BLUEOCEAN
ATS

MEMOIR Last Sale Feed
v1.3

This specification is being provided to you strictly for informational purposes solely for the purpose of developing or operating systems that interact with BOATS. All proprietary rights and interest in this specification and the information contained herein shall be vested in BOATS and all other rights including, but without limitation, patent, registered design, copyright, trademark, service mark, connected with this publication shall also be vested in BOATS. No part of this specification may be redistributed or reproduced in any form or by any means or used to make any derivative work (such as translation, transformation, or adaptation) without written permission from BOATS. BOATS reserves the right to withdraw, modify, or replace the specification at any time, without notice. No obligation is made by BOATS regarding the level, scope, or timing of BOATS's implementation of the functions or features discussed in this specification.

THE SPECIFICATION IS PROVIDED "AS IS", "WITH ALL FAULTS" AND BOATS MAKES NO WARRANTIES AND DISCLAIMS ALL WARRANTIES, EXPRESSED OR IMPLIED, OR STATUTORY RELATED TO THE SPECIFICATIONS. BOATS IS NOT LIABLE FOR ANY INCOMPLETENESS OR INACCURACIES IN THE SPECIFICATIONS. BOATS IS NOT LIABLE FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES RELATING TO THE SPECIFICATIONS OR THEIR USE.

Table of Contents

< THIS PAGE INTENTIONALLY LEFT BLANK>

1 Overview

MEMOIR Last Sale Feed is a real-time trade feed.

The MEMOIR Last Sale Feed provides an event based binary messaging protocol that will support the following functionality:

- **Trading Session Information** - Provides the state of the trading session.
- **Trade Data** - Provides trade reporting, trade cancellation and trade correction for executions on the exchange.
- **Instrument Directory** - Provides a mapping of all supported security IDs to all their basic attributes.
- **Trading Actions** - Supplies trading status messages to inform participants of events for an instrument.

The MEMOIR Last Sale Feed is unidirectional. It cannot be used to enter orders.

2 Architecture

The MEMOIR Last Sale 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 gap fill replay service.

At the start of the day the session ID will be provided, and messages 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.

Given that this feed needs to be read in from its entirety, there is no snapshot service available.

3 Encoding

The MEMOIR Last Sale 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 Last Sale 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 String

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

3.1.3 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`.

3.1.4 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 message do not use repeating groups or variable-length fields.
TemplateID	2	1	UINT8	Identifier of the message template. (ie. the message type)
SchemaID	3	1	UINT8	The identifier of a message schema. SchemaId=`4` for MEMOIR Last Sale.
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 exchange.

Messages that contain a field of SecurityTradingStatusReasonType 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

4.4 SCSettlementType

SCSettlementType represents Level-1 sale condition as transmitted by the UTP (Unlisted Trading Privileges) and CTS (Consolidated Trading System).

SCSettlementType is a 1-byte CHAR value.

Value	Name
@	Regular Trade

Messages that contain a field of SCSettlementType include: Trade Report, Trade Cancel, Trade Correct

4.5 SCTradeThroughExemptionType

SCTradeThroughExemptionType represents the exemption rule which allowed a trade-through.

SCTradeThroughExemptionType is a 1-byte CHAR value.

Value	Name
F	Intermarket Sweep
Space ()	Not Applicable

Messages that contain a field of SCTradeThroughExemptionType include: Trade Report, Trade Cancel, Trade Correct

4.6 SCExtendedHoursOrSoldType

SCExtendedHoursOrSoldType represents the extended trading hour sale condition from the CTS.

SCExtendedHoursOrSoldType is a 1-byte CHAR value.

Value	Name
T	Form-T
Space ()	Not applicable

Messages that contain a field of SCExtendedHoursOrSoldType include: Trade Report, Trade Cancel, Trade Correct

4.7 SCSROTradeDetailType

SCSROTradeDetailType represents the self-regulatory organization trade reporting category.

By default, this value will be `Not Applicable`.

SCSROTradeDetailType is a 1-byte CHAR value.

