the number of rows is exceeded. The vendor will not send a delete instruction when the number of rows is exceeded. The vendor will send the bottom row again when a higher level row is deleted. If a trade occurs in the market then the vendor will send Delete or Change instructions to update the Price-Depth. The trade instruction itself is not used to update the order book as the order may not have executed against the book.

2.6.3.3 Use of the Tags PriceLevel and MDEntryID

2.6.3.3.1 MDPriceLevel

The addition of the tag PriceLevel is used for markets that are not Price priority or in any situation where there can be confusion on the sequencing of the rows. PriceLevel is also beneficial for recovery after a gap in the data feed. If PriceLevel is present then the client can tell if a new price is the top price level and can delete higher price levels. Exhibit 1 shows how PriceLevel can be useful in detecting a book made inaccurate due to dropped message.

2.6.3.3.2 Indexing the Entry

Price-Depth entries can be referenced for maintenance purposes using several techniques. The first is to create a composite index which consists of the Security Identifier (see above), MDEntryType (bid/ask) and MDEntryPx. This set of fields acts as a composite key that allows the entry to be accessed and subsequently deleted.

The other approach is to use MDEntryID and assign a unique identifier to act as a key. Every active entry carries a unique ID which can be used for reference purposes. When the entry is no longer active, the ID can be re-used and assigned to the next active entry.

Both approaches have their advantages. The ‘composite key’ approach does not require that an external identifier to be carried on the initial Add or Delete instruction nor does it require the sending system to maintain a matrix of external identifiers. The MDEntryID approach provides a single field for referencing an entry which may improve system performance.

Note that use of MDEntryID does not mean that MDEntryType, MDEntryPx, or MDPriceLevel can be dropped on an Update or Delete instruction. These fields are necessary to ensure the integrity of the book when applying the instruction.

Price Level is a technique for conveying the position to which a given incremental instruction should be applied. The purpose is to reinforce the integrity of the instruction being received in relation to the current state of the book.

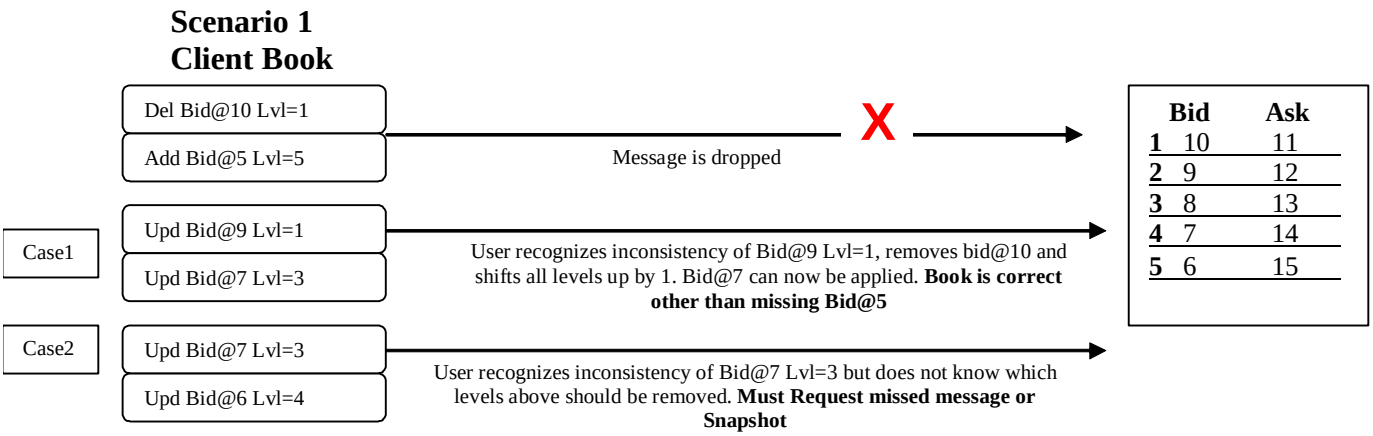


Figure 21 Price Level Client Order Book Adjustments

Note that with both the Price and Order Depth views there is a possibility that some orders are available for unconditional execution while others are not. Examples include various quantity conditions as "minimum fill quantity", "minimum execution increment", and "AON". Currently, this should be supported via the QuoteCondition field, tag 276 and enumerations that are yet to be defined.

2.6.4 Order-Depth

The vendor may provide limited information about each individual orders at each price level. Information is sent about each order at the price level rather that stating that a total quantity is available at each price level. The Client can summarize the information to create the Price-Depth view if desired.

It is possible in some markets to assume that orders can be sorted by price then time, however many markets do not prioritize all orders by time. The Orderbook can be viewed as a table with multiple orders at each price level. Each order has a unique Entry ID. When adding orders to the book the Add instruction contains the EntryPositionNumber within the PriceLevel at which the order is to be inserted. Change and Delete instructions will reference the order by its MDEntryID. The MDEntryID is not to be confused the ClOrdID which is the confidential order reference assigned by the client when the order was entered. MDEntryID is an identifier assigned by the vendor to track this order in the market data feed and may be unique across the entire feed. FIX requires that the MDEntryID be unique among all other active entries,

Note that MDEntryID is constant for an entry until that entry is removed from the book due either to deletion or dropping out of the stated market depth.

Order-Depth

BID

9740	ID=213, Qty=2
9730	ID=223, Qty=4; ID=227, Qty=1
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

The vendor must send the following instructions to maintain the Orderbook:

- **New**, to insert a new order
- **Delete**, to delete an order
- **Change**, to alter the details of an order

New is used to insert an order, identified by MDEntryID, into a position within a price level. The new entry is sequenced by Price Level Number within the book and then Entry Position Number within that PriceLevel.

Change is used to change the quantity or other information related to an order. Change should not be used to alter the PriceLevel or EntryPositionNumber of an entry as this is the responsibility of the client maintaining the book. The trading function of Cancel & Replace that changes the price of an order would result in a delete of the old order and an insert of the updated order into the new price. 'Change' should not be used when the price of the order is changing due to a cancel/replace or when the order is being re-ranked for any reason. Any action which causes the order to move to a different price level or position should be accomplished through use of a 'delete' followed by a 'new'.

Delete and Change Instructions use MDEntryID as the key to reference the order. They can optionally include the PriceLevel and EntryPositionNumber of the order. This can aid client systems when the market has deep orderbooks as it narrows the search for the subject order. It also provides a constant check regarding the integrity of the book.
The Recipient must know how many price levels are being supplied by the vendor and must delete the bottom row when the number of rows is exceeded. The vendor will not send a delete instruction when the number of rows is exceeded. The vendor will send the bottom row again when a higher level row is deleted.

Delete-Thru is an instruction that can be used to reduce the number of instructions necessary to maintain a book and further improve efficiency. The "Delete Thru" instruction causes the book to be deleted from the top down to the specified Price and PriceLevel. The empty price levels are then filled with Add instructions.

Indexing the Entry
Order-depth books should use MDEntryID as the key to reference the order for Delete and Change Instructions. They can optionally include the PriceLevel and EntryPositionNumber of the order. This can aid client systems when the market has deep orderbooks as it narrows the search for the subject order. It also provides a constant check regarding the integrity of the book. The 'composite key' approach using a Security Identifier, MDEntryType (bid/ask) and MDEntryPx is not sufficiently detailed to reference a specific order in the book.

A Trade instruction is not used to update the book. An execution against the book may not occur in Time priority. Following an execution a trade instruction is sent along with Change/Delete instructions to update the book as a result of the trade.
Summary:
Each order has an MDEntryID that is a unique reference within a feed or order book. The MDEntryID must be unique across the feed when active but can be reused once when no longer in use.
Price is used to sequence the order book unless PriceLevel is provided.
MDEntryID is used to update and delete an order. Price may be included to assist in finding the order within the book.
EntryPositionNumber is used to sequence the orders within a given price level.
The order-level view has the option of showing a limited number of price levels. When adding an order at a new price level it is implied that all orders at the bottom price level are to be removed. If an order is removed that clears an entire price level then the bottom price level must be resent by the vendor.

2.6.5 Trading and Security Status

The status of each instrument is normally sent as part of the data feed. When the state of an instrument is changed, a FIX Trading Session Status or Security Status message is sent with the instrument name and the

new state. The current quotes are not sent again. The state is associated with the instrument, not the Bid or the Offer.

The trading status of a security should be sent on the FIX Security Status message (35=h).

The trading status of a market should be sent on the FIX Trading Session Status message (35=f).

In some markets, there is the concept that the instruments belong to various markets and that the markets open and close at different times. The trading state of an instrument is the lowest state of either its state or the owning market state. When the market opens then all instruments in an open state that are in that market will open as well. If the market closes then all instruments in that market will close.

Some Derivative markets will open/close all instruments in the market at the open and the close. This can be a very large number of messages which can create capacity issues for other instruments that are still trading. Another method is to use wildcards to indicate that all instruments of a particular class have changed state.

2.6.6 Quote Markets

In a Quote Market there are dealers that provide quotes. There is no central market and the concept of a Central Orderbook does not exist. The prices displayed are not necessarily firm or available to all participants. A customer must contact the dealer to trade and the price is often negotiated as a points discount to the published price. Orders are sent directly to the dealer. In some equity OTC markets, the orders can be sent to a central routing hub which then routes the orders to the end dealers.

In the debt markets the dealers provide quotes for different products based upon the popularity of the product. Some markets provide unsolicited quotes while others are solicited and require a call to the dealer to obtain a quote. The dealer may provide discounts to the quoted rate depending upon the rating of the client or other negotiated arrangements.

The dealer in a quote market will publish its prices to multiple vendors. The vendors seek to pass the dealer quotes to the client as a montage of prices showing the price and quantity available from each dealer. Some Vendors will take consolidated feeds from other vendors and further consolidate the quotes into higher level montages such as adding markets from multiple countries.

A standard to cover this style of market has the same requirements as the Price-Depth view on the orderbook concept. The vendor will pass on quotes that have size and price however there can be many rows that have the same price, representing the different dealers. An argument can be made that it is also like an orderbook view with multiple orders at each price level. It is considered to be closer to a Price-Depth view than an Order Depth view since the number of quotes in the market is static with the prices constantly changing rather than the number of orders changing.

The Vendor must issue instructions to maintain the view of each table:

- **New**, to create a new price level
- **Delete**, to remove a price level
- **Change**, to update quantity at a price level

The Bid and Ask Sides are updated independently. A message may contain updates to many books and can update the bid and ask sides in the same message.

Each Dealer will be represented as its own price-level row and there can be many at the same price level.Each price-level row must be assigned a unique MDEntryID, price-level, a price and a price-position along with the quantity available at that position. Change and Delete must then reference MDEntryID and Dealer reference to update the book. With this approach it is possible to show multiple rows at the same price, show the rows in any sequence and minimize errors.

2.6.7 Optional Tags

2.6.7.1 Sequencing the Book

There are a number of cases where you cannot use price to sort the rows. Some orders can sit in the book with no price, such as Market orders prior to the open or Cabinet orders. A Market Order is generally shown with a null price at the top of the book. Cabinet orders are generally placed at the bottom of the book with an undefined price. Some debt markets show price inverse to the yield. The best bid is that order with the largest price, resulting in the priorities being reversed. Others quote relative to a standard rate which results in zero and negative prices.Prices can be consolidated from several markets that are at the same price.

If any of these situations lead to confusion, the vendor should use the optional tag PriceLevel in the Add instruction to specify which row in the table this new price level is to be inserted. When this convention is used, the row numbers start at one as the best price level.

Example:

PriceLevel	Bid	Ask	PriceLevel
1	100 @ $2.90	400 @ $3.10	1
2	65 @ $2.85	5000 @ $3.20	2
3	450 @ $2.80	200 @ $3.30	3

If a SELL order is entered onto the book at 3.15 then it would be Added as PriceLevel 2. Changes and deletes will refer to the price of 3.15.

Market Orders with no price could be shown at the top of the book by inserting PriceLevel 1 with a null price.

Multiple Markets or Multiple Dealers at the same price must use the market ID or Dealer ID to provide a unique key for each row.

Book Type

Many trading systems provide multiple views of an orderbook at the same time. The Market data update messages must specify which Book Type they are to be applied. In the event that the vendor is providing multiple views in parallel then the optional tag MDBookType should be used to indicate which view the instruction is to be applied.

Tags Referenced in the Document

The emphasis of this document is to provide a secure and reliable protocol for orderbook maintenance. Depending upon the application the Vendor may choose not to implement these recommendations in favor on increased throughput. A number of new tags are proposed:

- **MDBookType** (tag is 1021): Used to allow multiple views over a common network session. Values are **Top-Of-Book**, Price-Depth, Order-Depth.

- **MDPriceLevel** (tag is 1023): Used to specify sequencing of prices when a market is not Price priority.

- **MDPriceLevel** contains the row number at which a price is to be displayed, where the best price in the market is level 1. Also used to resynchronize a book if there are missing blocks in the data feed.

- **TradeCondition** (tag 277): Specifies the conditions which qualify a trade that is being reported as part of the market data stream. Common values are Opening (E), Cancel (O), Exchange Last (U).

- To support Reg-NMS, quotes must be marked as manual/Slow if they are manual, ie, not executed automatically.

Manual quotes are indicated with tag 276 QuoteCondition= L (Manual/slow quote)
Trades resulting from manual/slow quotes are also indicated as follows:
277 TradeCondition Valid values:
Y = Trade resulting from manual/slow quote
Z = Trade resulting from intermarket sweep

MDFeedType (tag 1022): Describes a class of service for given data feed

MDEntryPositionNo (tag 290): Used to identify the position of an order at a given PriceLevel in an order depth book.

MDEntryID (tag 278): Used to uniquely identify each entry in a book for more efficient update and deletion of that entry. Note that MDEntryID may also used as an index price levels for subsequent updates and deletions and may improve the performance of client systems under certain conditions. Each side is assigned a unique integer that can be used as an index to that row for future updates. The index should start at one and re-use freed entries. The maximum number should be the maximum number of bid/ask rows.
It may be useful to assign an MDEntryID series per security (order book) to avoid problems when processes are partitioned. It may not be practical for MDEntryID to be unique across the feed.

Example:
The first BID disseminated for stock ZZZ may have an MDEntryID of 333 assigned. From then on, all further updates to the ZZZ BID should include MDEntryID of 333. This will allow a client system to create a reference table and index directly into row to update or delete that row. If the bid is deleted then MDEntryID 333 is now free and can be reused for another row.

2.6.8 Examples

2.6.8.1 Initial Market

This example shows how the different views are updated for the same events. In this example we are using a Futures Market where the price is very large and the quantity is expressed in contracts and tend to be small.The following representation shows Top-Of-Book, Price-Depth and Order-Depth. In the Top-Of-Book and Price-Depth the cells show total quantity and price. On the bid-side you see qty/price and on the Ask side you see price/qty. In the Order-Depth you see the orders at each price level. For each order you see the reference ID and the quantity. The vendor may make additional information available but this is the minimum required.

Top-Of-Book – BBO

BID	ASK
2 / 9740	9760 / 5

Price-Depth

BID	ASK
2 / 9740	9760 / 5
5 / 9730	9770 / 3
1 / 9720	9780 / 5
	9790 / 7

Order-Depth

BID

9740	ID=213, Qty=2
9730	ID=223, Qty=4; ID=227, Qty=1
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

2.6.8.2 An Order is added to the book

An order is added to the orderbook creating a new best price level. It will cause all updates to occur to all three orderbook views.

Buy 3 @ 9750

Update Top-Of-Book

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
271	MDEntrySize	3	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price-Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	3	Quantity
270	MDEntryPx	9750	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	3	Quantity
270	MDEntryPx	9750	Price
290	MDEntryPositionNo	1	Position at this price level
278	MDEntryID	274	Order Reference

Result

Top-Of-Book

BID	ASK
3 / 9750	9760 / 5

Price-Depth

BID	ASK
3 / 9750	9760 / 5
2 / 9740	9770 / 3
5 / 9730	9780 / 5
1 / 9720	9790 / 7

Order-Depth

BID

9750	ID=274, Qty=3
9740	ID=213, Qty=2
9730	ID=223, Qty=4; ID=227, Qty=1
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

2.6.9 An Order is added Below Top Of Book

An order is added to a lower price level. The Top-of-book is not affected. The Price-Depth will update the size being displayed.

BUY 9 @ 9730
No change to Top-Of-Book

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	3	Price Level
279	MDUpdateAction	1	Change
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	14	Quantity
270	MDEntryPx	9730	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	3	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	9	Quantity
270	MDEntryPx	9730	Price
290	MDEntryPositionNo	3	Position at this price level
278	MDEntryID	275	Order Reference

Result

Top-Of-Book BBO

BID	ASK
3 / 9750	9760 / 5

Price-Depth

BID	ASK
3 / 9750	9760 / 5
2 / 9740	9770 / 3
14 / 9730	9780 / 5
1 / 9720	9790 / 7

Order-Depth

BID

9750	ID=274, Qty=3
9740	ID=213, Qty=2
9730	ID=223, Qty=4; ID=227, Qty=1; ID=275, Qty=9
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

2.6.10 Change an order

An order is changed to increase its size. This causes the order to be updated in the Order-Depth and changes to the size in the Top-Of-Book and Price-Depth.

EntryID 274 changes Qty from 3 to 5.

Update Top-Of-Book

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
271	MDEntrySize	5	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	1	Change
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	5	Quantity
270	MDEntryPx	9750	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	1	Change
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	5	Quantity
270	MDEntryPx	9750	Price,
290	MDEntryPositionNo	1	Position at this price level
278	MDEntryID	274	Order Reference

Result

Top-Of-Book, BBO

BID	ASK
5 / 9750	9760 / 5

Price-Depth

BID	ASK
5 / 9750	9760 / 5
2 / 9740	9770 / 3
14 / 9730	9780 / 5
1 / 9720	9790 / 7

Order-Depth

BID

9750	ID=274, Qty=5
9740	ID=213, Qty=2
9730	ID=223, Qty=4; ID=227, Qty=1; ID=275, Qty=9
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

Note that the trading function Cancel & Replace where an order changes price will cause two market data instructions in each of the Aggregate and Detail views. One instruction is required to delete the order from one price level and another to add a new order to the new price level.

