

FAST for OPRA

SIAC Technical Information for
OPRA Data Recipients

Document Version: 01.00.02
Date: March 26, 2007

Revision History

VERSION 1.0 – December 8, 2006	
PAGE(S)	DESCRIPTION
All	Initial Version

VERSION 1.00.01 – February 27, 2007	
PAGE(S)	DESCRIPTION
2	<ul style="list-style-type: none"> - Updated URL link for FAST 1.1 specification - Updated the Template (Field Operator) definition table to reflect OPRA Field Name, Field Operator and Data Type
3	<ul style="list-style-type: none"> - Added SOH, US, and ETX fields

VERSION 1.00.02 – March 26, 2007	
PAGE(S)	DESCRIPTION
ALL	<ul style="list-style-type: none"> - Added Revision History - Added FAST Template (Field Operator) descriptions - Updated definitions to reflect changes in how OPRA Fields are encoded - Change BID_INDEX_VALUE field to be encoded as a STRING Data Type - Change OFFER_INDEX_VALUE field to be encoded as a STRING Data Type - Added a one byte message length field preceding each encoded message - Updated the sample code - Added note regarding elimination of Unit Separator in encoded packets - Added note regarding relocation of Message Category field in encoded messages - A ‘C’ language OPRA FAST decoder supporting all current OPRA message formats has been provided in the attached zip file

FAST

Overview:

FAST (<http://www.fixprotocol.org/fast>) is an acronym for FIX Adapted for Streaming. The FAST protocol is defined by the FIX protocol Market Data Optimization Working Group, whose purpose is to develop recommended enhancements to support high frequency market data applications. A technical overview of the protocol can be downloaded from <http://www.fixprotocol.org/documents/2801/FIX%20Adapted%20for%20STreaming%20-%20FAST%20Protocol.pdf>

Why FAST:

FAST is designed to develop solutions for efficient dissemination of market data. The protocol optimizes communications in the electronic exchange of financial data by reducing the bandwidth between sender and receiver via algorithmic data compaction within each packet.

FAST for OPRA

The FAST protocol has been integrated with OPRA to reduce the bandwidth of OPRA messages. Pursuant to the OPRA notice dated December 20, 2006, beginning on April 9, 2007, OPRA will simultaneously disseminate production multicast data in a dual network mode: current (ASCII) and new FAST encoded feeds.

Implementation:

FAST API (<http://www.fixprotocol.org/documents/2317/fastapi-1.0.zip>), a ‘C’ language implementation of the FAST Protocol, is used to encode OPRA message formats into the FAST format. The FAST Protocol API must be used by OPRA Data Recipients to decode the encoded OPRA messages. The API reference manual and sample implementation can be downloaded from <http://www.fixprotocol.org/fastdownload>

OPRA FAST Template (Field Operator):

OPRA encodes data utilizing FAST Templates (Field Operators) as described in the table below. A Field Operator defines the structure of encoded data and specifies how data in each field of the OPRA message format is encoded. Data recipients should use these Field Operators during their decoding process. Please refer to chapter 6 of [FAST Specification Version 1.1](#) for a detailed explanation of the Field Operators.

Encode/Decode Field Operator explanations:

COPY CODE: Fields that frequently have the same value in successive instances.

The copy operator specifies that the value of a field is optionally present in the stream. If the value is present in the stream it becomes the new previous value.

When the value is not present in the stream there are three cases depending on the state of the previous value:

- assigned – the value of the field is the previous value.
- undefined – the value of the field is the initial value that also becomes the new previous value. Unless the field has optional presence, it is a dynamic error [ERR D5] if the instruction context has no initial value. If the field has optional presence and no initial value, the field is considered absent and the state of the previous value is changed to empty.
- empty – the value of the field is empty. If the field is optional the value is considered absent. It is a dynamic error [ERR D6] if the field is mandatory.

The copy operator is applicable to all field types.

Description:

The value of {F} will be equal to the previous instance of {F} or {V} if it is the first instance of {F}. A protocol error should be signaled if there is no previous value and {V} has not been specified.

The value of {F} can be set to NULL if the field type supports a NULL value.