Value	Name
H	Price Variation Trade
I	Odd Lot Trade
X	Cross Trade
Space ()	Not Applicable

Messages that contain a field of SCSROTradeDetailType include: Trade Report, Trade Cancel, Trade Correct

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=4, 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=4, 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=4, 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=4, 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 Trade Report

An order on the book was executed in part or whole, this message will report that execution with proper sale conditions.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=4, 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.
TradeID	16	8	UINT64	The unique identifier of the execution message.
TradeQty	24	4	UINT32	The quantity of the trade.
LastPrice	28	8	PriceType	The price of the trade
SaleCondition1	36	1	SCSettlementType	The sale condition code level 1 - Settlement Type.
SaleCondition2	37	1	SCTradeThroughExemptionType	The sale condition code level 2 - Trade Through Exemption Type.
SaleCondition3	38	1	SCExtendedHoursOrSoldType	The sale condition code level 3 - Extended Hours Type.
SaleCondition4	39	1	SCSROTradeDetailType	The sale condition code level 4 - SRO Trade Detail Type.

5.6 Trade Cancel

This message is sent when the exchange busts an execution. A execution can be canceled if it was determined that it was clearly erroneous. Trades once broken can not be reinstated..

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=4, TemplateId 11, 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
TradeID	16	8	UINT64	The unique identifier of the execution message.
TradeQty	24	4	UINT32	The quantity of the trade.
LastPrice	28	8	PriceType	The price of the trade.
SaleCondition1	36	1	SCSettlementType	The sale condition code level 1 - Settlement Type.
SaleCondition2	37	1	SCTradeThroughExemptionType	The sale condition code level 2 - Trade Through Exemption Type.
SaleCondition3	38	1	SCExtendedHoursOrSoldType	The sale condition code level 3 - Extended Hours Type.
SaleCondition4	39	1	SCSROTradeDetailType	The sale condition code level 4 - SRO Trade Detail Type.

5.7 Trade Correct

Based upon trade cancellation or correction policy, there will be a need to alter the details of an existing trade. This may involve correcting various trade details such as the price.

Field	Offset	Length	Type	Meaning
SBE Header	0	6	N/A	SBE Header with SchemaId=4, TemplateId=12, BlockLength=50
Timestamp	6	8	UTCTimeStampNanos	The timestamp when the event occurred.
SecurityID	14	2	UINT16	An instrument code uniquely identifying the security from

Field	Offset	Length	Type	Meaning
				the Instrument Directory.
TradeID	16	8	UINT64	The unique identifier of the execution message.
OriginalTradeQty	24	4	UINT32	The original quantity of the trade.
OriginalTradePrice	28	8	PriceType	The original price of the trade.
OriginalSaleCondition1	36	1	SCSettlementType	The original sale condition code level 1 - Settlement Type.
OriginalSaleCondition2	37	1	SCTradeThroughExemptionType	The original sale condition code level 2 - Trade Through Exemption Type.
OriginalSaleCondition3	38	1	SCExtendedHoursOrSoldType	The original sale condition code level 3 - Extended Hours Type.
OriginalSaleCondition4	39	1	SCSROTradeDetailType	The original sale condition code level 4 - SRO Trade Detail Type.
CorrectedTradeQty	40	4	UINT32	The corrected trade quantity.
CorrectedTradePrice	44	8	PriceType	The corrected trade price.
CorrectedSaleCondition1	52	1	SCSettlementType	The corrected sale condition code level 1 - Settlement Type.
CorrectedSaleCondition2	53	1	SCTradeThroughExemptionType	The corrected sale condition code level 2 - Trade Through Exemption Type.
CorrectedSaleCondition3	54	1	SCExtendedHoursOrSoldType	The corrected sale condition code level 3 - Extended Hours Type.

Field	Offset	Length	Type	Meaning
CorrectedSaleCondition4	55	1	SCSROTradeDetailType	The corrected sale condition code level 4 - SRO Trade Detail Type.

6 Message/State Recovery Methods

As MEMOIR Last Sale 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, a "gap fill" mechanism is provided for the client to recover lost messages.