2.6.10.1 Resequencing Entries at a PriceLevel

In this example the order remained at position one. If an order was changed such that it lost priority then a change instruction is sent with the new position. An Insert causes the new order to be inserted at the position stated in the instruction. A change will cause the order to be removed from the book then inserted at the position specified in the instruction. You must remove the order from the book before determining the new position for the order.

Delete Order which deletes the Price level. An order is deleted at a lower price level and it is the only order at that price level. It causes the Price-Depth to have the entire price-level removed.

Delete Order with EntryID=213

No change to Top-Of-Book

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	2	Price Level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9740	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	2	Price Level
290	MDEntryPositionNo	1	Position at this price level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9740	Price,
278	MDEntryID	213	Order Reference

Result

Top-Of-Book, BBO

BID	ASK
5 / 9750	9760 / 5

Price-Depth

BID	ASK
5 / 9750	9760 / 5
14 / 9730	9770 / 3
1 / 9720	9780 / 5
	9790 / 7

Order-Depth

BID

9750	ID=274, Qty=5
9730	ID=223, Qty=4; ID=227, Qty=1; ID=275, Qty=9
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

2.6.11 A Trade occurs over multiple price levels.

An order is entered to SELL 12 contracts at 9730.
It traded 5 @ 9750 then 7 @ 9730.

At 9730 there are three counterparties for quantities 4, 1 and 9.

The trade instructions do NOT update the view of the book. The trade instructions are followed by change and delete instructions to update the book.

Trade Instructions

Tag number	Tag name	Value	Description
279	MDUpdateAction	0	New
269	MDEntryType	2	Trade
271	MDEntrySize	5	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument
277	TradeCondition	Last, Opening, Block, Average Price	Trade condition codes, Indicators

Tag number	Tag name	Value	Description
279	MDUpdateAction	0	New
269	MDEntryType	2	Trade
271	MDEntrySize	7	Quantity
270	MDEntryPx	9730	Price
107	SecurityDesc	GEZ5	Instrument

Update Top-Of-Book

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	0	Bid
271	MDEntrySize	7	Quantity
270	MDEntryPx	9730	Price
107	SecurityDesc	GEZ5	Instrument

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9750	Price

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	1	Change
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	7	Quantity
270	MDEntryPx	9730	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
290	MDEntryPositionNo	1	Position at this price level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9750	Price,
278	MDEntryID	274	Order Reference

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
290	MDEntryPositionNo	1	Position at this price level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9730	Price
278	MDEntryID	223	Order Reference

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
290	MDEntryPositionNo	1	Position at this price level
279	MDUpdateAction	2	Delete
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument

270	MDEntryPx	9730	Price
278	MDEntryID	227	Order Reference

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
290	MDEntryPositionNo	1	Position at this price level
279	MDUpdateAction	1	Change
269	MDEntryType	0	Bid
107	SecurityDesc	GEZ5	Instrument
270	MDEntryPx	9730	Price
271	MDEntrySize	7	Quantity
278	MDEntryID	275	Order Reference

Result

Top-Of-Book, BBO

BID	ASK
7 / 9730	9760 / 5

Price-Depth

BID	ASK
7 / 9730	9760 / 5
1 / 9720	9770 / 3
	9780 / 5
	9790 / 7

Order-Depth

BID

9730	ID=275, Qty=7
9720	ID=973, Qty=1

ASK

9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

2.6.12 A new price causes the bottom price level to be deleted

If the Price-Depth and Order-Depth are only maintained to 5 price levels, then the addition of two more price levels to the example above will cause the bottom row to be deleted.

Sell 8 @ 9750, then Sell 6 @ 9740
Update Top-Of-Book for first order, Sell 8 @ 9750

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
271	MDEntrySize	8	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	8	Quantity
270	MDEntryPx	9750	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Bid
107	SecurityDesc	GEZ5	Instrument, it's the key to the book
271	MDEntrySize	8	Quantity
270	MDEntryPx	9750	Price, it's the key to insert the entry
290	MDEntryPositionNo	1	Position at this price level
278	MDEntryID	296	Order Reference

Result

Top-Of-Book, BBO

BID	ASK
7 / 9730	9750 / 8

Price-Depth

BID	ASK
7 / 9730	9750 / 8
1 / 9720	9760 / 5
	9770 / 3
	9780 / 5
	9790 / 7

Order-Depth

BID

9730	ID=275, Qty=7
9720	ID=973, Qty=1

ASK

9750	ID=296, Qty=8
9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5
9790	ID=233, Qty=4; ID=234, Qty=3

Price-Depth and Order-Depth are now at the maximum number of levels on the Ask side of the book. The addition of another price level will cause the bottom row to be deleted.

Update the Top-Of-Book for second order, 6 @ 9740

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
271	MDEntrySize	6	Quantity
270	MDEntryPx	9740	Price
107	SecurityDesc	GEZ5	Instrument

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	6	Quantity
270	MDEntryPx	9740	Price

Update Order-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	3	Order Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Bid
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	6	Quantity
270	MDEntryPx	9740	Price
290	MDEntryPositionNo	1	Position at this price level

278	MDEntryID	297	Order Reference

Result

Top-Of-Book, BBO

BID	ASK
7 / 9730	9740 / 6

Price-Depth

BID	ASK
7 / 9730	9740 / 6
1 / 9720	9750 / 8
	9760 / 5
	9770 / 3
	9780 / 5

9790 / 7 is deleted by client

Order-Depth

BID

9730	ID=275, Qty=7
9720	ID=973, Qty=1

ASK

9740	ID=297, Qty=6
9750	ID=296, Qty=8
9760	ID=230, Qty=5
9770	ID=231, Qty=3
9780	ID=232, Qty=5

Orders ID=233, Qty=4; ID=234, Qty=3 are deleted by client.

2.6.13 Use of MDEntryID for Top-Of-Book and Price-Depth

A vendor can include a the Tag MDEntryID on the Top-Of-Book and Price-Depth feeds to improve performance of client systems.

Sell 8 @ 9750

Update Top-Of-Book

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
271	MDEntrySize	8	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument
297	MDEntryID	2254	Unique key for this instrument/side

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	8	Quantity
270	MDEntryPx	9750	Price
297	MDEntryID	25244	Unique Key for this instr/price/side

The size changes to 10.

Update Top-Of-Book

Tag number	Tag name	Value	Description
1021	MDBookType	1	Top-Of-Book
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
271	MDEntrySize	10	Quantity
270	MDEntryPx	9750	Price
107	SecurityDesc	GEZ5	Instrument
297	MDEntryID	2254	Unique key for this instrument/side

Update Price-Depth

Tag number	Tag name	Value	Description
1021	MDBookType	2	Price Depth
1023	MDPriceLevel	1	Price Level
279	MDUpdateAction	0	New
269	MDEntryType	1	Ask
107	SecurityDesc	GEZ5	Instrument
271	MDEntrySize	10	Quantity
270	MDEntryPx	9750	Price
297	MDEntryID	25244	Unique Key for this instr/price/side

The Client does not need to find the commodity, price-level then side. The use of MDEntryID allows the client firm to index directly to the line to be changed.

2.6.14 Market Statistics

2.6.14.1 Market Statistics

There are a number of statistics often included in market data feeds which are related to changes in a book but which are not used to update the book. These include the Open High Low, Last trade prices. Also included are Highest Bid/ Lowest Offer and Last Best Bid/Last Best Offer.

2.6.14.2 Last Trade

The Last Trade indicates that the last trade has occurred at a given price level and provides consolidated time and sales information to a client. The Last Trade summarizes trade activity at a price level and can be used as an alternative or complement to individual trades that are generated as a result of a number of partial fills.

FIX Syntax for Last Trade

Tag number	Tag name	Value	Description
279	MDUpdateAction	0	New
269	MDEntryType	2	Trade
271	MDEntrySize	5	Quantity
270	MDEntryPx	9740	Price
107	SecurityDesc	GEZ5	Instrument
277	TradeCondition	U	ExchangeLast

The Opening Price is the first sale price for the trading session (day). The High Trade Price pertains to a trade event that has produced the highest trade price for the current session. Likewise, the Low Trade Price indicates that a trade produced the lowest trade price for a given day. In most countries a trading session equates to a 24 hour period but It might start at a different time to midnight. Some countries cease trading for lunch and have multiple trading sessions within a day but this is not common.

A vendor may choose to provide high and low prices only as a response to the event that produces them, or may also send the high and low prices on every trade.

2.6.14.3 Last Best Quote Price

The Last Best Price indicates that a new best bid or new best ask price has occurred. In certain cases, such as an order submission immediately followed by an order cancellation, the top of the book can be bettered without resulting in a new top of book instruction being sent. This is due to the transitory nature of the event. However, Last Best Price can be used to capture this information and convey it to the market.

FIX Syntax for Last Best Price (Bid or Ask)

Tag number	Tag name	Value	Description
279	MDUpdateAction	0	New
269	MDEntryType	0 or 1	0 = Bid 1 = Ask
271	MDEntrySize	5	Quantity
270	MDEntryPx	9740	Price
107	SecurityDesc	GEZ5	Instrument
276	QuoteCondition	C	Exchange Best

2.6.14.4 High and Low Trade

The High Trade Price pertains to a trade event that has produced the highest trade price for the current session. Likewise, the Low Trade Price indicates that a trade event has produced the lowest trade price for a given session. High and Low Trade Prices are helpful in tracking market trends. They also provide historical information for the current session regarding market behavior. A vendor may choose to provide high and low prices only as a response to the event that produces them, or may also send the high and low prices on every trade.

2.6.14.4.1 FIX Syntax for Session High and Low Trade Price

Tag	Tag name	Value	Description

number			
279	MDUpdateAction	0	New
269	MDEntryType	7 8	7 = Session High Trade 8 = Session Low Trade
270	MDEntryPx	9740	Price
107	SecurityDesc	GEZ5	Instrument
336	TradingSessionID	2	Normal

2.6.14.4.2 Best High Bid and Best Low Ask

The Best High Bid and Best Low Ask prices are used to indicate the highest bid and lowest ask quote prices of the session. These prices are useful in tracking market behavior. A Best High Bid and Best Low Ask may be different from the High Trade Price described above if an arriving order is given a price better than the specified limit order. An order to buy at 9751 may trade at 9750 if a resting sell order is already on the book at 9750. In this case, the Best High Bid Price would be higher than High Trade Price. A better priced order may not trade! A vendor may choose to provide Best High Bid and Best Low Ask prices only as a response to the event that produces them, or may also send the prices on every trade.

2.6.14.4.3 FIX Syntax for Best High Bid and Best Low Ask

Tag number	Tag name	Value	Description
279	MDUpdateAction	0	New
269	MDEntryType	N O	N = Session High Bid O = Session Low Ask
270	MDEntryPx	9751	Price
107	SecurityDesc	GEZ5	Instrument
336	TradingSessionID	2	Normal

2.6.15 Snapshots and Recovery

2.6.15.1 Requesting Snapshots

The Market Data specification provides for a Client application to request a snapshot of the orderbook for selected instruments. The Client can specify the type of information and the number of levels required.The FIX Market Data Request message is used for making either a “snapshot” or an “incremental” request in order to reset the book. The table below shows the basic request for returning a snapshot of the book and the corresponding response.

2.6.15.1.1 Guidelines for Requesting Snapshots and/or Updates

A market data feed may consist of both Market Data Snapshot Full Refresh messages and Market Data Incremental Refresh messages. Previously, the specification required that a feed consist of one message type or the other.The Market Data Request message is used to request a static book snapshot or subscribe to a stream of snapshots and updates.Market Data Snapshot Full Refresh should be used to provide a snapshot of the market when Snapshot is requested using SubscriptionRequestType (263). Use of Market Data Incremental Refresh is being discouraged for this purpose. Market Data Snapshot Full Refresh will be used to provide initial snapshot when Snapshot + Updates are requested using SubscriptionRequestType (263) The Market Data Request scenarios that will be supported are as follows:

Market Data Request - Request Type	**SubscriptionRequestType Tag 263**	**MDUpdateType Tag 265**	**Messages Received by Subscriber**
User requests state of the book and receives one and only one snapshot for each request	0=Snapshot	Not Provided	Full Refresh
User requests state of the book + updates and specifies that only Full Refresh Message is used	1=Snapshot+Updates	0=Full Refresh	Full Refresh only
User requests state of the book + updates and specifies that updates are to be sent using Incremental Refresh Message	1=Snapshot+Updates	1=Incremental Refresh	Full Refresh + Incremental Refresh

2.6.15.2 Snapshot Request Format

Example of a Market Data Snapshot Request:

Tag number	Tag name	Value	Description
Header		V	Snapshot request
263	SubscriptionRequestType	0	0 = Snapshot requested
265	MDUpdateType	1	1 = Full Refresh
266	AggregatedBook	Y	For Aggregate / Order-Depth
264	MarketDepth	3	Number of price levels
262	MDReqID	123	My request ID
146	NoRelatedSym	1	
107	SecurityDesc	IBM	Security name

The tags MarketDepth and Aggregate are used to specify the type of information and depth of information in the request. The Vendor must specify the choices that are available to the Client application.When choices are available then the following settings are used:

Orderbook type	Tag values	Comment
Top-Of-Book	Aggregate=Y MarketDepth=1	
Price-Depth, n levels	Aggregate=Y MarketDepth=n	When ths is the only choice then also add MarketDepth=0
Price-Depth, all levels	Aggregate=Y MarketDepth=0	
Order-Depth, n levels	Aggregate=N MarketDepth=n	When this is the only choice then also add MarketDepth=0
Order-Depth, all levels	Aggregate=N MarketDepth=0	

If the vendor offers 5 level Depth and the client requests 3 levels, the Vendor should provide 3 levels. If the Vendor offers three and the client requests 5 then the vendor should provide 3 levels. A request for MarketDepth=0 will get the deepest view available. The vendor should confirm the outcome when the requested level of depth is not available. Multiple instruments per request can be made. The Client must list the instruments for which the data is requested. Some vendors may allow wildcards to be specified or place limits on the maximum number of instruments that can be listed in the snapshot request.

2.6.15.3 Snapshot response

The response to a Snapshot is a list of Snapshot items. The FIX Market Data Snapshot Full Refresh message will be sent in response. The content of each message is determined by the type of data requested.

2.6.15.4 A Market Data Snapshot for Top-Of-Book

The Top-Of-Book Snapshot contains two items for best bid and best offer. Within each item are the quantity and price. Note that there is no need for price level.The following table is an example of a Top-Of-Book Snapshot using the FIX Market Data Snapshot Full Refresh message (35=W).

Tag number	Tag name	Value	Description
Header		W	Snapshot Full Refresh
262	MDReqID	123	My request ID
107	SecurityDesc	IBM	Security name
1021	MDBookType	1	Top-Of-Book
	NoMDEntries	2	Number of repeating entries
269	MDEntryType	0	A Buy Side Snapshot
271	MDEntrySize	3	Quantity
270	MDEntryPx	9730	Price
269	MDEntryType	1	A Sell Side Snapshot
271	MDEntrySize	5	Quantity
270	MDEntryPx	9750	Price
	Trailer		

2.6.15.5 A Market Data Snapshot for Price-Depth:

The Security description appears once at the head of the snapshot.MDBookType describes the type of information contained within the snapshot. It could be Top-Of-Book, Price-Depth or Order-Depth and describes the type of data included in each item.This is followed by a number of items then the list of items. Each item begins with the MDEntryType tag and describes one side of a price level including Price and total quantity at that price. The requester is expected to order the book using bid or ask and the price. In a Multicast environment, MDReqID would be sent back on a separate Market Data Request Reject message only if there

was a problem with the request. Otherwise, it should not be provided. If provided on the data, then competitors could determine who was requesting and take advantage of the situation. The Client must know that the snapshot relates to its request from the security code.The following table is an example of a Price-Depth snapshot using the FIX Market Data Snapshot Full Refresh message (35=W)

Tag number	Tag name	Value	Description
Header		W	Snapshot Full Refresh
262	MDReqID	123	My request ID
107	SecurityDesc	IBM	Security name
1021	MDBookType	2	Price-Depth
	NoMDEntries	5	Number of repeating entries
269	MDEntryType	0	A Buy Side Snapshot
271	MDEntrySize	3	Quantity
270	MDEntryPx	9730	Price
1023	*PriceLevel*	1	*RowNumber*
269	**MDEntryType**	**0**	**A Buy Side Snapshot**
271	MDEntrySize	10	Quantity
270	MDEntryPx	9720	Price
1023	*PriceLevel*	2	*RowNumber*
269	**MDEntryType**	**0**	**A Buy Side Snapshot**
271	MDEntrySize	15	Quantity
270	MDEntryPx	9710	Price
1023	*PriceLevel*	3	*RowNumber*
269	**MDEntryType**	**0**	**A Buy Side Snapshot**
271	MDEntrySize	5	Quantity
270	MDEntryPx	9700	Price
1023	*PriceLevel*	*4*	*RowNumber*
269	**MDEntryType**	**1**	**A Sell Side Snapshot**
271	MDEntrySize	25	Quantity
270	MDEntryPx	9760	Price
1023	*PriceLevel*	1	*RowNumber*
	Trailer		

2.6.16 A Market Data Snapshot for Order-Depth

The Order-Depth snapshot contains one item for each order. The data includes the side, quantity and price of each order along with the MDEntryID. The MDEntryID is generated by the vendor to uniquely identify this order. It is NOT the ClOrdID of the order that was entered by the owner of the order. The Client can assume that the sequence in the response is the sequence in the book.

