BLUEOCEAN
ATS

MEMOIR Top of Book Feed
v1.3

Table of Contents

1 Overview 5

2 Architecture 6

3 Encoding 7
3.1 Data Types 7
3.1.1 PriceType 7
3.1.2 ShortPriceType 7
3.1.3 String 8
3.1.4 BooleanType 8
3.1.5 UTC Timestamp Nanos 8
3.2 Header 8

4 Message Field Types 10
4.1 SecurityTradingStatusType 10
4.2 SecurityTradingStatusReasonType 10
4.3 TradingSessionType 10

5 Messages 11
5.1 Instrument Directory 11
5.2 Reg SHO Restriction 11
5.3 Security Trading Status 12
5.4 Trading Session Status 13
5.5 Best Bid Offer 13
5.6 Best Bid 13
5.7 Best Offer 14
5.8 Best Bid Short 14
5.9 Best Offer Short 14
5.10 Clear Book 14
5.11 Snapshot Complete 15

6 Message/State Recovery Methods 16
6.1 Gap Fill 16
6.2 Snapshot 17

7 Example Messages 19
7.1 Instrument Directory Message 20
7.2 Reg Sho Restriction Message 20
7.3 Security Trading Status Message 20
7.4 Best Bid Offer Message 20
7.5 Best Bid Message 21
7.6 Best Offer Message 21
7.7 Best Bid Short Message 21
7.8 Best Offer Short Message 21
7.9 Clear Book 22
7.10 Snapshot Complete Message 22

< THIS PAGE INTENTIONALLY LEFT BLANK>

1 Overview

MEMOIR Top of Book Feed is a real-time top of book feed.

MEMOIR Top of Book Feed will provide an event based binary messaging protocol that will support the following types of messages:

- **Trading Session Information** - Provides the state of the trading session.
- **Top of Book** - Provides the best bid and best offer on the exchange in efficient formats.
- **Instrument Directory** - Provides a mapping of all supported security IDs to all their basic attributes.
- **Trading Actions** - Supplies trading status messages which inform participants of events for an instrument.
- **Reg SHO Restriction** - Supplies updates to the Reg SHO short sale restriction status of an instrument

MEMOIR Top of Book Feed is unidirectional. It cannot be used to enter orders.

Application messages are implemented using a binary protocol based on SBE (Simple Binary Encoding).

2 Architecture

The MEMOIR Top of Book Feed is a sequenced message stream using fixed width binary messages.

The business level messaging is transported via the following protocols for framing and recovery:

- MEMX-UDP protocol - UDP Multicast-based session transport for the real-time delivery of messages.
- MEMX-TCP protocol - TCP-based transport used for gap fill and snapshot recovery of state (see Message/State Recovery modes later in this document for more info)

At the start of the day a session ID will be provided via the MEMX-UDP channel, and messages will begin at sequence number 1. An Instrument Directory message spin will be sent out to map all the traded symbols to a symbol locate code that will be used throughout the session. The locate code will not change during the course of the session, however intraday there may be subsequent updates to the same or new instruments.

3 Encoding

The MEMOIR Top of Book Feed uses the FIX Trading Community's Simple Binary Encoding (SBE) to specify message encoding. More information about SBE can be found at the FIX SBE XML Primer.

The MEMOIR Top of Book Feed is always encoded in Big Endian byte order.

3.1 Data Types

All encoding and decoding for SBE is centered around a set of basic primitive types.

For more information on primitive type encoding, see the SBE specification.

Type	Length (bytes)	Description	Value Range	Null Value	Null Value (Hex)
CHAR	1	ASCII Character	`0` (`NUL`) to `127` (`DEL`)	`0`	`0x00`
INT8	1	Signed Integer	`-127` to `127`	`-128`	`0x80`
INT16	2	Signed Integer	`-32767` to `32767`	`-32768`	`0x8000`
INT32	4	Signed Integer	`-2^31 + 1` to `2^31 - 1`	`-2^31` (`-2147483648`)	`0x80000000`
INT64	8	Signed Integer	`-2^63 + 1` to `2^63 - 1`	`-2^63`	`0x8000000000000000`
UINT8	1	Unsigned Integer	`0` to `254`	`255`	`0xFF`
UINT16	2	Unsigned Integer	`0` to `65534`	`65535`	`0xFFFF`
UINT32	4	Unsigned Integer	`0` to `2^32-2`	`2^32 - 1` (`4294967295`)	`0xFFFFFFFF`
UINT64	8	Unsigned Integer	`0` to `2^64 - 2`	`2^64 - 1`	`0xFFFFFFFFFFFFFFFF`

The MEMOIR specification does not use the SBE floating point data types.