Examples:

Field Operator Entry	Previous Value	Field Content	Field Value
167=	[None]	[Empty]	[Error]
167=	FUT	[Empty]	FUT
167=FUT	[None]	[Empty]	FUT
167=FUT	IDX	[Empty]	IDX
167=FUT	IDX	FUT	FUT

INCREMENT: Fields that frequently have successive values which are incrementally larger than the previous value (sequence numbers).

The increment operator specifies that the value of a field is optionally present in the stream. If the value is present in the stream it becomes the new previous value.

When the value is not present in the stream there are three cases depending on the state of the previous value:

- assigned – the value of the field is the previous value incremented by one. The incremented value also becomes the new previous value.

FAST for OPRA

- Undefined – the value of the field is the initial value that also becomes the new previous value. Unless the field has optional presence, it is a dynamic error [ERR D5] if the instruction context has no initial value. If the field has optional presence and no initial value, the field is considered absent and the state of the previous value is changed to empty.
- Empty – the value of the field is empty. If the field is optional, the value is considered absent. It is a dynamic error [ERR D6] if the field is mandatory.

The increment operator is applicable to integer field types.

An integer is incremented by adding one to it. If the value is the maximum value of the type it becomes the minimum value after the increment.

Description:

{N} is a numeric default value.

The value of {F}, if not specified in a message field, will be the value of the previous value of {F} incremented by one, or {N} if it is the first instance of {F}.

If a value is specified in the message it will be used as the current value of {F} and it will be used as the previous value in a subsequent instance of {F} in the same message.

The value of {F} can be set to NULL. The increment of NULL is NULL (which is essentially copy coding behavior for a NULL previous value).

Examples:

Field Operator Entry	Previous Value	Field Content	Field Value
34+	[None]	[Empty]	[Error]
34+	325	[Empty]	326
34+1	[None]	[Empty]	1
34+1	325	[Empty]	326
34+1	325	401	401

DELTA: Fields that frequently have values that are almost equal to the previous value in the same message.

The delta operator specifies that a delta value is present in the stream. If the field has optional presence, the delta value can be NULL. In that case the value of the field is considered absent. Otherwise the field is obtained by combining the delta value with a base value.

Delta = element delta { opContext }

The base value depends on the state of the previous value in the following way:

- assigned – the base value is the previous value.
- Undefined – the base value is the initial value if present in the instruction context. Otherwise a type dependant default base value is used.
- Empty – it is a dynamic error [ERR D6] if the previous value is empty.

The following sections define the delta value representations, the default base values and how values are combined depending on type.

Description:

The value of {F} will be the value of the previous instance of {F} plus the (delta) value specified for the current instance of {F} (the value given in the message is the delta from the previous instance of {F}). If the value is not specified, zero (0) is used as a default delta value.

For character string fields, the delta is defined as being the tail characters of the field. As a consequence, the delta value coding can only be used on character string fields with a fixed length.

The value of {F} can be set to NULL if the field type supports a NULL value. The default delta of NULL is NULL (which is essentially copy coding behavior for a NULL previous value).

Examples:

Field Operator Entry	Previous Value	Field Content	Field Value
270-	[None]	[Empty]	[Error]
270-	[None]	1010	1010
270-	1010	[Empty]	1010
270-	1010	0	1010
270-	1010	-20	990
48-	[None]	[Empty]	[NULL]
48-	[None]	CME000150112	CME000150112
48-	CME000150112	[Empty]	CME000150112
48-	CME000150112	413	CME000150413
48-	CME000150413	0	CME000150410