The first order at each price level also has an optional PriceLevel tag. This is used in situations where there could be confusion in the ranking of price levels.The following table is an example of an Order-Depth snapshot using the FIX Market Data Snapshot Full Refresh message (35=W)

Tag number	Tag name	Value	Description
Header		W	Snapshot Full Refresh
262	MDReqID	123	My request ID
107	SecurityDesc	IBM	Security name
TBD	MDBookType	3	Order-Depth
268	NoMDEntries	9	Number of Orders
269	MDEntryType	0	A Buy Side Order
271	MDEntrySize	2	Quantity Order 1
37	MDEntryID	123	
270	MDEntryPx	9730	Price
290	*MDEntryPositionNo*	1	*Position at PriceLevel*
1023	*PriceLevel*	1	*RowNumber*
269	**MDEntryType**	**0**	**A Buy Side Snapshot**
271	MDEntrySize	1	Quantity – Order 2
37	MDEntryID	124	
270	MDEntryPx	9730	Price
290	*MDEntryPositionNo*	2	*Position at PriceLevel*
1023	*PriceLevel*	1	*RowNumber*
269	MDEntryType	0	A Buy Side Order
271	MDEntrySize	6	Quantity Order 3
37	MDEntryID	125	
270	MDEntryPx	9720	Price
290	*MDEntryPositionNo*	1	*Position at PriceLevel*

1023	*PriceLevel*	2	*RowNumber*
269	**MDEntryType**	**0**	**269**
271	MDEntrySize	4	Quantity Order 3
37	MDEntryID	126	
270	MDEntryPx	9720	Price
290	*MDEntryPositionNo*	2	*Position at PriceLevel*
1023	*PriceLevel*	2	*RowNumber*
269	MDEntryType	0	A Buy Side Order
271	MDEntrySize	1	Quantity
37	MDEntryID	127	
270	**MDEntryPx**	**9710**	**Price**
290	*MDEntryPositionNo*	1	*Position at PriceLevel*
1023	*PriceLevel*	3	*RowNumber*
269	**MDEntryType**	**0**	**269**
271	MDEntrySize	5	Quantity
37	MDEntryID	128	
270	MDEntryPx	9700	Price
290	*MDEntryPositionNo*	1	*Position at PriceLevel*
1023	*PriceLevel*	4	*RowNumber*
269	**MDEntryType**	**0**	**269**
271	MDEntrySize	10	Quantity
37	MDEntryID	131	
270	MDEntryPx	9690	Price
290	*MDEntryPositionNo*	1	*Position at PriceLevel*
1023	*PriceLevel*	5	*RowNumber*
269	**MDEntryType**	**0**	**269**
271	MDEntrySize	3	Quantity
37	MDEntryID	132	
270	MDEntryPx	9680	Price
290	*MDEntryPositionNo*	1	*Position at PriceLevel*

1023	*PriceLevel*	5	*RowNumber*
	Trailer		

2.6.17 Start Of Day Snapshot

The trading day normally begins with a snapshot of each product. The snapshot provides client systems with all product codes and provides the initial book in markets that support long orders. Long Orders are orders that stay in the book for more than one day, also known as Good-Till-Cancelled or GTC orders.
The snapshot also verifies that the network is working to its full capacity. Markets normally accept orders prior to the opening of the market. Once the snapshot is complete, the vendor can begin to update the book with Pre-open orders, leading up to the opening of the market.

2.6.18 Issues with Snapshot Functionality

The snapshot function allows a Client to request the orderbook for a single instrument or a list of instruments. This function is ideally suited to the workstation in a trading systems environment where it is moving between instruments, requesting a snapshot, then updating that book with the broadcasts. When the trader moves to the next security the workstation will suspend the broadcast of the first instrument, request a snapshot and broadcasts for the next security.

Vendors must be cautious with the use of snapshots to reconnect clients when using multicast networks. During market hours, the time to provide a snapshot of the entire market could be considerable, depending upon the available bandwidth.

There may also be synchronization issues between the snapshot responses and the data feed. Are the snapshots in the correct position in relation to the data feed updates? Clients may be required to cache real time messages while waiting for a snapshot. When the snapshot arrives clients may be expected to apply all cached messages to snapshot in order to create a current book.

Another possible solution is that the Client system subscribe for broadcasts to a limited number of instruments, then request snapshots for a limited number of instruments and apply the updates until the data feed has caught up. Then repeat this process until all instruments have been loaded.

2.6.19 Synchronization of Snapshots and Data Feed

These are the approaches to the synchronization of snapshots that are sent through a broadcast mechanism and updates that are being disseminated.

Synchronous Snapshots. Some Trading systems will send the snapshot request to the orderbook server and the response will be inserted into the data feed. The network must be designed to assure that the response and subsequent updates are delivered in the correct sequence to the requestor. This approach places a significant burden upon the network design to assure that every update and snapshot are delivered in the correct sequence to all clients. With this solution, clients should be able to rely on snapshots being properly sequenced within the data feed. Synchronous snapshots may be supported in either a broadcast of point-to-point feed.

Asynchronous Snapshots. Some vendors provide out-of-band snapshots which the client must synchronize with the real-time data feed. Out-of-band snapshots can either be provided as an unsolicited stream which is

subscribed to by the client or be provided in response to a client request as described in Section 4.1 above. In both cases, snapshots will carry the sequence number of the last real-time message that was applied and clients will be expected to synchronize the snapshot based on this information. Sequence numbers are described in greater detail below.

Sequence Numbers. Some systems provide a sequence number on the snapshot. This sequence number represents the last message in the feed that is reflected in the snapshot. The client system must already be receiving the data feed when it requests a snapshot. When the Client system receives the snapshot it must go back and apply those messages with sequence numbers greater than the sequence number on the snapshot. These messages must be applied to the snapshot in order to ensure that it reflects the latest state of the book. This approach moves the responsibility to the Client system to manage synchronization issues.

Subscribe first and apply all updates. Some systems manage to synchronize snapshots and updates without sequence numbers. The client must subscribe to the updates for an instrument, store those updates while the snapshot is requested. Once the snapshot is received then the updates are applied. Some updates will have been received prior to the snapshot but by applying all updates, the view is correct by the time you get to the last update. This method works well for Top-Of-Book but the vendor must specify synchronization rules for Price-Depth and Order-Depth. For example, a Delete of an order that does not exist is to be ignored. A change of an order that does not exist is to be created. The insertion of an order that is already in the book is to be ignored.

2.7 Eurex EBS Usage Examples [Windows Platform]

The following section presents coding examples of several operations to interface with the EBS system on a PC Platform.

2.7.1 Eurex EBS Create a Socket Connection (Receiver Application)

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

//Include files for Socket

#include <winsock2.h>


// before Winsock functions can be used WSAStartup must be called.
// Error codes will be recieved with WSAGetLastError()

int startWinsock()
{
WSADATA wsa;
if (WSAStartup(MAKEWORD(2,2),&wsa)!=0)
{
        std::cout<<"WSAStartup failed"<<WSAGetLastError()<<endl;
        return 1;
}


int main(int argc, const char *args[])
{

// local variable

int i, rc, iSelRet, bytes;

int sendsize = 1500;

char msg[BUF_SIZE];

define BUF_SIZE 2048;

struct timeval tVal;

struct ip_mreq mcastReq;

struct sockaddr_in remote;


// start Winsock

i=startWinsock();
if(i!=0)
        {
                std::cout<<"error starting winsock"<<endl;
                exit(0);
        }


// create UDP Socket to recieve EBS data
// Declarartion: int socket(int domain, int type, int protocol)
// PF_INET = Protocol Family
// SOCK_DGRAM = UDP datagram feature
// IPPROTO_UDP = UDP protocol
// INVALID SOCKET = return value if call fails

sock=socket(PF_INET,SOCK_DGRAM,IPPROTO_UDP);

if(sock==INVALID_SOCKET)
        {
                std::cout<<" error code: "<<WSAGetLastError()<<endl;
                closesocket(sock);
                exit(0);
        }
```

Figure 16 EBS Receiver Create a Socket Connection

```
//bind socket to local address
// Declaration:
// struct sockaddr_in{
// sa_family_t         sin_family;
// in_port_t           sin_port;
// struct in_addr      sin_addr;
// char                sin_zero(8);
// bind(sock, (struct sockaddr*)&addr, sizeof(addr));
// SOCKET_ERROR = return value if call fails


addr.sin_family=PF_INET;
addr.sin_port=htons(50199);                          // port in network byte order
addr.sin_addr.s_addr=htonl(INADDR_ANY);              // bind socket to any local interface


rc=bind(sock,(struct sockaddr*)&addr,sizeof(addr));
if(rc==SOCKET_ERROR)
        {
                std::cout<<" error code: "<<WSAGetLastError()<<endl;
                closesocket(sock);
                exit(0);
        }



// socketoption to manipulate recieve buffer
// declaration:
// int setsockopt( sock,int level,int optname,const void* optval, socklen_t optlen)
// UDP data will be lost when recieve buffer size is to suffucient
// SOCKET_ERROR = return value if call fails


if(setsockopt(sock,SOL_SOCKET,SO_RCVBUF,(char *)&sendsize,(int )sizeof(sendsize))==
SOCKET_ERROR )
        {
                std::cout<<" error code: "<<WSAGetLastError()<<endl;
                closesocket(sock);
                exit(0);
        }



//join the multicast group we want to receive datagrams from


// declaration:
// struct ip_mreq{
// struct in_addr  imr_multiaddr;
// struct in_addr  imr_interface;
// }
// int setsockopt( sock,int level,int optname,const void* optval, socklen_t optlen)
// UDP data will be lost when recieve buffer size is to suffucient
// SOCKET_ERROR = return value if call fails


mcastReq.imr_multiaddr.s_addr = inet_addr(233.49.81.127); // multicast adrress
mcastReq.imr_interface.s_addr = inet_addr(172.1.1.10);      // interface ip address


if(setsockopt(sock,IPPROTO_IP,IP_ADD_MEMBERSHIP,(char*)&mcastReq,sizeof(mcastReq))==
SOCKET_ERROR )
        {
                std::cout<<" error code: "<<WSAGetLastError()<<endl;
                closesocket(sock);
                exit(0);
        }


}

std::cout << „we are ready to recieve data";
```

Figure 17 EBS Receive Create Socket Connections Cont'd

```
//leave the multicast group if we dont want to receive datagrams from
// declaration:
// struct ip_mreq{
//      struct in_addr  imr_multiaddr;
//      struct in_addr  imr_interface;
// }
// int setsockopt( sock,int level,int optname,const void* optval, socklen_t optlen)
// UDP data will be lost when recieve buffer size is to suffucient
// SOCKET_ERROR = return value if call fails


if(setsockopt(sock,IPPROTO_IP,IP_DROP_MEMBERSHIP,(char*)&mcastReq,sizeof(mcastReq))==
SOCKET_ERROR )
        {
                std::cout<<" error code: "<<WSAGetLastError()<<endl;
                closesocket(sock);
                exit(0);
        }

closesocket(sock);
WSACleanup();
}
```

Figure 18 EBS Receiver Create Socket Connection Cont'd

2.7.2 Eurex EBS Receive UDP/IP Multicast Datagrams (Receiver Application)

```
// start recieve data with select()
// declaration for select()
// int select( int nfds,fd_set *readfds, fd_set *writefds,
// fd_set * exceptfds, struct timeval *timeout)

//Macros for access and manipulate socket descriptor

FD_SET fdset;
FD_ZERO(&fdset);
FD_SET(sock,&fdset);
FD_ISSET(sock,&fdset);
tVal.tv_sec=1;
tVal.tv_usec=600;


do
{

        iSelRet = select(sock,&fdset,NULL,NULL,&tVal);

        switch(iSelRet)
                {

                case SOCKET_ERROR:

                        switch ( selectError )
                        {
                        case WSANOTINITIALISED:
                        cstd::cout<<"polling got not initialised"<<endl;
                                break;
                        case WSAEFAULT:
                                std::cout<<"polling got WSAEFAULT"<<endl;
                                break;
                        case WSAENETDOWN:
                                std::cout<<"polling got WSAENETDOWN"<<endl;
                                break;
                        case WSAEINTR:
                                std::cout<<"polling got WSAEINTR"<<endl;
                                break;
                        case WSAEINPROGRESS:
                                std::cout<<"polling got WSAEINPROGRESS"<<endl;
                                break;
                        case WSAENOTSOCK:
                                std::cout<<"polling got WSAENOTSOCK"<<endl;
                                break;
                        default:
                                std::cout<<"polling got no data"<<endl;
                                break;
                        }

        default:

        // recieve data from socket
        // declaration:
        // ssize_t recvfrom( int s, void * restrict buf, size_t len, int  // flags, struct
        // sockaddr * restrict from, socklen_t * restrict  //  fromlen);
        // returnvalue is number of read bytes

        remote_len = sizeof(remote);

        bytes = recvfrom(sock,msg, sizeof(msg)-1,0,(struct sockaddr*)&remote,&remote_len);

        if (result == SOCKET_ERROR)
        {
        std::cout<<"recvfrom()failed" << endl;

        }

}while(bytes > 0);
```

Figure 19 EBS Receiver - Receive Mulit-Cast UDP Datagrams

2.7.3 Eurex EBS Receive FAST Encoded Messages (Receiver Application)

```
// This example illustrates how a to get EBS messages from a FAST decoder
// The access to the decoder is built via a API

// With recvfrom() you will get the msg from the socket.
// This msg will be put into the FAST decoder to decode the EBS message
// Please visit www.fixprotocol.org/fast for a guideline how to develop a FAST decoder
// This example will decode Reference Data messages


fast::MemBuffer membuffer(msg,sizeof(buffer));

while(membuffer.good())
{


        fast::Message message;
        decoder.decode(membuffer, message);


        const fast::UInt32Value *vtimestamp = message.getUInt32(MFN_TIMESTAMP);
        int timestamp = vtimestamp->getValue();

        const fast::UInt32Value *vsourceId = message.getUInt32(MFN_SRC_ID);
        int sourceId = vsourceId->getValue();

        const fast::UInt32Value *vseqNo = message.getUInt32(MFN_SEQ_NUM);
        int seqNum = vseqNo->getValue();

        const fast::ASCIIStringValue * vprodId = message.getASCII(MFN_PROD_ID);
        string prodId = vprodId->getValue();

        const fast::UInt32Value *vncOfContracts = message.getUInt32(MFN_NO_OF_CONTRACTS);
        int ncOfContracts = vncOfContracts->getValue();

        const fast::UInt32Value *vncOfStreams = message.getUInt32(MFN_NO_OF_STREAMS);
        int ncOfStreams = vncOfStreams->getValue();

                for (i=0;i< ncOfStreams;i++)
                {
                const fast::SequenceValue * vstreams = message.getSequence(MFN_STREAMS);
                const fast::SegmentValue & vstream = (*vstreams)[i];
                const fast::ASCIIStringValue * vstreamService =
                vstream.getASCII(MFN_STREAM_SERVICE);
                string streamService = vstreamService->getValue();
                const fast::ASCIIStringValue * vstreamType = vstream.getASCII(MFN_STREAM_TYPE);
                string streamType = vstreamType->getValue();
                const fast::ASCIIStringValue * vstreamAddr = vstream.getASCII(MFN_STREAM_ADDR);
                string streamAddr = vstreamAddr->getValue();
                const fast::UInt32Value *vstreamPort = vstream.getUInt32(MFN_STREAM_PORT);
                int streamPort = vstreamPort->getValue();
                const fast::UInt32Value *vmarketDepth = vstream.getUInt32(MFN_MKT_DPTH);

                int mktDepth = vmarketDepth->getValue();

                }//for

}//while
```

Figure 20 EBS Receiver – Receive FAST Encoded Messages

2.8 FAST Decode.c

This section presents the decode function for the FAST protocol obtained from the following location:

- http://www.fixprotocol.org/fastdownload

```
// $Id: decode.c,v 1.2 2006/02/09 19:14:25 Daniel.May Exp
// FIX Adapted for STreaming (sm) (FAST Protocol (sm))
// Copyright (c) 2005-2006, Pantor Engineering AB (http://www.pantor.com )
// Copyright (c) 2005-2006, SpryWare LLC (http://www.spryware.com )
// Copyright (c) 2005-2006, FIX Protocol Ltd (http://www.fixprotocol.org)
// All Rights Reserved.
// This work is distributed under the W3C® Software License [1] in the
// hope that it will be useful, but WITHOUT ANY WARRANTY; without even the
// implied warranty of MERCHANTABILITY, SATISFACTORY QUALITY or FITNESS
// FOR A PARTICULAR PURPOSE.
// [1] http://www.w3.org/Consortium/Legal/2002/copyright-software-20021231
// [FPL's Intellectual Property details] http://www.fixprotocol.org/ip
// [FAST Protocol details] http://www.fixprotocol.org/fast
// [FAST Protocol credits] http://fixprotocol.org/fastcredits
// The example application 'decode' will decode a sample data feed from the FAST format back to it's original
// format.  The FAST data is read from stdin and decoded into original messages.  Ouptut is written to stdout.
// See the sample test scripts example.sh (Linux) or example.bat (Windows) for example usage.
#include "fastapi.h"
#include "common.h"
// define the template ID
enum test_tids {    TEST_BASE_TID = 0};
// define the fields
enum test_fields
{   TID         = MAKE_TAG (FAST_TYPE_U32,  FAST_OP_COPY,  TEST_BASE_TID,  0),
   SEQ_NUM      = MAKE_TAG (FAST_TYPE_U32,  FAST_OP_INCR,  TEST_BASE_TID,  1),
   TIME         = MAKE_TAG (FAST_TYPE_U32,  FAST_OP_COPY,  TEST_BASE_TID,  2),
   SYMBOL       = MAKE_TAG (FAST_TYPE_STR,  FAST_OP_COPY,  TEST_BASE_TID,  3),
   PRICE        = MAKE_TAG (FAST_TYPE_U32,  FAST_OP_COPY,  TEST_BASE_TID,  4),
   SHARES       = MAKE_TAG (FAST_TYPE_U32,  FAST_OP_COPY,  TEST_BASE_TID,  5),
   EXCH         = MAKE_TAG (FAST_TYPE_STR,  FAST_OP_COPY,  TEST_BASE_TID,  6)};
int main(int argc, char* argv[]) {
    fast_codec_t *codec;
    u32 tid,seq_num,price,time,shares;
    char symbol[16],exch[2];
                // We want binary input from stdio and stdout
    init_platform_io();
   // Create the Codec
                codec = fast_create_codec ();
                assert(NULL!=codec);
                // Decode the data
                fprintf(stdout,"; SeqNum, Time, Symbol,Price, Shares, Exch\r\n");
    // keep decoding while data is present
                while(fast_decode_new_msg (codec, TID) > 0)
                {
                                fast_decode_u32 (codec, TID, &tid);
                                assert(tid==1);
                                // SeqNum is first 6 chars in input file, starting at pos 0
                                fast_decode_u32 (codec, SEQ_NUM, &seq_num);
                                // Time is next 6 chars in input file, starting at pos 7
                                fast_decode_u32 (codec, TIME, &time);
                                // Symbol is next 3 chars in input file, starting at pos 14
                                fast_decode_str (codec, SYMBOL, symbol, 3);
                                // Price is next 5 chars in input file, starting at pos 18
                                fast_decode_u32 (codec, PRICE, &price);
                                // Shares are next 3 chars in input file, starting at pos 24
                                fast_decode_u32 (codec, SHARES, &shares);
                                // Exchange is next char in input file, starting at pos 28
                                fast_decode_str (codec, EXCH, exch,1);
                                // we are done with this message
                                if(fast_decode_end_msg (codec,0) != FAST_ERR_NONE)
                                                fast_print_error (codec, stderr);
                                else
```