3.1.1 PriceType

Prices are encoded as a fixed-point scaled decimal, consisting of a signed long (int64) mantissa and a constant exponent of -6.

Type Name	Length	Type	Description
Mantissa	8	INT64	The fixed-point decimal representation of the price.
Exponent	N/A	INT8	MEMOIR uses a constant exponent of -6 for all prices.

For example, a mantissa of 1234567 with the constant exponent of -6 represents the decimal number 1.234567 and would appear encoded on the wire as the hex value `000000000012D687`.

3.1.2 ShortPriceType

To conserve bytes on the wire, some prices are encoded as a fixed-point scaled decimal, consisting of a signed short (int16) mantissa and a constant exponent of -2.

Type Name	Length	Type	Description
Mantissa	2	INT16	The fixed-point decimal representation of the price.
Exponent	N/A	INT8	MEMOIR uses a constant exponent of -2 for all short prices.

For example, a mantissa of 12345 with the constant exponent of -2 represents the decimal number 123.45, and would appear encoded on the wire as the hex value `00003039`

3.1.3 String

Strings are fixed length ASCII based character (CHAR) sequences. The length will be defined in the schema. Variable length fields are not supported.

3.1.4 BooleanType

Boolean values are not defined specifically in SBE. The schema defines a BooleanType which represents a numeric value with True = `1` and False = `0`

3.1.5 UTC Timestamp Nanos

Fields with the `UTCTimestampNanos` type represents a timestamp in Coordinated Universal Time (UTC) which begins at the UNIX epoch (January 1, 1970 at 00:00:00 UTC).

UTCTimestampNanos has the time unit of nanoseconds and is encoded as follows:

Type Name	Length	Type	Description
Time	8	UINT64	UTC timestamp since unix epoch with nanosecond precision.
Unit	N/A	UINT8	Unit of time. UTCTimestampNanos are represented in nanoseconds. This field is constant (value=`9`) and as such will not be transferred on the wire.

3.2 Header

SBE includes a header for each message. The SBE header is followed by the SBE body for the message.

The common SBE message header contains the fields that allows the decoder to identify what codec should be used as the template for a message.

1. **blockLength**: The number of bytes in the message body (does not include the header bytes).
2. **templateID**: The identifier for the template type of the message that is to follow.
3. **schemaID**: The identifier for the schema the message belongs to.
4. **version**: The version of the schema allowing for extension. Two pieces of information are packed into the (UINT16) version field, a major version and a minor update version. So for example a version of 258 = 0x0102 indicates major version 1, minor update 2.

The MEMOIR SBE header appears on the wire as:

Field	Offset	Length	Type	Description
BlockLength	0	2	UINT16	The number of bytes in the message body (does not include the header bytes). Note that MEMOIR messages do not use repeating groups or variable-length fields.

Field	Offset	Length	Type	Description
TemplateID	2	1	UINT8	Identifier of the message template. (ie. the message type.)
SchemaID	3	1	UINT8	The identifier of a message schema. SchemaId=3 for MEMOIR Top of Book.
Version	4	2	UINT16	The version number of the message schema that was used to encode a message. Two pieces of information are packed into the (UINT16) version field, a major version and a minor update version. So for example a version of 258 = 0x0102 indicates major version 1, minor update 2.

4 Message Field Types

All messages are composed of fields. Each field has a type. This section defines the field types, their underlying on the wire type, acceptable values and a description of the type.

4.1 SecurityTradingStatusType

SecurityTradingStatusType represents the current security trading state based upon the listing market data off of the SIP (Securities Information Processor).

SecurityTradingStatusType is a 1-byte CHAR value.