6.1 Gap Fill

A client detects missing messages over the MEMX-UDP transport for MEMOIR Last Sale 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 Last Sale Gap Fill Server, containing the `SessionID` 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 Last Sale** 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

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: 00230104 00010005 e2c60a7f 5972abcd   |.#..........Yr..|
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: 000b0204 00010005 e2c60d18 6084abcd   |............`...|
00000010: 01                                    |.|
```

7.3 Security Trading Status Message

```
securityTradingStatusEncoder.timestamp().time(getTimeInNanos());
securityTradingStatusEncoder.securityID(0xABCD);
securityTradingStatusEncoder.securityTradingStatus(SecurityTradingStatusType.
Quoting);
securityTradingStatusEncoder.securityTradingStatusReason(SecurityTradingStatu
sReasonType.Administrative);

Hex code:
00000000: 000c0304 00010005 e2c60d28 459dabcd   |...........(E...|
00000010: 5141                                  |QA|
```

7.4 Trade Report Message

```
tradeReportEncoder.timestamp().time(getTimeInNanos());
tradeReportEncoder.securityID(0xABCD);
tradeReportEncoder.tradeID(0x0102030405060708L);
tradeReportEncoder.tradeQty(40L);
tradeReportEncoder.tradePrice().mantissa(123450000);
tradeReportEncoder.saleCondition1(SCSettlementType.Regular);
tradeReportEncoder.saleCondition2(SCTradeThroughExemptionType.IntermarketSwee
p);
tradeReportEncoder.saleCondition3(SCExtendedHoursOrSoldType.NotApplicable);
tradeReportEncoder.saleCondition4(SCSROTradeDetailType.Cross);

Hex code:
00000000: 00220a04 00010005 e2c60d90 97a2abcd   |."..............|
00000010: 01020304 05060708 00000028 00000000   |...........(....|
00000020: 075bb290 40462058                     |.[..@F X|
```

7.5 Trade Cancel Message

```
tradeCancelEncoder.timestamp().time(getTimeInNanos());
tradeCancelEncoder.securityID(0xABCD);
tradeCancelEncoder.tradeID(0x0102030405060708L);
tradeCancelEncoder.tradeQty(1000L);
tradeCancelEncoder.lastPrice().mantissa(123450000);
tradeCancelEncoder.saleCondition1(SCSettlementType.Regular);
tradeCancelEncoder.saleCondition2(SCTradeThroughExemptionType.IntermarketSwee
p);
tradeCancelEncoder.saleCondition3(SCExtendedHoursOrSoldType.NotApplicable);
tradeCancelEncoder.saleCondition4(SCSROTradeDetailType.Cross);

Hex code:
00000000: 00220b04 00010005 e2c60d50 b9caabcd   |."........P....|
00000010: 01020304 05060708 000003e8 00000000   |................|
00000020: 075bb290 40462058                     |.[..@F X|
```

7.6 Trade Correct Message

```
tradeCorrectEncoder.timestamp().time(getTimeInNanos());
tradeCorrectEncoder.securityID(0xABCD);
tradeCorrectEncoder.tradeID(0x0102030405060708L);
tradeCorrectEncoder.originalTradeQty(1000L);
tradeCorrectEncoder.originalTradePrice().mantissa(123450000);
tradeCorrectEncoder.originalSaleCondition1(SCSettlementType.Regular);
tradeCorrectEncoder.originalSaleCondition2(SCTradeThroughExemptionType.Interm
arketSweep);
tradeCorrectEncoder.originalSaleCondition3(SCExtendedHoursOrSoldType.NotAppli
cable);
tradeCorrectEncoder.originalSaleCondition4(SCSROTradeDetailType.Cross);
tradeCorrectEncoder.correctedTradeQty(1100L);
tradeCorrectEncoder.correctedTradePrice().mantissa(123440000);
tradeCorrectEncoder.correctedSaleCondition1(SCSettlementType.Regular);
tradeCorrectEncoder.correctedSaleCondition2(SCTradeThroughExemptionType.Inter
marketSweep);
tradeCorrectEncoder.correctedSaleCondition3(SCExtendedHoursOrSoldType.NotAppl
icable);
tradeCorrectEncoder.correctedSaleCondition4(SCSROTradeDetailType.Cross);

Hex code:
00000000: 00320c04 00010005 e2c60d7c 963dabcd   |.2.........|.=..|
00000010: 01020304 05060708 000003e8 00000000   |................|
00000020: 075bb290 40462058 0000044c 00000000   |.[..@F X...L....|
00000030: 075b8b80 40462058                     |.[..@F X|
```