```
                fprintf(stdout,"%06d %06d %-3.3s %05d %03d %c\r\n",seq_num,time,symbol,price,shares,exch[0]);
        }

        // clean up
        fast_destroy_codec (codec);
        return 0;
}
```

2.8.1 FAST Functions

fast_codec_t * fast_create_codec (void)
Create and initialize the codec.

Int fast_destroy_codec (fast_codec_t *codec)
Destroy the codec and release associated memory.

Void fast_reset_state (fast_codec_t *codec, fast_tag_t tag)
Reset the codec state for a specific tag.

Int fast_set_codec_input (fast_codec_t *codec, FILE *fptr)
Set the input FILE stream for a codec.

int fast_set_codec_output (fast_codec_t *codec, FILE *fptr)
Set the output FILE stream for a codec.

fast_tag_t fast_make_tag (fast_op_t, fast_type_t, u32 tid, u32 slot)
Make a FAST tag.

int fast_decode_new_msg (fast_codec_t *codec, fast_tag_t tag)
Decode the first tag of a new message.

int fast_decode_end_msg (fast_codec_t *codec, fast_tag_t tag)
Complete the decoding of a message.

int fast_decode_i32 (fast_codec_t *codec, fast_tag_t tag, i32 *data)
Decode a 32-bit signed integer.

int fast_decode_u32 (fast_codec_t *codec, fast_tag_t tag, u32 *data)
Decode a 32-bit unsigned integer.

int fast_decode_str (fast_codec_t *codec, fast_tag_t tag, u8 *data, int size)
Decode an ASCII String.

int fast_encode_new_msg (fast_codec_t *codec, fast_tag_t tag)
Encode the first tag of a new message.

int fast_encode_end_msg (fast_codec_t *codec, fast_tag_t tag)
Complete the encoding of a message.

int fast_encode_i32 (fast_codec_t *codec, fast_tag_t tag, i32 data)
Encode a 32-bit signed integer.

int fast_encode_u32 (fast_codec_t *codec, fast_tag_t tag, u32 data)
Encode a 32-bit unsigned integer.

int fast_encode_str (fast_codec_t *codec, fast_tag_t tag, u8 *data, int size)
Encode an ASCII String.

int fast_print_error (fast_codec_t *codec, FILE *fptr)
Format and print the last reported codec error to a human readable string to a FILE stream.

const char * fast_error_string (fast_codec_t *codec)
Convert the last reported codec error to a human readable string.

u32 fast_ascii_to_u32 (u8 *data, int size)

Convert a u8 byte arrray of characters to a u32.

2.8.2 FAST Function Documentation

u32 fast_ascii_to_u32(u8 * *data*,
int *size*
)

Convert a u8 byte arrray of characters to a u32.

Parameters:

data Pointer to a u8 containing ASCII number
size Lenght of data

Returns:

The converted u32 number

Remarks:

This function will convert an ASCII string of numbers to a binary value All characters must be in range of '0' - '9'

Definition at line 1357 of file fastapi.c.

fast_codec_t* fast_create_codec(void)

Create and initialize the codec.

Returns:

A pointer to opaque type fast_codec_t

Remarks:

This function will allocate a segment of memory on your behalf to manage the codec state information. The pointer returned should be treated as an opaque handle, it is a required parameter to most other fastapi functions.

Definition at line 1317 of file fastapi.c.

References FAST_CODEC_MAGIC, and init_buffer().

int fast_decode_end_msg(fast_codec_t * *codec*,
fast_tag_t *tag*
)

Complete the decoding of a message.

Parameters:

codec Pointer to a fast_codec_t
tag The fast_codec_tag we are decoding

Remarks:

This function must be called after the last tag of a message has been decoded

Definition at line 730 of file fastapi.c.

References check_codec().

```
int fast_decode_i32( fast_codec_t * codec,
                     fast_tag_t     tag,
                     i32 *          data
                   )
```

Decode a 32-bit signed integer.

Parameters:

codec	Pointer to a fast_codec_t
tag	The fast_codec_tag we are decoding
data	Pointer to a i32 that will contain the result

Definition at line 843 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_get_i32(), cv_is_valid(), cv_set_i32(), FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_I32, get_tag_op(), parse_i32(), and value_error().

```
int fast_decode_new_msg( fast_codec_t * codec,
                         fast_tag_t     tag
                       )
```

Decode the first tag of a new message.

Parameters:

codec	Pointer to a fast_codec_t
tag	The fast_codec_tag we are decoding

Remarks:

This function must be called to decode the first tag of a new message. The fast_codec_tag should be the template ID

Definition at line 720 of file fastapi.c.

References check_codec(), and parse_pmap().

```
int fast_decode_str( fast_codec_t * codec,
                     fast_tag_t     tag,
                     u8 *           data,
                     int            size
                   )
```

Decode an ASCII String.

Parameters:

codec Pointer to a fast_codec_t
tag The fast_codec_tag we are decoding
data Pointer to a char * that will contain the result
size The maximum number of chars data can hold

Remarks:

The resulting sting will NOT be NULL terminated, this function will only fill the string with char's decoded from the input.

Definition at line 742 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_get_str(), cv_is_valid(), cv_set_str(), FAST_ERR_SIZE, FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_STR, get_tag_op(), parse_str(), set_err(), and value_error().

```
int fast_decode_u32( fast_codec_t * codec,
                     fast_tag_t     tag,
                     u32 *          data
                   )
```

Decode a 32-bit unsigned integer.

Parameters:

codec Pointer to a fast_codec_t
tag The fast_codec_tag we are decoding
data Pointer to a u32 that will contain the result

Definition at line 950 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_get_u32(), cv_is_valid(), cv_set_u32(), FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_U32, get_tag_op(), parse_i32(), parse_u32(), and value_error().

```
int fast_destroy_codec( fast_codec_t * codec )
```

Destroy the codec and release associated memory.

Parameters:

codec Pointer to a fast_codec_t

Remarks:

This function will free the segment of memory on your behalf originally allocated by the fast_create_codec() function. The codec is invalid upon return.

Definition at line 1338 of file fastapi.c.

References FAST_CODEC_MAGIC, and fast_codec_t::magic.

```
int fast_encode_end_msg( fast_codec_t * codec,
                         fast_tag_t     tag
```

)

Complete the encoding of a message.

Parameters:

codec Pointer to a fast_codec_t

tag The fast_codec_tag we are encoding

Remarks:

This function must be called after the last tag of a message has been encoded

Definition at line 1142 of file fastapi.c.

References check_codec(), FAST_ERR_CALL_SEQ, flush_msg(), fast_codec_t::in_message, and set_err().

```
int fast_encode_i32( fast_codec_t * codec,
                     fast_tag_t     tag,
                     i32            data
                   )
```

Encode a 32-bit signed integer.

Parameters:

codec Pointer to a fast_codec_t

tag The fast_codec_tag we are decoding

data i32 integer containing the value to encode

Definition at line 1158 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_eq_i32(), cv_set_i32(), emit_i32(), FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_I32, and get_tag_op().

```
int fast_encode_new_msg( fast_codec_t * codec,
                         fast_tag_t     tag
                       )
```

Encode the first tag of a new message.

Parameters:

codec Pointer to a fast_codec_t

tag The fast_codec_tag we are decoding

Remarks:

This function must be called to encode the first tag of a new message. The fast_codec_tag should be the template ID

Definition at line 1127 of file fastapi.c.

References check_codec(), fast_codec_t::curr_tag, FAST_ERR_CALL_SEQ, fast_codec_t::in_message, and

set_err().

int fast_encode_str(fast_codec_t * *codec*,
fast_tag_t *tag*,
u8 * *data*,
int *size*
)

Encode an ASCII String.

Parameters:

codec Pointer to a fast_codec_t
tag The fast_codec_tag we are decoding
data char * containing the value to encode
size The number of characters to encode

Remarks:

This function will encode exactly size number of char's, it will not not assume a NULL terminated string.

Definition at line 1205 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_eq_str(), cv_get_str(), cv_set_str(), emit_str(), FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_STR, find_char_delta_offset(), and get_tag_op().

int fast_encode_u32(fast_codec_t * *codec*,
fast_tag_t *tag*,
u32 *data*
)

Encode a 32-bit unsigned integer.

Parameters:

codec Pointer to a fast_codec_t
tag The fast_codec_tag we are decoding
data u32 integer containing the value to encode

Definition at line 1259 of file fastapi.c.

References bad_op_error(), check_codec(), check_type(), cv_eq_u32(), cv_get_u32(), cv_is_valid(), cv_set_u32(), emit_i32(), emit_u32(), FAST_OP_COPY, FAST_OP_DELTA, FAST_OP_INCR, FAST_OP_NONE, FAST_TYPE_U32, and get_tag_op().

const char* fast_error_string(fast_codec_t * *codec*)

Convert the last reported codec error to a human readable string.

Parameters:

codec Pointer to a fast_codec_t

Returns:

The error string

Definition at line 351 of file fastapi.c.

References fast_codec_error_t::code, fast_codec_t::error, fast_codec_error_t::fn, format_error_code(), format_tag_op(), format_tag_type(), get_tag_op(), get_tag_slot(), get_tag_tid(), get_tag_type(), fast_codec_error_t::tag, and fast_codec_error_t::text.

Referenced by fast_print_error(), and set_err().

fast_tag_t fast_make_tag(fast_op_t ,
fast_type_t ,
u32 *tid*,
u32 *slot*
)

Make a FAST tag.

int fast_print_error(fast_codec_t * *codec*,
FILE * *fptr*
)

Format and print the last reported codec error to a human readable string to a FILE stream.

Parameters:

codec Pointer to a fast_codec_t

fptr FILE stream

Returns:

The error string

Definition at line 369 of file fastapi.c.

References fast_error_string().

void fast_reset_state(fast_codec_t * *codec*,
fast_tag_t *tag*
)

Reset the codec state for a specific tag.

Parameters:

codec Pointer to a fast_codec_t

tag The fast_codec_tag we are reseting

Remarks:

This function will reset the state of the codec for the specific tag.

int fast_set_codec_input(fast_codec_t * *codec*,

FILE * *fptr*
)

Set the input FILE stream for a codec.

Parameters:

codec Pointer to a fast_codec_t
fptr FILE Pointer to a FILE stream

Remarks:

The default input stream in stdin

int fast_set_codec_output(fast_codec_t * *codec*,
FILE * *fptr*
)

Set the output FILE stream for a codec.

Parameters:

codec Pointer to a fast_codec_t
fptr FILE Pointer to a FILE stream

Remarks:

The default output stream in stdout

2.8.3 Eurex EBS Header Files [EurexHeaderPackageUnix.tar.gz]

The Eurex EBS System provides members a downloadable zip file:

- **EurexHeaderPackageUnix.tar.gz** - This file provides header files be used to linked in with the Receiver Application(s). The include files are extracted into three folders
  - Error Codes Folder
    - XEUR_elbcode.h, Eurex Error Code Messages
  - Tables Folder
    - **DRIV_app_rid.h** – Collectio of all request ids defined for Eurex Application request
    - **DRIV_rsrc_index.hxx** – Header file contains resource access levels available to Eurex Members
    - **DRIV_streamtypes.hxx** – Collection of all stream types which specify the broadcast data stream defines its own stream types unique to the context of Eurex exchange applications
  - Messages Folder
    - **DRIV_common_structs.hxx** – Header file contains all common structures
    - **DRIV_data_types.hxx**
    - **DRIV_deprecated.hxx –** Header file contains definitions which are depreciated but still provided. Recommend replace with values from DRIV_data_types.hxx
    - **DRIV_login.h -** Header file contains authorization data used with VCI_Login entry point
    - **Application Request Header Files** – Header files for Eurex VALUES API requests, responses, subscriptions. Section Five of "***VALUE API Member Front End Development Guide Volume -2 Eurex Application Request Final Version Eurex Release 10.0***" provides details of the specific calls

2.9 Multi-Thread Unix Programming Notes

2.9.1 Streams Queue(s)

Queues

The queue is the fundamental component of a Stream. It is the interface between a STREAMS module and the rest of the Stream, and is the repository for deferred message processing. For each instance of an open driver or pushed module or Stream head, a pair of queues is allocated, one for the read side of the Stream and one for the write side.

The **RD**(9F), **WR**(9F), and **OTHERQ**(9F) routines allow reference of one queue from the other. Given a queue **RD**(9F) returns a pointer to the read queue, **WR**(9F) returns a pointer to the write queue and **OTHERQ**(9F) returns a pointer to the opposite queue of the pair. Also see **QUEUE**(9S).

By convention, queue pairs are depicted graphically as side- by-side blocks, with the write queue on the left and the read queue on the right (see Figure Figure 7–7).

Figure 7–7 Queue Pair Allocation

queue(9S) Structure

As previously discussed, messages are ordered in message queues. Message queues, message priority, service procedures, and basic flow control all combine in STREAMS. A service procedure processes the messages in its queue. If there is no service procedure for a queue, **putq**(9F) does not schedule the queue to be run. The module developer must ensure that the messages in the queue are processed. Message priority and flow control are associated with message queues.

The queue structure is defined in `stream.h` as the `typedef queue_t`, and has the following public elements:

```
struct  qinit   *q_qinfo;       /* procs and limits for queue */
struct  msgb    *q_first;       /* first data block */
struct  msgb    *q_last;        /* last data block */
struct  queue   *q_next;        /* Q of next stream */
struct  queue   *q_link;        /* to next Q for scheduling */
void            *q_ptr;         /* to private data structure */
size_t          q_count;        /* number of bytes on Q */
uint            q_flag;         /* queue state */
ssize_t         q_minpsz;       /
* min packet size accepted by this module */
```

(continued)

Figure 22 Multi-Threaded Streams Queues –queue(9s).

(Continuation)