Value	Name
H	Halted
P	Paused
Q	Quoting
T	Trading

Messages that contain a field of SecurityTradingStatusType include: Security Trading Status

4.2 SecurityTradingStatusReasonType

SecurityTradingStatusReasonType represents the reason for this security status.

SecurityTradingStatusReasonType is a 1-byte CHAR value.

Value	Name	Description
X	None	The security trading status does not apply (e.g the security is trading normally).
R	Regulatory	The security trading status originated from a regulator or the primary listing exchange for the security and was received via the SIP.
A	Administrative	The security trading status change originated from the exchange.

Messages that contain a field of SecurityTradingStatusType include: Security Trading Status

4.3 TradingSessionType

TradingSessionType represents the current trading session.

TradingSessionType is a 1-byte CHAR value.

Value	Name	Description
1	Opening	Pre-Market session
2	Trading	Market session
3	PostTrading	Post-Market session
4	Closed	Market closed

Messages that contain a field of TradingSessionType include: Trading Session Status

5 Messages

This section defines the messages that make up the protocol. For each message, it lists the fields in the message, as well as each field's position and length in the message, its underlying type, and a description of its purpose.

5.1 Instrument Directory

At the start of the trading day, an instrument directory message will be sent for all tradable securities on the exchange. In order to reduce the message sizes and improve performance, a SecurityID will be used in place of a ticker symbol on subsequent messages. SecurityIDs will start at 1 and increment. Do not rely on SecurityIDs being the same in subsequent trading days. The exchange uses CMS as its trading symbology, which requires a root and suffix combination.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=1,BlockLength=35
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	A unique code identifying the instrument for any messages in this session.
Symbol	16	6	String	The CMS root for the tradable instrument.
SymbolSfx	22	6	String	The CMS suffix for the tradable instrument.
RoundLot	28	4	UINT32	The number of shares in a round lot for the instrument.
IsTestSymbol	32	1	BooleanType	Determines if this security is a test instrument.
MPV	33	8	PriceType	The minimum price variation for an instrument. (the smallest price increment of the stock)

5.2 Reg SHO Restriction

In 2010, the Securities and Exchange Commission adopted Rule 201 of Regulation SHO. Rule 201 restricts the price at which short sales may occur when a stock has experienced significant downward price pressure. Rule 201 is designed to prevent short selling, including potentially manipulative or abusive short selling, from driving down further the price of a security that has already experienced a significant intra-day price decline, and to facilitate the ability of long sellers to sell first upon such a decline.

This message is used to indicate when a specific security is subject to the short sale price restriction. If this message is sent and the ShortSaleRestriction flag is set to true, Rule 201 is now in effect for that security ID.

NOTE: This message may be sent immediately after the related `Instrument Directory` message is sent for the security if the restriction is already in effect at the start of the MEMOIR data session.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=2, BlockLength=11
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
ShortSaleRestriction	16	1	BooleanType	Identifies if this security is subject to the Regulation SHO short sales restrictions.

5.3 Security Trading Status

This message will inform the client of the current trading status of a security on the exchange. The trading status of a security on the exchange may change due to the following reasons:

- The primary market for a security has the authority to halt or pause a security via the SIP based upon upcoming news dissemination, or for regulatory purposes such as LULD threshold violations.
- The exchange may internally halt trading for a security for administrative or operational reasons.

Security Trading Status messages are sent out after Instrument Directory messages have been sent.

This message can and will be sent out throughout the trading session to indicate realtime changes in the security state.

If a `SecurityTradingStatus` message is not received for a security, it should be assumed that the status is `Halted`.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=3, BlockLength=12
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
SecurityTradingStatus	16	1	SecurityTradingStatusType	The current trading state.

Field	Offset	Length	Type	Meaning
SecurityTradingStatusReason	17	1	SecurityTradingStatusReasonType	The source of the trading status change.

5.4 Trading Session Status

The trading system has entered a new trading session.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=5, BlockLength=9
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
TradingSession	14	1	TradingSessionType	The trading session which was entered.

5.5 Best Bid Offer

Provides an entire top of book update for a particular security during the snapshot procedure. Note that this message will never be received when processing an incremental feed.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=10, BlockLength=34
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
BidSize	16	4	UINT32	The best bid size.
BidPrice	20	8	PriceType	The best bid price.
OfferSize	28	4	UINT32	The best offer size.
OfferPrice	32	8	PriceType	The best offer price.