```
ssize_t         q_maxpsz;       /
* max packet size accepted by this module */
size_t          q_hiwat;        /* queue high--water mark */
size_t          q_lowat;        /* queue low--water mark */
```

q_first points to the first message on the queue, and q_last points to the last message on the queue. q_count is used in flow control and contains the total number of bytes contained in normal and high-priority messages in band 0 of this queue. Each band is flow controlled individually and has its own count. See " qband(9S) Structure " on page 107 for more details. qsize(9F) can be used to determine the total number of messages on the queue. q_flag indicates the state of the queue. See Table 7–3 for the definition of these flags.

q_minpsz contains the minimum packet size accepted by the queue, and q_maxpsz contains the maximum packet size accepted by the queue. These are suggested limits, and some implementations of STREAMS may not enforce them. The SunOS™ Stream head enforces these values but is voluntary at the module level. Design modules to handle messages of any size.

q_hiwat indicates the limiting maximum number of bytes that can be put on a queue before flow control occurs. q_lowat indicates the lower limit where STREAMS flow control is released.

q_ptr is the element of the queue structure where modules can put values or pointers to data structures that are private to the module. This data can include any information required by the module for processing messages passed through the module, such as state information, module IDs, routing tables, and so on. Effectively, this element can be used any way the module or driver writer chooses. q_ptr can be accessed or changed directly by the driver, and is typically initialized in the open(9E) routine.

When a queue pair is allocated, streamtab initializes q_qinfo, and module_info initializes q_minpsz, q_maxpsz, q_hiwat, and q_lowat. Copying values from the module_info structure allows them to be changed in the queue without modifying the streamtab and module_info values.

Figure 23 Multi-Threaded Streams Queues –queue(9s) Cont'd

2.9.2 Posix Thread Programming - Pthreads

Reference: https://computing.llnl.gov/tutorials/pthreads/#Abstract

In the Unix Operating System a Unix Process is an executable file executed under control of the operating system scheduler. The process contains its own memory address space and working environment. The process under control of the scheduler manages the states of the program execution state. The parameter include Process ID, Process Group ID, User ID and Group ID, Environment Parameters, Working Directory, Program Instructions, Register, Stack/Heap, File Descriptor, Signal & Exception Management routines, Shared Libraries and Inter-Process Communications (queues, pipes, semaphores, shared memory)

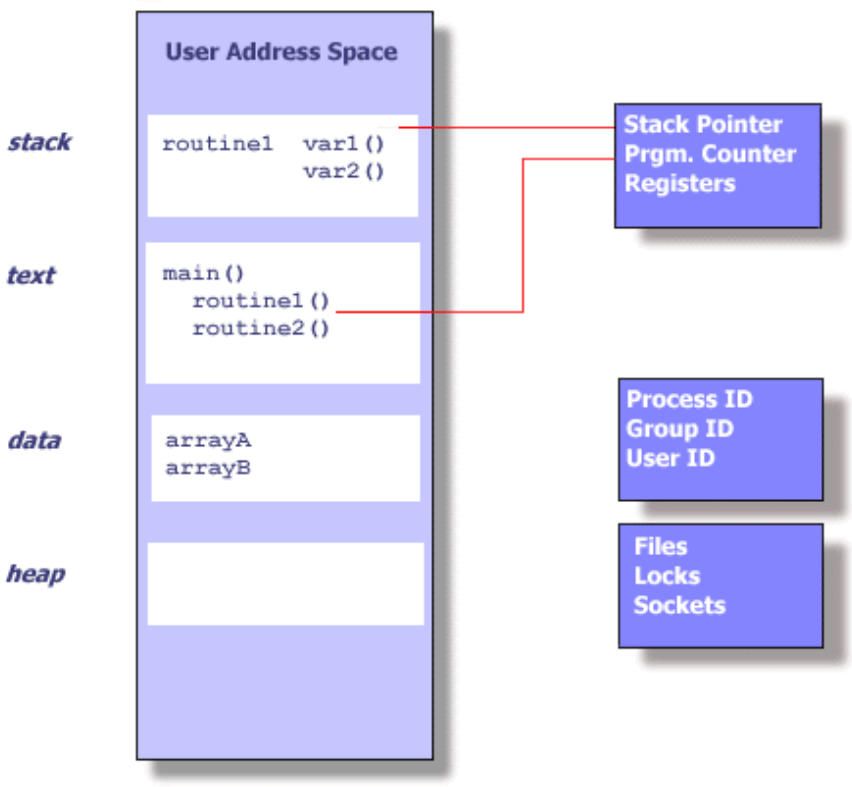


Figure 24 Unix Process

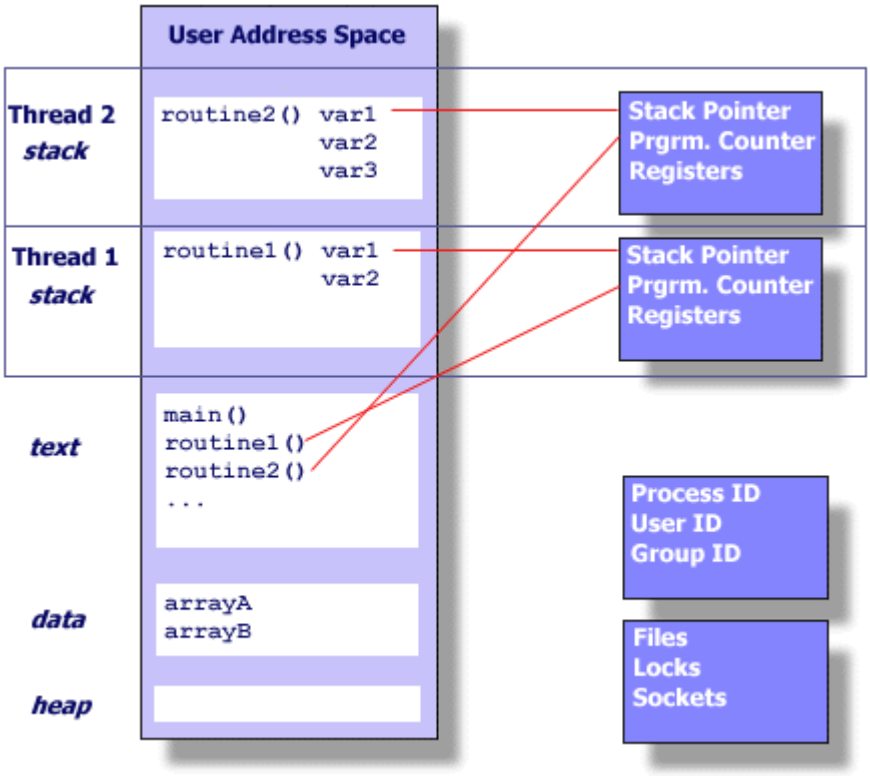


Figure 25 Unix Process With Threads

Threads exist with the context of a process. Threads work with the resources of the process. However, thread execution are under the management of the operating system scheduler and execute as independent entities. A thread execution flow is provided an independent set of resources but share the same memory address space and process resources of the Unix process. Each thread within the Unix Process are provided the following resources for management of independent execution. Stack Point, Register, Scheduling Priorities, Thread Specific Data, Set of pending and blocked signals. Threads have an independent execution flow put exist only for the duration of the life cycle of the Unix Process.

In the Unix programming environment, a standardized programming interface for threads was established (IEEE POSIX 1003.1c 1995 www.ieee.org) . The standard known as Posix Thread or Pthreads is provided for application API programming. Pthreads are defined as a set of C language programming types and procedures calls. A header file pthread.h and a link library are provided to the application program to support POSIX Thread Application programming.

2.9.2.1 Creating and Terminating Threads

In the initial creation of a Unix process only a single thread is executing. In the main() program as the single default thread. All other threads must be created explicitly.

The following are pthreads routines established for threads:

- **pthread_create (thread,attr,start_routine,arg)** – Creates a new thread and makes it executable. When a thread is created a threadID is returned. The thread created can execute **pthread_equal()** to obtain its threadID

- **pthread_exit (status) –** Terminates execution of a thread
- **pthread_attr_init (attr) –** Initializes or Creates Thread Attributes
- **pthread_attr_destroy (attr) –** Destroy or Remove Thread Attributes

The following is a coding example to create and terminate threads. In the creation of the thread one of the arguments is the routine that will be executed by the thread.

```
#include <pthread.h>
#include <stdio.h>
#define NUM_THREADS     5

void *PrintHello(void *threadid)
{
   int tid;
   tid = (int)threadid;
   printf("Hello World! It's me, thread #%d!\n", tid);
   pthread_exit(NULL);
}

int main (int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc, t;
   for(t=0; t<NUM_THREADS; t++){
      printf("In main: creating thread %d\n", t);
      rc = pthread_create(&threads[t], NULL, PrintHello, (void *)t);
      if (rc){
         printf("ERROR; return code from pthread_create() is %d\n", rc);
         exit(-1);
      }
   }
   pthread_exit(NULL);
}
```

Figure 26 Coding Example Pthread Creation and Termination

2.9.2.2 Thread Safe Considerations

Thread safe refers to the ability to execute multiple threads simultaneously without causing shared data to become corrupted or creating race conditions between the threads. The POSIX Threads API provides capabilities to synchronize thread execution to prevent these issues.

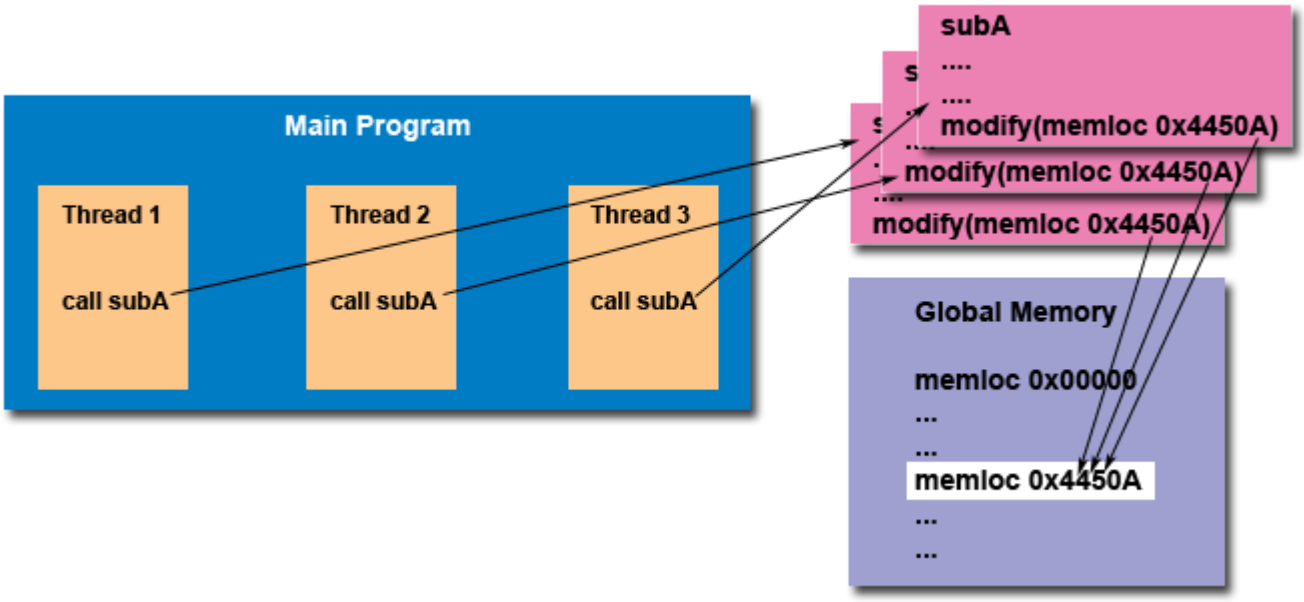


Figure 27 Thread Safe Programming Considerations

Some Synchronization methods provided in the POSIX API are:

- **pthread_join(threaded, status)** – blocks the calling thread from executing until the thread that was joined to completes execution (subordinate thread executes the pthread_exit() function).
- **pthread_detach()** – routine used to explicitly detach a thread

2.9.2.3 pthreads - Mutex Variables

Mutex (Mutual Exclusion) Variables is a primary means to implement thread synchronization and protecting shared data. When multiple write operations occur. The Mutex variable ensure the when several threads operates on the same data thank that the updates are managed.

A typical sequence uses the following steps:

- Create and Initialize Mutex Variable
- Several Threads Attempt to lock Mutex
- Only one is permitted by the Unix scheduler and that thread owns the mutex
- The owner thread performs a set of operations
- The owner unlocks the Mutex
- Another thread aquires the Mutex and performs data operations on the Mutex Variable(s)
- Finally mutex is destroyed

When several threads compete for a mutex the losing threads block at the call until an unblocking call releases the Mutex Variable(s).

2.9.2.3.1 pthreads – Creating and Destroying Mutexs

The following POSIX threads API routines support these operations:

- **pthread mutex init (mutex,attr)**
- **pthread mutex destroy (mutex)**
- **pthread mutexattr init (attr)**
- **pthread mutexattr destroy (attr)**

2.9.2.3.2 pthreads – Locking and Unlocking Mutexs

The following POSIX threads API routines support these operations:

- **pthread mutex lock (mutex) –** used by a thread to lock a mutex resource. It the mutex resource is locked already, then the thread will be blocked from access. Hence the thread execution is halted until the mutex resource becomes unblocked.
- **pthread mutex trylock (mutex) –** attempt to lock a mutex resource, however if the resource is unavailable the routine will immediately return a busy status. In this case, the thread execution continues.
- **pthread mutex unlock (mutex) –** used to release access to a mutex varaiable. The Unix scheduler will then permit another thread attempting to access the mutex resource to continue execution, hence unblocking another thread.

The following is a program example of using Mutex Resources

```
#include <pthread.h>
#include <stdio.h>
#include <malloc.h>

/*
The following structure contains the necessary information
to allow the function "dotprod" to access its input data and
place its output into the structure.
*/

typedef struct
 {
   double      *a;
   double      *b;
   double     sum;
   int     veclen;
 } DOTDATA;

/* Define globally accessible variables and a mutex */

#define NUMTHRDS 4
#define VECLEN 100
   DOTDATA dotstr;
   pthread_t callThd[NUMTHRDS];
   pthread_mutex_t mutexsum;

/*
The function dotprod is activated when the thread is created.
All input to this routine is obtained from a structure
of type DOTDATA and all output from this function is written into
this structure. The benefit of this approach is apparent for the
multi-threaded program: when a thread is created we pass a single
argument to the activated function - typically this argument
is a thread number. All  the other information required by the
function is accessed from the globally accessible structure.
*/

void *dotprod(void *arg)
{

   /* Define and use local variables for convenience */

   int i, start, end, offset, len ;
   double mysum, *x, *y;
   offset = (int)arg;

   len = dotstr.veclen;
   start = offset*len;
   end   = start + len;
   x = dotstr.a;
   y = dotstr.b;
```

```
   /*
   Perform the dot product and assign result
   to the appropriate variable in the structure.
   */

   mysum = 0;
   for (i=start; i<end ; i++)
    {
      mysum += (x[i] * y[i]);
    }

   /*
   Lock a mutex prior to updating the value in the shared
   structure, and unlock it upon updating.
   */
   pthread_mutex_lock (&mutexsum);
   dotstr.sum += mysum;
   pthread_mutex_unlock (&mutexsum);

   pthread_exit((void*) 0);
}

/*
The main program creates threads which do all the work and then
print out result upon completion. Before creating the threads,
the input data is created. Since all threads update a shared structure,
we need a mutex for mutual exclusion. The main thread needs to wait for
all threads to complete, it waits for each one of the threads. We specify
a thread attribute value that allow the main thread to join with the
threads it creates. Note also that we free up handles when they are
no longer needed.
*/

int main (int argc, char *argv[])
{
   int i;
   double *a, *b;
   int status;
   pthread_attr_t attr;

   /* Assign storage and initialize values */
   a = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));
   b = (double*) malloc (NUMTHRDS*VECLEN*sizeof(double));

   for (i=0; i<VECLEN*NUMTHRDS; i++)
    {
     a[i]=1.0;
     b[i]=a[i];
    }

   dotstr.veclen = VECLEN;
   dotstr.a = a;
   dotstr.b = b;
   dotstr.sum=0;

   pthread_mutex_init(&mutexsum, NULL);
```

```
   /* Create threads to perform the dotproduct  */
   pthread_attr_init(&attr);
   pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

        for(i=0; i<NUMTHRDS; i++)
        {
        /*
        Each thread works on a different set of data.
        The offset is specified by 'i'. The size of
        the data for each thread is indicated by VECLEN.
        */
        pthread_create( &callThd[i], &attr, dotprod, (void *)i);
        }

        pthread_attr_destroy(&attr);

        /* Wait on the other threads */
        for(i=0; i<NUMTHRDS; i++)
        {
          pthread_join( callThd[i], (void **)&status);
        }

   /* After joining, print out the results and cleanup */
   printf ("Sum =  %f \n", dotstr.sum);
   free (a);
   free (b);
   pthread_mutex_destroy(&mutexsum);
   pthread_exit(NULL);
}
```

Figure 28 POSIX threads coding example using Mutex Resources

2.9.3 Unix File Descriptor Processing Multi-Threaded

```
#include <sys/time.h>
#include <sys/types.h>
#include <sys/select.h>
int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval
*timeout);
FD_SET(int fd, fd_set *fdset);
FD_CLR(int fd, fd_set *fdset);
FD_ISSET(int fd, fd_set *fdset);
FD_ZERO(fd_set *fdset);
```

2.9.3.1 DESCRIPTION

The `select()` function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, `select()` blocks, up to the specified *timeout* interval, until the specified condition is true for at least one of the specified file descriptors or until a signal arrives that needs to be delivered.

The `select()` function supports regular file descriptors, console descriptors, pipe descriptors, FIFO descriptors, socket descriptors, and communication port descriptors. A `select()` on file descriptors that refer to other types of file fails with `errno` set to `EBADF`.

The *nfds* argument specifies the range of file descriptors to be tested. The `select()` function tests file descriptors in the range of 0 to *nfds*-1.

If the *readfds* argument is not `NULL`, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the *writefds* argument is not `NULL`, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the *exceptfds* argument is not `NULL`, it points to an object of type `fd_set` that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

On successful completion, the objects pointed to by the *readfds*, *writefds*, and *exceptfds* arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than *nfds*, the corresponding bit is set on successful completion if it was set on input and the associated condition is true for that file descriptor.

If the *timeout* argument is not a `NULL` pointer, it points to an object of type `struct timeval` that specifies a maximum interval to wait for the selection to complete. If the *timeout* argument points to an object of type

`struct timeval` whose members are 0, `select()` does not block. If the *timeout* argument is a `NULL` pointer, `select()` blocks until an event causes one of the masks to be returned with a valid (non-zero) value or until a signal occurs that needs to be delivered. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, `select()` completes successfully and returns 0.

The use of a timeout does not affect any pending timers set by `alarm()` or `setitimer()`.

If the *readfds*, *writefds*, and *exceptfds* arguments are all `NULL` pointers and the *timeout* argument is not `NULL`, `select()` blocks for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *exceptfds* arguments are all `NULL` pointers and the *timeout* argument is `NULL`, `select()` blocks until interrupted by a signal.

File descriptors associated with regular files always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *exceptfds* arguments are not modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *exceptfds* arguments have all bits set to 0.

If a process is blocked on a `select()` while waiting for input from a socket and the sending process closes the socket, the `select()` notes this as an exception rather than as data. Therefore, if the `select()` is not currently looking for exceptions, it waits indefinitely.

File descriptor masks of type `fd_set` can be initialized and tested with `FD_CLR()`, `FD_ISSET()`, `FD_SET()`, and `FD_ZERO()` macros.

`FD_CLR(`*fd*`, &`*fdset*`)`
: Clears the bit for the file descriptor *fd* in the file descriptor set *fdset*.

`FD_ISSET(`*fd*`, &`*fdset*`)`
: Returns a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

`FD_SET(`*fd*`, &`*fdset*`)`
: Sets the bit for the file descriptor *fd* in the file descriptor set *fdset*.

`FD_ZERO(&`*fdset*`)`
: Initializes the file descriptor set *fdset* to have zero bits for all file descriptors.

Unexpected errors may occur if *fd* is less than 0 or greater than or equal to `FD_SETSIZE` in any of these macros.

Note:
: The default value of `FD_SETSIZE` (currently 256) is smaller than the default limit on the number of open files. To accommodate programs that may use a larger number of open files with `select()`, it is possible to increase this size within a program by providing a larger definition of `FD_SETSIZE` before the inclusion of `<sys/select.h>`.

2.9.3.2 PARAMETERS

nfds
: Specifies how many descriptors should be examined. The descriptors checked are 0 through *nfds*-1.

readfds
: Points to a bit mask that specifies the file descriptors to check for reading.

writefds
: Points to a bit mask that specifies the file descriptors to check for writing.

exceptfds
: Points to a bit mask that specifies the file descriptors to check for exception conditions.

timeout
: When non-`NULL`, contains the address of a `struct timeval` that specifies how long to wait for the required condition before returning to the caller. When `NULL`, forces the call to block until a descriptor becomes ready or until a signal occurs.

2.9.3.3 RETURN VALUES

If successful, `select()` returns the number of ready descriptors that are contained in the bit masks. If the time limit expires, select returns zero and sets `errno` to `EINTR`. On failure, it returns -1 and sets `errno` to one of the following values:

`EBADF`
: One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor or specified a file descriptor that does not support selection.

`EINTR`
: A signal interrupted the call or the time limit expired.

`EISDIR`
: One or more the file descriptor sets specified a file descriptor that refers to an open directory.

`EINVAL`
: A component of *timeout* is outside the acceptable range.

2.9.3.4 MULTITHREAD SAFETY LEVEL

MT-Safe.

2.10 Velox Mailbox Application Interface Coding Example

2.10.1 Velox Mailbox API Header Interface

The section presents the header file API to be used by the Receiving Application to support Interprocess communications within the Velox Server Environment and to the Trading Workstation. The receiving application uses the Velox libgen.a library to link this interface into the executable binary file created.