5.6 Best Bid

In order to reduce bandwidth, if only the bid (buy) side has changed for a particular security, this message will be sent.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=11, BlockLength=22
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
BidSize	16	4	UINT32	The best bid size.
BidPrice	20	8	PriceType	The best bid price.

5.7 Best Offer

In order to reduce bandwidth, if only the offer (sell) side has changed for a particular security, this message will be sent.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=12, BlockLength=22
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
OfferSize	16	4	UINT32	The best offer size.
OfferPrice	20	8	PriceType	The best offer price.

5.8 Best Bid Short

In order to reduce bandwidth, if only the bid (buy) side has changed for a particular security and the price <= $327.67 and size < 65535 this message will be sent.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=13, BlockLength=14
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
BidSize	16	2	UINT16	The best bid size.
BidPrice	18	2	ShortPriceType	The best bid price.

5.9 Best Offer Short

In order to reduce bandwidth, if only the offer (sell) side has changed for a particular security and the price <= $327.67 and size < 65535 this message will be sent.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=14, BlockLength=14
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.
OfferSize	16	2	UINT16	The best offer size.
OfferPrice	18	2	ShortPriceType	The best offer price.

5.10 Clear Book

An operational or regulatory action has resulted in the book being cleared.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=15, BlockLength=10
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from the Instrument Directory.

5.11 Snapshot Complete

End of the snapshot message, all messages have been sent. Used only in Snapshot recovery (see Message/State Recovery Methods below)

Upon receipt of this message, clients should disconnect from the Top of Book Snapshot feed and follow the recovery procedure to reconcile their state against the incremental Top of Book feed.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=3, TemplateId=4, BlockLength=16
Timestamp	6	8	UTCTimestampNanos	The timestamp when the event occurred.
AsOfSequenceNumber	14	8	UINT64	The sequence number on the real-time multicast channel that this snapshot is based upon. All messages buffered from the real-time feed that are less than or equal to this number should be discarded. All messages buffered after this number should be applied. (with appropriate gaps filled normally).

6 Message/State Recovery Methods

As MEMOIR Top of Book is disseminated primarily over the MEMX-UDP transport protocol, consumers of the feed may periodically miss messages due to the unreliable nature of the UDP protocol. Two independent Multicast groups will be provided for standard A/B arbitration and quick recovery of messages dropped due to packet loss on a single channel. However, there are cases, such as a catastrophic system issue on the client's end, or the loss of the same message from both channels, where the client may need to request data to be re-sent from the exchange. For these situations,two modes of recovery are available for users. The first is a "gap fill" mechanism, used to recover small numbers of messages, typically for small episodes of packet loss that affect the same messages on both Multicast channels. The second is a "snapshot" mechanism, where the current business state of the feed is sent, followed by a "live" sequence number for the client to begin processing the real-time MEMX-UDP channels.

6.1 Gap Fill

A client detects missing messages over the MEMX-UDP transport for MEMOIR Top of Book by using the sequence number in the MEMX-UDP datagram header to determine if a gap in sequence numbers has occurred. In order to recover these messages, the client may use a connection to a Gap Fill Server via the MEMX-TCP Replay mode to request the missing messages.

NOTE: The client may initiate and maintain a connection to the MEMX-TCP Gap Fill Server early (to verify connectivity) and stay connected (while periodically sending heartbeats) until the functionality is needed.

The process by which a client recovers missing messages via the Gap Fill Server is as follows:

1. Issue a MEMX-TCP `ReplayRequest` to the MEMOIR Top of Book Gap Fill Server, containing the `Session ID` received over the MEMX-UDP channel(s), the `NextSequenceNumber` of the first missing message, and the `Count` of messages required (including the sequence provided)
2. While the above request is being processed, the client should buffer any received MEMX-UDP multicast messages and not apply them until the missing sequence numbers have been recovered.
3. After the receipt of the MEMX-TCP Replay Request, the Gap Fill Server will send the following:
    a. A MEMX-TCP `ReplayBegin` message, with `NextSequenceNumber` set to the requested Sequence number, and a `PendingMessageCount` set to the number of messages to follow in the replay. (This may be less than the requested count - see NOTE below)
    b. The requested number of **MEMOIR Top of Book** messages (or less given the constraints listed above on request size), starting at the requested sequence number
    c. A MEMX-TCP `ReplayComplete` message
4. After the `ReplayComplete` message is received, the client is free to send another request (if this request did not satisfy the entire gap)
5. Once the client has received all of the missing messages from one or more replay requests to the MEMX-TCP Replay server, the client may then process the messages buffered from the MEMX-UDP channel, discarding duplicate sequence numbers and applying future sequence numbers in order.