```
#ifndef _MBX_H
#define _MBX_H 1

#include <signal.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

#define MBX_MAX 10            /* default max number of slots */
#define DIR_READ       1
#define DIR_WRITE      2
#define MBX_NODE_LEN 8
                              /* see ipcs -ql */
#define MBX_MAX_QSIZE 16384   /* if bigger requested then use PGS */
#define MBX_MAX_MSIZE 4000    /* if bigger requested then use PGS */
#define MBX_MAX_PIDRD 10      /* max number of PIDs of readers of mbx */
                              /* stored in logical <mbxname>_RD */

typedef struct {
  char  node[MBX_NODE_LEN];   /* node name of sender */
  char  pname[16];            /* zero-terminated process name of sender */
} MBX_NPNAME;

typedef struct {
  int   pgsidx;               /* index of data in PGS or 0 */
  MBX_NPNAME npname;          /* nodename and process name of sender */
} MBX_HEADER;

typedef struct {
  long  aci;                  /* long mtype */
  MBX_HEADER mbx_header;
  char  data[1];
} MBX_BUF;

typedef struct {
  char  logname[32];          /* Zero terminated uppercase log mbx name */
  char   realname[32];        /* LNM translation of logname or copy */
  char  target_node[MBX_NODE_LEN]; /* for WRITE only */
  int   direction;            /* READ / WRITE / BOTH */
  int   id;                   /* msqID */
  int   max_msize;            /* max message data size excluding header */
  int   max_qsize;            /* max bytes in message queue */
  int   receiver;             /* PID of (primary) receiver */
  int   use_pgs;              /* mbx has a PGS */
  int   mbxpgs_size;          /* size of PGS [bytes] */
  /* MBXPGS *mbxpgs_ptr; (later)   points to memory of PGS */
  MBX_BUF *buffer;            /* size = sizeof(MBX_BUF) + max_msize - 1 */
} MBX_DESCRIPTOR;

typedef struct {
  int   direction;            /* READ / WRITE / BOTH */
  int   id;                   /* msqID */
  int   receiver;             /* PID of (primary) receiver */
  int   use_pgs;              /* mbx has a PGS */
  struct msqid_ds msqbuf;     /* returned by msgctl() */
  char   logname[32];         /* log actual length only */
} MBX_PELINFO;                /* for mbx_info() */
```

```
void mbx_init(int num_mbx, int use_pel, int use_dynmap);
int  mbx_rec_init(const char *name, int *msize, int qsize);
int  mbx_info(const char *name);
MBX_BUF *mbx_rec(const char *name, long *aci, int nowait, int *size, char **msg);
int  mbx_send(const char *name, int aci, int size, const void *msg);
void mbx_send_retries(int retries, int mswait);
#endif /* mbx.h */
```

Figure 29 Velox Mailbox API Header Interface

2.10.2 Velox Mailbox Coding Example

Presented here is a testing example of how an application interfaces with. The Velox Server Mailbox messages. The Mailbox Header file mbox.h provided by Velox provides the Receiving Application the Mailbox API interface routines.

```
/* test_mbx.c  test functions of package mbx.c
 * ----------
 *
 * created:    26-APR-2001 H. Moerchen
 * 1. revision:        30-APR-2001 H. Moerchen
 *      add optional arguments 2 , 3
 *
 * Arguments: 1-3
 * 1  ProcessName      test_mbx0, test_mbx1, ..., test_mbx9
 *                     1 .. 10 instances of test_mbx may run in parallel
 *                     PEL should be enabled for at least TEST_MBX0.
 * 2  generr_enable    0 no default - to be used for timing tests
 *                     1 yes
 * 3  delay [msec]     0    default
 *                     >0   useful to test "MBX full"
 *
 * The processes TEST_MBXn build a ring of processes for receiving and
 * forwarding MBX messages.
 * They must be started in descending order of n.
 * At start each process TEST_MBXn checks if process n+1 exists. If it
 * exists then it is the receiver of forwarded messages.
 * Otherwise received messages are forwarded to TEST_MBX0.
 *
 * Each process creates its own mailbox <processname>_MBX and reads from it.
 * After receiving a message the ACI is decremented. When ACI still > 0 the
 * process forwards the message to the next TEST_MBXn process.
 * ACI == 987654321 is a request to terminate.
 *
 * Injection of an initial message is done once or more with tool mbs.
 * This test environment may be used for debugging and tuning mbx.c
 *
 * Timing results on 600 MHz PIII
 * ------------------------------
 Send + Receive of 1000 messages with full PEL logging
 A) after tuning of mbx.c
 1.     header only            110 msec
 2.     header + 40 bytes data 130 msec
 3.     header + 400 bytes data       130 msec
 B) after 1. revision - remove one generr() call
 1.     header only             30 msec
 2.     header + 40 bytes data  50 msec
 3.     header + 400 bytes data        50 msec

 1000 generr() calls take 80 msec ==> generr.c needs tuning
 */

/* include files */
#include       <stdio.h>
#include        <stdlib.h>
#include        <string.h>
#include        <stdio.h>
#include        <sys/time.h>
#include        <unistd.h>
#include        "pel.h"
#include        "mbx.h"
#include       "wait_msec.h"
#include        "lnm.h"
#include       "pgs.h"

/* static data */
```

```
extern char *ProcessName;       /* set by gen_init() */

int     main    (int argc, char *argv[])
{
  int   status;
  int   msize;                  /* size of max message */
  int   i;
  char  pname_lc;               /* last character of ProcessName */
  char   *p;
  char  mymbxname[32];
  char   partpname[16];
  char   partmbxname[32];
  char  partpidstr[13];
  MBX_BUF mbx_buf;
  MBX_BUF *pmbx_buf;
  char  *msg_text;
  int   msg_textsize;
  long  aci;
  int   generr_enable = 0;
  int   mswait = 0;
  int    mypid = getpid(); /* needs unistd.h */


  if (!(p = getenv("PROCESSNAME"))) {
    if (argc < 2) {
      printf("usage: %s mailboxname aci ...", argv[0]);
      return(1);
    }
    p = argv[1];
  }
  if (argc > 2)
    generr_enable = atoi(argv[2]);
  if (argc > 3)
    mswait = atoi(argv[3]);
  gen_init(p, GEN_PUNI);        /* ProcessName must be unique */
  pgs_init(1);                  /* for shared global section PEL */
  pelpstart(__FILE__, mypid, 0);
  mbx_init(2, 1, 0);

  /* create my mailbox */
  sprintf(mymbxname, "%s_MBX", ProcessName);
  msize = 0;                   /* use default */
  status = mbx_rec_init(mymbxname, &msize, 0);
  if (status) return(status); /* error reporting done in mbx_rec_init() */
  generr("mailbox %s created with max msg size = %d", mymbxname, msize);

  /* determine name of process to forward received messages */
  i = strlen(ProcessName);
  pname_lc = *(ProcessName + i - 1);
  if (pname_lc < '9') {
    pname_lc += 1;
    sprintf(partpname, "%.*s%c", i-1, ProcessName, pname_lc);
    if (status = readlnm("PNAME", partpname, partpidstr, sizeof(partpidstr) -1) > 0)
      generr("my partner process %s is active with PID = %s", partpname, partpidstr);
    else
      pname_lc = '0';
  }
  else
      pname_lc = '0';
  /* and construct name of target mailbox */
  sprintf(partmbxname, "%.*s%c_MBX", i-1, ProcessName, pname_lc);

  for (;;) {
    aci = 0;                    /* accept any message */
    if (pmbx_buf = mbx_rec(mymbxname, &aci, 0, &msg_textsize, &msg_text)) {
      if (generr_enable)
        generr("got msg  aci=%d  size=%d  ptr=%0x",
               aci, msg_textsize, (int) msg_text);
      if (aci == 987654321)
        generrex("exit request received");
      if (--aci <= 0)
```

```
        continue;
      status = mbx_send(partmbxname, aci, msg_textsize, msg_text);
      if (status != 0) {
        if (status > 0)
          generr("mbx_send() to %s - mailbox full", partmbxname);
        else
          generrst(status, "mbx_send() to %s", partmbxname);
      }
    } else {
      generrex("mbx_rec(%s) unsuccessful", mymbxname);
    }
    if (mswait > 0)
      wait_msec(mswait);
  }
}
```

Figure 30 Coding Example Interfacing with Velox Server Mailbox Communication System

2.11 EBS Testing Considerations

To properly test the application. Here one would need to this at three levels.

1. **Unit Testing application** with via a program the allow one to act like the EBS system and send messages through the system. This driver program would need to read files and/or generate messages desired. Consideration here should be to develop a message generation program also.

2. **Simulation Testing**, after completion of Unit Testing. The program should be testing with Eurex EBS Simulation interfaces

3. **Production Testing,** final check out step for this effort.

2.11.1 EBS Testing Data Sets [ebs_sample_data.zip]

The Eurex EBS provides to Members an archive file that contains testing data sets to be used to validate Member Receiving Applications.

1. Binary – Contains the raw binary messages. Please note that the messages are not delimited. The binary data is continuous in the file.
2. Step-by-step debug – Contains the step by step debug output for all the messages in the file. This output is intended to help the members to understand the message decoding process.
3. XML – Contains the decoded messages in XML format.

Binary Data:

The raw binary data as mentioned above occurs continuously in the file. Standard file binary read IO operations should be used to access the data from the file.

Step-by-step decoding:

A row of debug output represents decoding logic applied to the stream bytes. Every row of the debug output is as follows.

FIELD :: OPTIONALITY :: TYPE :: OPERATOR :: PRESENCE :: --LEN-- BYTES :: [OLDVALUE] DECODEDVALUE

FIELD	This is name of the field. In addition to the fields described in the EBS interface specification PMAP (for FAST Presence Map) and TID (for Template ID) are used.
OPTIONALITY	This is the presence attribute of the field from the message template. Please note that default is mandatory. MAN and OPT is used to denote this.
TYPE	This is the FAST data type of the field.
OPERATOR	This is the FAST operator for the field. If no operator is present in the message template [NONE] is printed.
The above four values are known directly from the EBS templates. They are printed here so that the reader gets a complete view of the decoding of the field.	
PRESENCE	This says whether the field occurred in the stream. This is known from the PMAP.
EX/MN/LN/SR	This is optionally printed for decimal and string fields with FAST delta operator. EX stands for Exponent MN stands for Mantissa LN stands for length to be subtracted SR stands for the string bytes to use added to the old value.
--LEN-- BYTES	These are the bytes actually sent for the field in the stream.
{OLDVALUE}	This is printed when the new field value is derived from earlier sent value. This is the old value from the dictionary.
DECODEDVALUE	This is the final decoded value of the field. This is also the new value to be stored in the FAST dictionary.

XML:

This is the final message instance sent. Every message corresponds to the debug output in the step-by-step decode output. This output will help the reader to have a complete view of the sent fields in a message.

Disclaimer:

- The data used in the examples is simulated. The products and/or contracts used are not available in production environment.
- The intention of providing this information is to help the future users of EBS to understand the format in which data will be sent and to demonstrate the decoding process.

Some description of the data files:

The data set carries four files:

1. REF.DAT – Around 1000 packets from the Static Reference data stream have been captured and decoded.
2. SNAPSHOTMSGS.DAT – Carries the beginning of day snapshots with empty book indicator.
3. DELTAMSGS.DAT – Carries order book delta messages, trade information messages, product state change messages. (Watch out for cntrId = 60001)
4. SNAPSHOTMSGS_60001.DAT – Carries snapshots after some order book entries was created for some contracts. (Watch out for cntrId = 60001)
5. DELTAREQS.DAT – Carries strategy request messages and strategy order book delta messages.

Figure 31 EBS Testing Data Sets

2.12 Eurex EBS Multi-Cast Subscription Channels for Simulation Mode

product	stream	channel	depth	addr (A)	port	addr (B)	port
$SYS	1	STAT	0	233.49.81.127	50199	233.49.81.255	50199
ABBN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
ABBN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
ABBN	1	B	5	233.49.81.18	50132	233.49.81.146	50132
ABBN	1	C	10	233.49.81.19	50132	233.49.81.147	50132
AFR	1	S	0	233.49.81.16	50133	233.49.81.144	50133
AFR	1	A	0	233.49.81.17	50132	233.49.81.145	50132
AFR	1	B	5	233.49.81.18	50132	233.49.81.146	50132
AFR	1	C	10	233.49.81.19	50132	233.49.81.147	50132
AIRG	1	S	0	233.49.81.40	50133	233.49.81.168	50133
AIRG	1	A	0	233.49.81.41	50132	233.49.81.169	50132
AIRG	1	B	10	233.49.81.42	50132	233.49.81.170	50132
ALBK	1	S	0	233.49.81.8	50133	233.49.81.136	50133
ALBK	1	A	0	233.49.81.9	50132	233.49.81.137	50132
ALBK	1	B	5	233.49.81.10	50132	233.49.81.138	50132
ALBK	1	C	10	233.49.81.11	50132	233.49.81.139	50132
ALV	1	S	0	233.49.81.8	50133	233.49.81.136	50133
ALV	1	A	0	233.49.81.9	50132	233.49.81.137	50132
ALV	1	B	5	233.49.81.10	50132	233.49.81.138	50132
ALV	1	C	10	233.49.81.11	50132	233.49.81.139	50132
ALVF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
ALVF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
ALVF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
AVE	1	S	0	233.49.81.8	50133	233.49.81.136	50133
AVE	1	A	0	233.49.81.9	50132	233.49.81.137	50132
AVE	1	B	5	233.49.81.10	50132	233.49.81.138	50132
AVE	1	C	10	233.49.81.11	50132	233.49.81.139	50132
BARC	1	S	0	233.49.81.16	50133	233.49.81.144	50133
BARC	1	A	0	233.49.81.17	50132	233.49.81.145	50132
BARC	1	B	3	233.49.81.18	50132	233.49.81.146	50132
BARC	1	C	6	233.49.81.19	50132	233.49.81.147	50132
BARC	1	D	10	233.49.81.20	50132	233.49.81.148	50132
BARF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
BARF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
BARF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
BAY	1	S	0	233.49.81.8	50133	233.49.81.136	50133
BAY	1	A	0	233.49.81.9	50132	233.49.81.137	50132
BAY	1	B	3	233.49.81.10	50132	233.49.81.138	50132
BAY	1	C	6	233.49.81.11	50132	233.49.81.139	50132
BAY	1	D	10	233.49.81.12	50132	233.49.81.140	50132
BBVD	1	S	0	233.49.81.8	50133	233.49.81.136	50133
BBVD	1	A	0	233.49.81.9	50132	233.49.81.137	50132
BBVD	1	B	3	233.49.81.10	50132	233.49.81.138	50132
BBVD	1	C	6	233.49.81.11	50132	233.49.81.139	50132

BBVD	1	D	10	233.49.81.12	50132	233.49.81.140	50132
BMW	1	S	0	233.49.81.8	50133	233.49.81.136	50133
BMW	1	A	0	233.49.81.9	50132	233.49.81.137	50132
BMW	1	B	3	233.49.81.10	50132	233.49.81.138	50132
BMW	1	C	6	233.49.81.11	50132	233.49.81.139	50132
BMW	1	D	10	233.49.81.12	50132	233.49.81.140	50132
BMWF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
BMWF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
BMWF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
BOTD	1	S	0	233.49.81.8	50133	233.49.81.136	50133
BOTD	1	A	0	233.49.81.9	50132	233.49.81.137	50132
BOTD	1	B	3	233.49.81.10	50132	233.49.81.138	50132
BOTD	1	C	6	233.49.81.11	50132	233.49.81.139	50132
BOTD	1	D	10	233.49.81.12	50132	233.49.81.140	50132
BP	1	S	0	233.49.81.16	50133	233.49.81.144	50133
BP	1	A	0	233.49.81.17	50132	233.49.81.145	50132
BP	1	B	3	233.49.81.18	50132	233.49.81.146	50132
BP	1	C	6	233.49.81.19	50132	233.49.81.147	50132
BP	1	D	10	233.49.81.20	50132	233.49.81.148	50132
CIS	1	S	0	233.49.81.8	50133	233.49.81.136	50133
CIS	1	A	0	233.49.81.9	50132	233.49.81.137	50132
CIS	1	B	2	233.49.81.10	50132	233.49.81.138	50132
CIS	1	C	4	233.49.81.11	50132	233.49.81.139	50132
CIS	1	D	6	233.49.81.12	50132	233.49.81.140	50132
CIS	1	E	8	233.49.81.13	50132	233.49.81.141	50132
CIS	1	F	10	233.49.81.14	50132	233.49.81.142	50132
CONF	1	S	0	233.49.81.60	50101	233.49.81.188	50101
CONF	1	A	0	233.49.81.61	50100	233.49.81.189	50100
CONF	1	B	10	233.49.81.62	50100	233.49.81.190	50100
CSGF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
CSGF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
CSGF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
CSGN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
CSGN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
CSGN	1	B	2	233.49.81.18	50132	233.49.81.146	50132
CSGN	1	C	4	233.49.81.19	50132	233.49.81.147	50132
CSGN	1	D	6	233.49.81.20	50132	233.49.81.148	50132
CSGN	1	E	8	233.49.81.21	50132	233.49.81.149	50132
CSGN	1	F	10	233.49.81.22	50132	233.49.81.150	50132
DB1	1	S	0	233.49.81.8	50133	233.49.81.136	50133
DB1	1	A	0	233.49.81.9	50132	233.49.81.137	50132
DB1	1	B	10	233.49.81.10	50132	233.49.81.138	50132
DBK	1	S	0	233.49.81.8	50133	233.49.81.136	50133
DBK	1	A	0	233.49.81.9	50132	233.49.81.137	50132
DBK	1	B	10	233.49.81.10	50132	233.49.81.138	50132
EMI	1	S	0	233.49.81.16	50133	233.49.81.144	50133
EMI	1	A	0	233.49.81.17	50132	233.49.81.145	50132
EMI	1	B	10	233.49.81.18	50132	233.49.81.146	50132

ERCB	1	S	0	233.49.81.8	50133	233.49.81.136	50133
ERCB	1	A	0	233.49.81.9	50132	233.49.81.137	50132
ERCB	1	B	10	233.49.81.10	50132	233.49.81.138	50132
EXS1	1	S	0	233.49.81.64	50133	233.49.81.192	50133
EXS1	1	A	0	233.49.81.65	50132	233.49.81.193	50132
EXS1	1	B	10	233.49.81.66	50132	233.49.81.194	50132
EXSF	1	S	0	233.49.81.64	50133	233.49.81.192	50133
EXSF	1	A	0	233.49.81.65	50132	233.49.81.193	50132
EXSF	1	B	10	233.49.81.66	50132	233.49.81.194	50132
F0BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F0BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F0BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F0PM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0PM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0PM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F0PQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0PQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0PQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F0PY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F0PY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F0PY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1PE	1	S	0	233.49.81.72	50133	233.49.81.200	50133
F1PE	1	A	0	233.49.81.73	50132	233.49.81.201	50132
F1PE	1	B	10	233.49.81.74	50132	233.49.81.202	50132
F1PM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1PM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1PM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1PQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1PQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1PQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1PT	1	S	0	233.49.81.72	50133	233.49.81.200	50133
F1PT	1	A	0	233.49.81.73	50132	233.49.81.201	50132
F1PT	1	B	10	233.49.81.74	50132	233.49.81.202	50132

F1PY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F1PY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F1PY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F1TA	1	S	0	233.49.81.48	50133	233.49.81.176	50133
F1TA	1	A	0	233.49.81.49	50132	233.49.81.177	50132
F1TA	1	B	20	233.49.81.50	50132	233.49.81.178	50132
F2BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F2BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F2BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F2MX	1	S	0	233.49.81.48	50133	233.49.81.176	50133
F2MX	1	A	0	233.49.81.49	50132	233.49.81.177	50132
F2MX	1	B	20	233.49.81.50	50132	233.49.81.178	50132
F2PE	1	S	0	233.49.81.72	50133	233.49.81.200	50133
F2PE	1	A	0	233.49.81.73	50132	233.49.81.201	50132
F2PE	1	B	10	233.49.81.74	50132	233.49.81.202	50132
F2PM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2PM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2PM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F2PQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2PQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2PQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F2PY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F2PY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F2PY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F3BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F3BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F3BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F3BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F3BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F3BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F3BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F3BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F3BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F4BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F4BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F4BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132

F4XM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4XM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4XM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F4XQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4XQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4XQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F4XY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
F4XY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
F4XY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
F5C0	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5C0	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5C0	1	B	10	233.49.81.6	50132	233.49.81.134	50132
F5C1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5C1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5C1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
F5E0	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5E0	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5E0	1	B	10	233.49.81.6	50132	233.49.81.134	50132
F5E1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5E1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5E1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
F5H0	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5H0	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5H0	1	B	10	233.49.81.6	50132	233.49.81.134	50132
F5H1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
F5H1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
F5H1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FD01	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FD01	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FD01	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FD02	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FD02	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FD02	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FD03	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FD03	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FD03	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FDAX	1	S	0	233.49.81.108	50101	233.49.81.236	50101
FDAX	1	A	0	233.49.81.109	50100	233.49.81.237	50100
FDAX	1	B	10	233.49.81.110	50100	233.49.81.238	50100
FDAX	1	C	20	233.49.81.111	50100	233.49.81.239	50100
FDAY	1	S	0	233.49.81.52	50101	233.49.81.180	50101
FDAY	1	A	0	233.49.81.53	50100	233.49.81.181	50100
FDAY	1	B	10	233.49.81.54	50100	233.49.81.182	50100
FEO1	1	S	0	233.49.81.76	50101	233.49.81.204	50101
FEO1	1	A	0	233.49.81.77	50100	233.49.81.205	50100
FEO1	1	B	10	233.49.81.78	50100	233.49.81.206	50100
FESB	1	S	0	233.49.81.52	50101	233.49.81.180	50101
FESB	1	A	0	233.49.81.53	50100	233.49.81.181	50100

FESB	1	B	10	233.49.81.54	50100	233.49.81.182	50100
FESI	1	S	0	233.49.81.52	50133	233.49.81.180	50133
FESI	1	A	0	233.49.81.53	50132	233.49.81.181	50132
FESI	1	B	10	233.49.81.54	50132	233.49.81.182	50132
FEST	1	S	0	233.49.81.52	50133	233.49.81.180	50133
FEST	1	A	0	233.49.81.53	50132	233.49.81.181	50132
FEST	1	B	10	233.49.81.54	50132	233.49.81.182	50132
FESX	1	S	0	233.49.81.96	50101	233.49.81.224	50101
FESX	1	A	0	233.49.81.97	50100	233.49.81.225	50100
FESX	1	B	5	233.49.81.98	50100	233.49.81.226	50100
FESX	1	C	10	233.49.81.99	50100	233.49.81.227	50100
FESX	1	D	20	233.49.81.100	50100	233.49.81.228	50100
FESY	1	S	0	233.49.81.52	50133	233.49.81.180	50133
FESY	1	A	0	233.49.81.53	50132	233.49.81.181	50132
FESY	1	B	10	233.49.81.54	50132	233.49.81.182	50132
FEU3	1	S	0	233.49.81.76	50101	233.49.81.204	50101
FEU3	1	A	0	233.49.81.77	50100	233.49.81.205	50100
FEU3	1	B	10	233.49.81.78	50100	233.49.81.206	50100
FFOX	1	S	0	233.49.81.48	50133	233.49.81.176	50133
FFOX	1	A	0	233.49.81.49	50132	233.49.81.177	50132
FFOX	1	B	20	233.49.81.50	50132	233.49.81.178	50132
FGBL	1	S	0	233.49.81.80	50101	233.49.81.208	50101
FGBL	1	A	0	233.49.81.81	50100	233.49.81.209	50100
FGBL	1	B	5	233.49.81.82	50100	233.49.81.210	50100
FGBL	1	C	10	233.49.81.83	50100	233.49.81.211	50100
FGBL	1	D	20	233.49.81.84	50100	233.49.81.212	50100
FGBM	1	S	0	233.49.81.92	50101	233.49.81.220	50101
FGBM	1	A	0	233.49.81.93	50100	233.49.81.221	50100
FGBM	1	B	10	233.49.81.94	50100	233.49.81.222	50100
FGBM	1	C	20	233.49.81.95	50100	233.49.81.223	50100
FGBS	1	S	0	233.49.81.88	50101	233.49.81.216	50101
FGBS	1	A	0	233.49.81.89	50100	233.49.81.217	50100
FGBS	1	B	20	233.49.81.90	50100	233.49.81.218	50100
FGBX	1	S	0	233.49.81.60	50101	233.49.81.188	50101
FGBX	1	A	0	233.49.81.61	50100	233.49.81.189	50100
FGBX	1	B	10	233.49.81.62	50100	233.49.81.190	50100
FGCL	1	S	0	233.49.81.60	50101	233.49.81.188	50101
FGCL	1	A	0	233.49.81.61	50100	233.49.81.189	50100
FGCL	1	B	10	233.49.81.62	50100	233.49.81.190	50100
FGCS	1	S	0	233.49.81.60	50101	233.49.81.188	50101
FGCS	1	A	0	233.49.81.61	50100	233.49.81.189	50100
FGCS	1	B	10	233.49.81.62	50100	233.49.81.190	50100
FGTI	1	S	0	233.49.81.48	50101	233.49.81.176	50101
FGTI	1	A	0	233.49.81.49	50100	233.49.81.177	50100
FGTI	1	B	20	233.49.81.50	50100	233.49.81.178	50100
FGTM	1	S	0	233.49.81.60	50101	233.49.81.188	50101
FGTM	1	A	0	233.49.81.61	50100	233.49.81.189	50100
FGTM	1	B	10	233.49.81.62	50100	233.49.81.190	50100

FGTS	1	S	0	233.49.81.60	50133	233.49.81.188	50133
FGTS	1	A	0	233.49.81.61	50132	233.49.81.189	50132
FGTS	1	B	10	233.49.81.62	50132	233.49.81.190	50132
FIC3	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FIC3	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FIC3	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FIC4	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FIC4	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FIC4	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FIC5	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FIC5	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FIC5	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FID3	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FID3	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FID3	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FID4	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FID4	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FID4	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FID5	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FID5	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FID5	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FRDX	1	S	0	233.49.81.48	50133	233.49.81.176	50133
FRDX	1	A	0	233.49.81.49	50132	233.49.81.177	50132
FRDX	1	B	20	233.49.81.50	50132	233.49.81.178	50132
FSC1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FSC1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FSC1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FSE1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FSE1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FSE1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FSH1	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FSH1	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FSH1	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FSH2	1	S	0	233.49.81.4	50133	233.49.81.132	50133
FSH2	1	A	0	233.49.81.5	50132	233.49.81.133	50132
FSH2	1	B	10	233.49.81.6	50132	233.49.81.134	50132
FSMI	1	S	0	233.49.81.104	50133	233.49.81.232	50133
FSMI	1	A	0	233.49.81.105	50132	233.49.81.233	50132
FSMI	1	B	10	233.49.81.106	50132	233.49.81.234	50132
FSMI	1	C	20	233.49.81.107	50132	233.49.81.235	50132
FSTB	1	S	0	233.49.81.52	50101	233.49.81.180	50101
FSTB	1	A	0	233.49.81.53	50100	233.49.81.181	50100
FSTB	1	B	10	233.49.81.54	50100	233.49.81.182	50100
FSTX	1	S	0	233.49.81.48	50101	233.49.81.176	50101
FSTX	1	A	0	233.49.81.49	50100	233.49.81.177	50100
FSTX	1	B	20	233.49.81.50	50100	233.49.81.178	50100
FT2M	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT2M	1	A	0	233.49.81.69	50132	233.49.81.197	50132

FT2M	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FT2Q	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT2Q	1	A	0	233.49.81.69	50132	233.49.81.197	50132
FT2Q	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FT2Y	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT2Y	1	A	0	233.49.81.69	50132	233.49.81.197	50132
FT2Y	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FT4M	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT4M	1	A	0	233.49.81.69	50132	233.49.81.197	50132
FT4M	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FT4Q	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT4Q	1	A	0	233.49.81.69	50132	233.49.81.197	50132
FT4Q	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FT4Y	1	S	0	233.49.81.68	50133	233.49.81.196	50133
FT4Y	1	A	0	233.49.81.69	50132	233.49.81.197	50132
FT4Y	1	B	10	233.49.81.70	50132	233.49.81.198	50132