NOTE: the replay provided may consist of fewer messages than requested due to some constraints on the size of the response. The number of messages in the replay response will be the minimum of:

- the requested count
- the configured maximum messages per request for the service
- the number of messages remaining from the requested start sequence to the highest published sequence

6.2 Snapshot

If a catastrophic failure occurs on the client side that requires a full state reset on the client, and the client does not wish to manually replay the entire day so far via multiple MEMX-TCP Replay requests, the client may opt to use the MEMX-TCP snapshot mechanism to recover state. Once the snapshot has completed, the client should use normal Gap Fill processing (described above) to recover any further gaps in data that may occur during that trading session, as Snapshots can be quite large compared to a gap of one or two datagrams.

NOTE: The client may initiate and maintain a connection to the MEMX-TCP snapshot server early (to verify connectivity) and stay connected (while periodically sending heartbeats) until the functionality is needed.

The process by which a client recovers via snapshot is as follows:

1. Join the real-time MEMX-UDP multicast group(s) for the data feed, and begin buffering all received messages.
2. Issue a MEMX-TCP `ReplayAllRequest` to the MEMOIR Top of Book snapshot server, containing the `SessionID` received over the MEMX-UDP channel(s)
3. After the receipt of the MEMX-TCP ReplayAll request, the Snapshot Server will send the following:
    a. A MEMX-TCP `ReplayBegin` message, with `NextSequenceNumber` set to 1 and `PendingMessageCount` set to the number of MEMOIR Top of Book Business Level messages contained in the snapshot
    b. MEMOIR Top of Book Business Level `InstrumentDirectory`messages containing a symbol description and associated SecurityID to be used throughout the session
    c. MEMOIR Top of Book Business Level `RegShoRestriction` messages containing the current Reg-SHO Restriction State for each symbol
    d. MEMOIR Top of Book Business Level `SecurityTradingStatus` messages containing the current trading status for each symbol
    e. MEMOIR Top of Book Business Level `TradingSessionStatus` message containing the current trading session status.
    f. MEMOIR Top of Book Business Level 1 or 2-sided book updates `BestBidOffer`, `BestBid`, `BestOffer`, `BestBidShort`, or `BestOfferShort` messages for each symbol
    g. A single MEMOIR Top of Book Business Level `SnapshotComplete` event containing an `AsOfSequenceNumber` set to the sequence number on the MEMX-UDP multicast channel that this snapshot was taken (clients should resume processing at this number + 1 on the multicast feed after applying the snapshot)

   h. A MEMX-TCP `ReplayComplete` message

4. Discard all buffered events from the MEMX-UDP real-time feed at or before the `AsOfSequenceNumber` in the received Snapshot Complete event.
5. Apply all buffered events from the MEMX-UDP real-time feed after the `AsOfSequenceNumber` in the received Snapshot Complete event, in sequence order.
6. Continue to process the MEMX-UDP real-time multicast feed normally from this point onward, using the **Gap Fill** mechanism above to recover dropped messages.

7 Example Messages

Note: the code fragments shown are representative of how to set field values in the SBE messages. Your interface may be be different.

7.1 Instrument Directory Message

```
instrumentDirectoryEncoder.timestamp().time(getTimeInNanos());
instrumentDirectoryEncoder.securityID(0xABCD);
instrumentDirectoryEncoder.symbol("AAPL");
instrumentDirectoryEncoder.roundLot(100);
instrumentDirectoryEncoder.isTestSymbol(BooleanType.False);
instrumentDirectoryEncoder.mPV().mantissa(10000);

Hex code:
00000000: 00230103 00010005 e23d3666 701cabcd   |.#.......=6fp...|
00000010: 4141504c 00000000 00000000 00000064   |AAPL...........d|
00000020: 00000000 00000027 10                  |.......'.|
```

7.2 Reg Sho Restriction Message

```
regSHORestrictionEncoder.timestamp().time(getTimeInNanos());
regSHORestrictionEncoder.securityID(0xABCD);
regSHORestrictionEncoder.shortSaleRestriction(BooleanType.True);

Hex code:
00000000: 000b0203 00010005 e25524ac 5c64abcd   |.........U$.\d..|
00000010: 01                                    |.|
```

7.3 Security Trading Status Message

```
securityTradingStatusEncoder.timestamp().time(getTimeInNanos());
securityTradingStatusEncoder.securityID(0xABCD);
securityTradingStatusEncoder.securityTradingStatus(SecurityTradingStatusType.
Quoting);
securityTradingStatusEncoder.securityTradingStatusReason(SecurityTradingStatu
sReasonType.None);

Hex code:
00000000: 000c0303 00010005 e25524b9 e801abcd   |.........U$.....|
00000010: 5158                                  |QX|
```

7.4 Best Bid Offer Message

```
bestBidOfferEncoder.timestamp().time(getTimeInNanos());
bestBidOfferEncoder.securityID(0xABCD);
bestBidOfferEncoder.bidSize(8600);
bestBidOfferEncoder.bidPrice().mantissa(123450000);
bestBidOfferEncoder.offerSize(19800);
bestBidOfferEncoder.offerPrice().mantissa(123470000);

Hex code:
00000000: 00220a03 00010005 e2552510 d705abcd   |."........U%.....|
00000010: 00002198 00000000 075bb290 00004d58   |..!.......[....MX|
00000020: 00000000 075c00b0                     |.....\..|
```

7.5 Best Bid Message

```
bestBidEncoder.timestamp().time(getTimeInNanos());
bestBidEncoder.securityID(0xABCD);
bestBidEncoder.bidSize(865000);
bestBidEncoder.bidPrice().mantissa(123450000);

Hex code:
00000000: 00160b03 00010005 e25524e0 b495abcd   |.........U$.....|
00000010: 000d32e8 00000000 075bb290            |..2.......[..|
```

7.6 Best Offer Message

```
bestOfferEncoder.timestamp().time(getTimeInNanos());
bestOfferEncoder.securityID(0xABCD);
bestOfferEncoder.offerSize(19800);
bestOfferEncoder.offerPrice().mantissa(123450000);

Hex code:
00000000: 00160c03 00010005 e255251e 6218abcd   |.........U%.b...|
00000010: 00004d58 00000000 075bb290            |..MX.....[..|
```

7.7 Best Bid Short Message

```
bestBidShortEncoder.timestamp().time(getTimeInNanos());
bestBidShortEncoder.securityID(0xABCD);
bestBidShortEncoder.bidSize(7600);
bestBidShortEncoder.bidPrice().mantissa((short) 1234);

Hex code:
00000000: 000e0d03 00010005 e25524ff 72e9abcd   |.........U$.r...|
00000010: 1db004d2                              |....|
```

7.8 Best Offer Short Message

```
bestOfferShortEncoder.timestamp().time(getTimeInNanos());
bestOfferShortEncoder.securityID(0xABCD);
bestOfferShortEncoder.offerSize(19800);
bestOfferShortEncoder.offerPrice().mantissa((short) 1234);

Hex code:
00000000: 000e0e03 00010005 e255252b 5f31abcd   |.........U%+_1..|
00000010: 4d5804d2                              |MX..|
```

7.9 Clear Book

```
clearBookEncoder.timestamp().time(getTimeInNanos());
clearBookEncoder.securityID(0xABCD);

Hex code:
00000000: 000a0f03 00010005 e2552537 a3a1abcd   |.........U%7....|
```

7.10 Snapshot Complete Message

```
snapshotCompleteEncoder.timestamp().time(getTimeInNanos());
snapshotCompleteEncoder.asOfSequenceNumber(0x11223344L);

Hex code:
00000000: 00100403 00010005 e2552543 ec4c0000   |.........U%C.L..|
00000010: 00001122 3344                         |..."3D|
```