FTDX	1	S	0	233.49.81.48	50133	233.49.81.176	50133
FTDX	1	A	0	233.49.81.49	50132	233.49.81.177	50132
FTDX	1	B	20	233.49.81.50	50132	233.49.81.178	50132
FVDX	1	S	0	233.49.81.36	50133	233.49.81.164	50133
FVDX	1	A	0	233.49.81.37	50132	233.49.81.165	50132
FVDX	1	B	10	233.49.81.38	50132	233.49.81.166	50132
FVSM	1	S	0	233.49.81.36	50133	233.49.81.164	50133
FVSM	1	A	0	233.49.81.37	50132	233.49.81.165	50132
FVSM	1	B	10	233.49.81.38	50132	233.49.81.166	50132
FVSX	1	S	0	233.49.81.36	50133	233.49.81.164	50133
FVSX	1	A	0	233.49.81.37	50132	233.49.81.165	50132
FVSX	1	B	10	233.49.81.38	50132	233.49.81.166	50132
G0BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G0BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G0BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G0BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G0BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G0BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G0BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G0BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G0BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G2BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G2BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G2BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G2BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G2BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G2BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G2BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G2BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G2BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G4BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G4BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132

G4BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G4BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G4BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G4BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G4BS	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G4BS	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G4BS	1	B	10	233.49.81.70	50132	233.49.81.198	50132
G4BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
G4BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
G4BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
GAZ	1	S	0	233.49.81.16	50133	233.49.81.144	50133
GAZ	1	A	0	233.49.81.17	50132	233.49.81.145	50132
GAZ	1	B	10	233.49.81.18	50132	233.49.81.146	50132
GAZF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
GAZF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
GAZF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
GSK	1	S	0	233.49.81.16	50133	233.49.81.144	50133
GSK	1	A	0	233.49.81.17	50132	233.49.81.145	50132
GSK	1	B	10	233.49.81.18	50132	233.49.81.146	50132
HMSB	1	S	0	233.49.81.8	50133	233.49.81.136	50133
HMSB	1	A	0	233.49.81.9	50132	233.49.81.137	50132
HMSB	1	B	10	233.49.81.10	50132	233.49.81.138	50132
HOLN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
HOLN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
HOLN	1	B	10	233.49.81.18	50132	233.49.81.146	50132
LOG	1	S	0	233.49.81.16	50133	233.49.81.144	50133
LOG	1	A	0	233.49.81.17	50132	233.49.81.145	50132
LOG	1	B	5	233.49.81.18	50132	233.49.81.146	50132
LOG	1	C	10	233.49.81.19	50132	233.49.81.147	50132
LUK	1	S	0	233.49.81.16	50133	233.49.81.144	50133
LUK	1	A	0	233.49.81.17	50132	233.49.81.145	50132
LUK	1	B	5	233.49.81.18	50132	233.49.81.146	50132
LUK	1	C	10	233.49.81.19	50132	233.49.81.147	50132
MSF	1	S	0	233.49.81.8	50133	233.49.81.136	50133
MSF	1	A	0	233.49.81.9	50132	233.49.81.137	50132
MSF	1	B	5	233.49.81.10	50132	233.49.81.138	50132
MSF	1	C	10	233.49.81.11	50132	233.49.81.139	50132
NDB	1	S	0	233.49.81.8	50133	233.49.81.136	50133
NDB	1	A	0	233.49.81.9	50132	233.49.81.137	50132
NDB	1	B	3	233.49.81.10	50132	233.49.81.138	50132
NDB	1	C	6	233.49.81.11	50132	233.49.81.139	50132
NDB	1	D	10	233.49.81.12	50132	233.49.81.140	50132
NNIA	1	S	0	233.49.81.16	50133	233.49.81.144	50133
NNIA	1	A	0	233.49.81.17	50132	233.49.81.145	50132
NNIA	1	B	3	233.49.81.18	50132	233.49.81.146	50132
NNIA	1	C	6	233.49.81.19	50132	233.49.81.147	50132
NNIA	1	D	10	233.49.81.20	50132	233.49.81.148	50132
NOA3	1	S	0	233.49.81.8	50133	233.49.81.136	50133

NOA3	1	A	0	233.49.81.9	50132	233.49.81.137	50132
NOA3	1	B	3	233.49.81.10	50132	233.49.81.138	50132
NOA3	1	C	6	233.49.81.11	50132	233.49.81.139	50132
NOA3	1	D	10	233.49.81.12	50132	233.49.81.140	50132
NOVN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
NOVN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
NOVN	1	B	3	233.49.81.18	50132	233.49.81.146	50132
NOVN	1	C	6	233.49.81.19	50132	233.49.81.147	50132
NOVN	1	D	10	233.49.81.20	50132	233.49.81.148	50132
O1BM	1	S	0	233.49.81.68	50133	233.49.81.196	50133
O1BM	1	A	0	233.49.81.69	50132	233.49.81.197	50132
O1BM	1	B	10	233.49.81.70	50132	233.49.81.198	50132
O1BQ	1	S	0	233.49.81.68	50133	233.49.81.196	50133
O1BQ	1	A	0	233.49.81.69	50132	233.49.81.197	50132
O1BQ	1	B	10	233.49.81.70	50132	233.49.81.198	50132
O1BY	1	S	0	233.49.81.68	50133	233.49.81.196	50133
O1BY	1	A	0	233.49.81.69	50132	233.49.81.197	50132
O1BY	1	B	10	233.49.81.70	50132	233.49.81.198	50132
ODAX	1	S	0	233.49.81.120	50133	233.49.81.248	50133
ODAX	1	A	0	233.49.81.121	50132	233.49.81.249	50132
ODAX	1	B	10	233.49.81.122	50132	233.49.81.250	50132
OES1	1	S	0	233.49.81.116	50133	233.49.81.244	50133
OES1	1	A	0	233.49.81.117	50132	233.49.81.245	50132
OES1	1	B	10	233.49.81.118	50132	233.49.81.246	50132
OES5	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OES5	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OES5	1	B	10	233.49.81.26	50132	233.49.81.154	50132
OESB	1	S	0	233.49.81.28	50133	233.49.81.156	50133
OESB	1	A	0	233.49.81.29	50132	233.49.81.157	50132
OESB	1	B	10	233.49.81.30	50132	233.49.81.158	50132
OESI	1	S	0	233.49.81.28	50133	233.49.81.156	50133
OESI	1	A	0	233.49.81.29	50132	233.49.81.157	50132
OESI	1	B	10	233.49.81.30	50132	233.49.81.158	50132
OESX	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OESX	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OESX	1	B	5	233.49.81.26	50132	233.49.81.154	50132
OESX	1	C	10	233.49.81.27	50132	233.49.81.155	50132
OEU3	1	S	0	233.49.81.76	50101	233.49.81.204	50101
OEU3	1	A	0	233.49.81.77	50100	233.49.81.205	50100
OEU3	1	B	10	233.49.81.78	50100	233.49.81.206	50100
OFOX	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OFOX	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OFOX	1	B	10	233.49.81.26	50132	233.49.81.154	50132
OGBL	1	S	0	233.49.81.32	50101	233.49.81.160	50101
OGBL	1	A	0	233.49.81.33	50100	233.49.81.161	50100
OGBL	1	B	5	233.49.81.34	50100	233.49.81.162	50100
OGBL	1	C	10	233.49.81.35	50100	233.49.81.163	50100
OGBM	1	S	0	233.49.81.32	50101	233.49.81.160	50101

OGBM	1	A	0	233.49.81.33	50100	233.49.81.161	50100
OGBM	1	B	10	233.49.81.34	50100	233.49.81.162	50100
OGBS	1	S	0	233.49.81.32	50101	233.49.81.160	50101
OGBS	1	A	0	233.49.81.33	50100	233.49.81.161	50100
OGBS	1	B	10	233.49.81.34	50100	233.49.81.162	50100
OGTI	1	S	0	233.49.81.24	50101	233.49.81.152	50101
OGTI	1	A	0	233.49.81.25	50100	233.49.81.153	50100
OGTI	1	B	10	233.49.81.26	50100	233.49.81.154	50100
OGTS	1	S	0	233.49.81.60	50101	233.49.81.188	50101
OGTS	1	A	0	233.49.81.61	50100	233.49.81.189	50100
OGTS	1	B	10	233.49.81.62	50100	233.49.81.190	50100
OSM1	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OSM1	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OSM1	1	B	10	233.49.81.26	50132	233.49.81.154	50132
OSMI	1	S	0	233.49.81.112	50133	233.49.81.240	50133
OSMI	1	A	0	233.49.81.113	50132	233.49.81.241	50132
OSMI	1	B	5	233.49.81.114	50132	233.49.81.242	50132
OSMI	1	C	10	233.49.81.115	50132	233.49.81.243	50132
OSTX	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OSTX	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OSTX	1	B	10	233.49.81.26	50132	233.49.81.154	50132
OTDX	1	S	0	233.49.81.24	50133	233.49.81.152	50133
OTDX	1	A	0	233.49.81.25	50132	233.49.81.153	50132
OTDX	1	B	10	233.49.81.26	50132	233.49.81.154	50132
PHI1	1	S	0	233.49.81.8	50133	233.49.81.136	50133
PHI1	1	A	0	233.49.81.9	50132	233.49.81.137	50132
PHI1	1	B	10	233.49.81.10	50132	233.49.81.138	50132
REP	1	S	0	233.49.81.8	50133	233.49.81.136	50133
REP	1	A	0	233.49.81.9	50132	233.49.81.137	50132
REP	1	B	10	233.49.81.10	50132	233.49.81.138	50132
RIO	1	S	0	233.49.81.16	50133	233.49.81.144	50133
RIO	1	A	0	233.49.81.17	50132	233.49.81.145	50132
RIO	1	B	10	233.49.81.18	50132	233.49.81.146	50132
RWE	1	S	0	233.49.81.8	50133	233.49.81.136	50133
RWE	1	A	0	233.49.81.9	50132	233.49.81.137	50132
RWE	1	B	10	233.49.81.10	50132	233.49.81.138	50132
SAPB	1	S	0	233.49.81.8	50133	233.49.81.136	50133
SAPB	1	A	0	233.49.81.9	50132	233.49.81.137	50132
SAPB	1	B	10	233.49.81.10	50132	233.49.81.138	50132
SAPF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
SAPF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
SAPF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
SET	1	S	0	233.49.81.8	50133	233.49.81.136	50133
SET	1	A	0	233.49.81.9	50132	233.49.81.137	50132
SET	1	B	10	233.49.81.10	50132	233.49.81.138	50132
SGN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
SGN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
SGN	1	B	10	233.49.81.18	50132	233.49.81.146	50132

SIE	1	S	0	233.49.81.8	50133	233.49.81.136	50133
SIE	1	A	0	233.49.81.9	50132	233.49.81.137	50132
SIE	1	B	10	233.49.81.10	50132	233.49.81.138	50132
TNE5	1	S	0	233.49.81.8	50133	233.49.81.136	50133
TNE5	1	A	0	233.49.81.9	50132	233.49.81.137	50132
TNE5	1	B	5	233.49.81.10	50132	233.49.81.138	50132
TNE5	1	C	10	233.49.81.11	50132	233.49.81.139	50132
TQI5	1	S	0	233.49.81.8	50133	233.49.81.136	50133
TQI5	1	A	0	233.49.81.9	50132	233.49.81.137	50132
TQI5	1	B	5	233.49.81.10	50132	233.49.81.138	50132
TQI5	1	C	10	233.49.81.11	50132	233.49.81.139	50132
TUI	1	S	0	233.49.81.8	50133	233.49.81.136	50133
TUI	1	A	0	233.49.81.9	50132	233.49.81.137	50132
TUI	1	B	5	233.49.81.10	50132	233.49.81.138	50132
TUI	1	C	10	233.49.81.11	50132	233.49.81.139	50132
UBSN	1	S	0	233.49.81.16	50133	233.49.81.144	50133
UBSN	1	A	0	233.49.81.17	50132	233.49.81.145	50132
UBSN	1	B	10	233.49.81.18	50132	233.49.81.146	50132
UNI	1	S	0	233.49.81.8	50133	233.49.81.136	50133
UNI	1	A	0	233.49.81.9	50132	233.49.81.137	50132
UNI	1	B	10	233.49.81.10	50132	233.49.81.138	50132
VOD	1	S	0	233.49.81.16	50133	233.49.81.144	50133
VOD	1	A	0	233.49.81.17	50132	233.49.81.145	50132
VOD	1	B	10	233.49.81.18	50132	233.49.81.146	50132
VODF	1	S	0	233.49.81.40	50133	233.49.81.168	50133
VODF	1	A	0	233.49.81.41	50132	233.49.81.169	50132
VODF	1	B	10	233.49.81.42	50132	233.49.81.170	50132
VOW	1	S	0	233.49.81.8	50133	233.49.81.136	50133
VOW	1	A	0	233.49.81.9	50132	233.49.81.137	50132
VOW	1	B	10	233.49.81.10	50132	233.49.81.138	50132
XMT	1	S	0	233.49.81.64	50133	233.49.81.192	50133
XMT	1	A	0	233.49.81.65	50132	233.49.81.193	50132
XMT	1	B	10	233.49.81.66	50132	233.49.81.194	50132
XMTF	1	S	0	233.49.81.64	50133	233.49.81.192	50133
XMTF	1	A	0	233.49.81.65	50132	233.49.81.193	50132
XMTF	1	B	10	233.49.81.66	50132	233.49.81.194	50132

3 Glossary

Term	Description
Acceptor	Computer Application that accepts FAST Session from an Initiator SCP 1.0/1.1
Channel	A Unix Socket Channel of the EBS Receiving Application that contains a specific set of products and a specific range or order book depths. For these products. Each EBS channel has a multicast address and port allowing EBS Client [Member Receiving Applications] to select from a set of predefined market depth subscription packages. A channel is the basic unit of subscription with the EBS servers.
Data Link Layer	The Data Link Layer is the part of the ISO model that resides about the IP Packet and physical message transport layer. Two examples are TCP and UDP.
Delta/Incremental Broadcasts	Delta Broadcasts are provided for a market effecting event (such as order entry or a trade), only changes to the market are broadcast. Members use the information to update their copy of the corresponding order book(s) for contract(s).
EBS	Eurex Broadcast Solutions
Eurex	European Exchange Frankfurt Germany
FAST	FIX Adapted for Streaming.
FAST Session	A bidirectional initiation, protocol negotiation and subsequent exchange of FAST messages via SCP 1.0/1.1 Data Link Control Layer {underlying layer is UDP messages }
FIX	Financial Information Exchange Protocol
FPL	FIX Protocol Limited
Live-Live	Data transmitted in parallel over two physical distinct networks configuration. Used in IP Multicast using unreliable UDP packets. The dual network redundancy allows the EBS Client to compare and reconcile situations of lost UDP message transmission in order to increase reliability.
Initiator	Computer Application that initiates a FAST session with an Acceptor,
IP	Internet Protocol RFC 791
Joining	Electronically connecting to the information source. Once joined to a source, all information is disseminated by the source will broadcast and continue until the source in un-joined (disconnect).
Member/Exchange Participant	A organization authorized by Eurex to receive the services offered through the EBS interface.
Non-Overlapped Mode	In this mode, each channel will carry order book for depth levels above the ones sent in previous channel and for all channels will be dependent on the others eg. Consider FDAX configured with three channels (Eurex Configures channels) Channel 1 : Top of Book Only : Multicast Address 255.1.1.37 Channel 2 :Depths 1-5: 225.1.1.38 Channel 3 :Depths 6-50: 225.1.1.39 A member interested in order book depth up to full 50 levels needs to join all channels.
Order Book (ODB)	A digital book of orders for a tradable unit (contract) available for matching and managed by Eurex.
Overlapping Mode	In this mode, every channel carries a complete order book up to the configured depth level. The channels are independent. Eg. Consider FDAX configured with three

	channels (Eurex Configures channels) Channel 1 : Top of Book Only : Multicast Address 255.1.1.37 Channel 2 :Top Book +Depths 1-5: 225.1.1.38 Channel 3 :Top Book +Depths 1-50: 225.1.1.39 A member interested in order book depts. Of 5 levels needs only join channel two.
Presence Map (PMAP)	The Presence Map is a bit combination indicating the presence or absence of a field in the message body. The allocation of bit for a field in the presence map is governed by the FAST field encoding rules.
Product [Reference Message]	Information related to a product and all contracts of the product are broadcast in product reference information. Should be considered static for a business day.
Reference Information Messages	The Reference Data Stream provides Reference Information messages. These messages provide a description of all available products and their active contracts for which the EBS interface will broadcast price updates. Reference information available from pre-defined multi-cast address. Reference Messages contain information on the granularity of the data offered for each contract. Reference types are 1) Products, 2) Single Leg Contracts 3) Strategies. The product and single leg contract reference information can be different for each business day. Members must receive this information first before connecting to the price delivery channels (Delta Incremental Streams), Reference Data Streams provide reference information for products, single leg contracts, strategies and configuration data. Reference information is disseminated on two interfaces 1) static reference data interface and dynamic reference data interface. Static Reference information remains static for the business day. Dynamic Reference information interface broadcasts strategy contracts defined by the members. The dynamic information changes when members create, change or delete strategy contracts.
SCP	Session Control Protocol 1.0/1.1
Single Leg Contract [Reference Message]	Single Leg contract information broadcast in the form of a single leg-contract messages. The information should be considered static for a business day.
Security and Instrument	Individual tradable component, corresponds to contract, series, combinations or strategies in Eurex.; Implies Eurex products correspond to group of securities
Snapshot Broadcasts	A broadcast that contains all current market information for a given entity, regardless of recent movements and will be sent on a different multicast address to the “real time” Delta Broadcasts A Snapshot will be sent infrequently and members should join the stream long enough to receive a snapshot enabling them to establish a baseline market picture. Upon completion of establishing the baseline market picture, Members should leave the snapshot broadcast stream and rely on delta broadcasts to maintain market picture. Snapshots contain complete order book information up to the depth indicated in the reference information. The snapshot message should be used only for the creation of the market depth at the beginning of a trading day and for its recovery in case of data loss. Snapshots provide information about recent contract status, details on last trade and the daily statistical information for a contract.
Strategies [Reference Message]	Strategy information deals with all options strategies created, modified or deleted by members. In addition, future combinations defined by Eurex are broadcast in the form of a strategy reference information. Request, orders, and quotes can be entered for active strategies. Strategies can consist of up to four individual legs which correspond to contracts defined by the exchange Single Leg Contracts.

Stream	A Collection of One or more channels for a specific set of financial products; A Collection of specific set of message types that are sent in a specific way with the specific semantic protocol. Eurex supports the following streams: 1) Reference Information Streams, 2) Snapshot Streams , 3) Delta Streams . One or more tradable products on Eurex can be grouped to form a logical group. Data pertaining to all products and all contracts of such a group will be disseminated over one set of multicast addresses constitute a stream.
Subscription	Process of Joining an IP Multicast address, receiving UDP datagram packets sent to the joined address [data feed]
TBD	To Be Determined
Template Identifier (TID)	The Template ID is a Uint32 field to inform the template to be used for decoding the message.
Trade Information	Messages that carry information of all trades executed on the exchange Information is disseminated for single leg contracts only. Trade Statistics are provide with this message
UDP	User Datagram Protocol RFC 768. This is a data link protocol like TCP however this facility does not provide for checking, and retransmission control in cases of lost UDP message packets.
VALUES API	Virtual Access Link Using Exchange Services Application Programming Interfaces

4 References

[1] Eurex Enhanced Broadcast Solution Quick Start Guide www.eurexexchange.com

[2] Eurex Technology Roadmap; Introduction of the Enhanced Broadcast Solution in the Eurex 10.0 Simulation

[3] Environment;Minor Changes to the Eurex New Socket Data Feed Sept 25, 2007
http://www.eurexchange.com/download/documents/publications/eurex_infomail_e.pdf

[4] Eurex BroadCast Solution (EBS) Usage Examples www.eurexexchange.com

[5] RFC 768 User Datagram Protocol
http://www.faqs.org/ftp/rfc/pdf/rfc768.txt.pdf

[6] RFC 791 Internet Protocol
http://www.faqs.org/ftp/rfc/pdf/rfc791.txt.pdf

[7] FIX Adapted for Streaming (FAST Protocol) Session Control Protocol Version 1.00 2006-05-10
http://www.fixprotocol.org/fast

[8] FIX Protocol – A Basic User's Guide to Implementing The Fast Protocol Version 1.0 January 2006
http://www.fixprotocol.org/documents/3066/FAST%20Specification%201%20x%201.pdf

[9] FIX Protocol FAST Specification Version 1.x.1 2006-12-20 http://www.fixprotocol.org/fast

[10] Eurex Broadcast Solution (EBS) Usage Examples www.eurexechange.com

[11] Eurex Technology Roadmap: Enhanced Broadcast Solution Eurex Circular 1115/07 Frankfurt, Jun 8, 2007
http://www.eurexexchange.com

[12] FAST Session Control Protocol Draft Version 1.1 2007-08-27 http://www.fixprotocol.org/fast

[13] UNIX Network Programming, W. Richard Stevens. ISBN 0-13-949876-1

[14] FIX Adapted for Streaming (FAST Protocol) Session Control Protocol Version 1.00 2006-05-10
http://www.fixprotocol.org/fast

[15] FAST ™ Specification Version 1.x.1 2006-12-20 http://www.fixprotocol.org/fast

[16] Financial Information Exchange Protocol (FIX) Version 1.1 December 2006
http://www.fixprotocol.org/fast

[17] FIX Market Data Over a Multicast Transport Recommended Practices http://www.fixprotocol.org/fast

[18] FIX Protocol Limited Market Data Working Group, Recommended Practices for Book Management Version 2.00 January 2007 http://www.fixprotocol.org/documents/2518/MDOWG_Book_Mgt%20v20.doc

[19] FIX Functionality Matrix http://www.fixprotocol.org/fast
FIX Functionality Matrix.pdf

[20] Values API Member Front End Development Guide Volume 2- Eurex Application Requests Programming Version Eurex Release 10.0 Final Version www.eurexexchange.com

[21] VALUES API Member Front End Development Guide Volume-4 CCP Application Requests and Responses Final Version CCP Release 4.0 www.eurexexchagne.com

[22] VALUES API Member Front End Development Guide Volume 2 Eurex Application Requests Final Version, Eurex Release 9.0 www.eurexexchange.com

[23] Eurex – Enhanced Broadcast Solution Interface Specification – (Final Version) EUREX® Release 10.0 www.eurexexchange.com

[24] Streams Programming Guide Sun Micro Systems Part No 816-4855-10, Jan 2005 http://dlc.sun.com/pdf/816-4855/816-4855.pdf

[25] Multithreading in the Solaris Operating Environment – Technical White Paper http://www.sun.com/software/whitepapers/solaris9/multithread.pdf

[26] Posix Threads Programming https://computing.llnl.gov/tutorials/pthreads/