OPRA FAST ENCODING/DECODING SPECIFICATION		
OPRA FIELD NAME	Field Operator	Data Type
BBO INDICATOR	COPY CODE	Unsigned Integer
BEST OFFER PRICE	COPY CODE	Unsigned Integer
BEST OFFER PRICE DENOMINATOR CODE	COPY CODE	Unsigned Integer
BEST OFFER SIZE	COPY CODE	Unsigned Integer
BEST BID PARTICIPANT ID	COPY CODE	Unsigned Integer
BEST BID PRICE	COPY CODE	Unsigned Integer
BEST BID PRICE DENOMINATOR CODE	COPY CODE	Unsigned Integer
BEST BID SIZE	COPY CODE	Unsigned Integer
BEST OFFER PARTICIPANT ID	COPY CODE	Unsigned Integer
BID INDEX VALUE	COPY CODE	STRING
BID PRICE	COPY CODE	Unsigned Integer
BID SIZE	COPY CODE	Unsigned Integer
DECIMAL PLACEMENT INDICATOR	COPY CODE	Unsigned Integer
DEF MSG	COPY CODE	STRING
EXPIRATION DATE	COPY CODE	Unsigned Integer
EXPLICIT STRIKE PRICE	COPY CODE	Unsigned Integer
FCO SYMBOL	COPY CODE	STRING
HIGH PRICE	COPY CODE	Unsigned Integer
INDEX SYMBOL	COPY CODE	STRING
INDEX VALUE	COPY CODE	STRING
LAST PRICE	COPY CODE	Unsigned Integer
LOW PRICE	COPY CODE	Unsigned Integer
MESSAGE CATEGORY	COPY CODE	Unsigned Integer
MESSAGE SEQUENCE NUMBER	INCREMENT	Unsigned Integer
MESSAGE TYPE	COPY CODE	Unsigned Integer
NET CHANGE	COPY CODE	Unsigned Integer
NET CHANGE INDICATOR	COPY CODE	Unsigned Integer
NUMBER OF FOREIGN CURRENCY SPOT VALUES IN GROUP	COPY CODE	Unsigned Integer
NUMBER OF INDICES IN GROUP	COPY CODE	Unsigned Integer
OFFER INDEX VALUE	COPY CODE	STRING
OFFER PRICE	COPY CODE	Unsigned Integer
OFFER SIZE	COPY CODE	Unsigned Integer
OPEN INT VOLUME	COPY CODE	Unsigned Integer
OPEN PRICE	COPY CODE	Unsigned Integer
PARTICIPANT ID	COPY CODE	Unsigned Integer
PREMIUM PRICE	COPY CODE	Unsigned Integer
PREMIUM PRICE DENOMINATOR CODE	COPY CODE	Unsigned Integer
RESERVED 1	COPY CODE	STRING
RESERVED 2	COPY CODE	STRING
RESERVED 3	COPY CODE	STRING
RETRANSMISSION REQUESTER	COPY CODE	Unsigned Integer
SECURITY SYMBOL	COPY CODE	STRING
SESSION INDICATOR	COPY CODE	Unsigned Integer
STRIKE PRICE CODE	COPY CODE	Unsigned Integer
STRIKE PRICE DENOMINATOR CODE	COPY CODE	Unsigned Integer
TEXT	COPY CODE	STRING
TIME	COPY CODE	Unsigned Integer
UNDERLYING PRICE DENOM	COPY CODE	Unsigned Integer
UNDERLYING STOCK PRICE	COPY CODE	Unsigned Integer
VOLUME	COPY CODE	Unsigned Integer
YEAR	COPY CODE	Unsigned Integer

Sample code for decoding category ‘k’ OPRA messages:

```
main
{
    string asciiMsg, dMsg;
    codec = create_fast_codec()

    // receive encoded pkt from wire
    u32 ePktLen = recvfrom(pkt_buf)
    u32 readLen = ePktLen;

    // drop packet if SOH and ETX not present
    if ( pkt_buf[0] != 1 and pkt_buf[ePktLen - 1] != 3 )
    {
        print "Not a valid packet";
        return -1;
    }

    // add SOH
    asciiMsg.append(SOH);           // add SOH
    pkt_buf++                      // skip 1 byte SOH
    readLen--;

    while (readLen > 1)           // decode until ETX
    {
        // read 1 byte size of encoded msg
        msgLen = *pkt_buf;
        pkt_buf++// skip 1 byte msg size
        readLen--;

        // if msgLen is 0xFF, the size is actual packet length (ePktLen)

        // assign encoded msg to fast codec
        fast.setBuffer(codec, pkt_buf, msgLen) // assigns pkt_buf to codec input buffer

        fast_decode_new_msg(OPRA_BASE_TID)      // OPRA template ID

        // read message category
        msgCategory = decode_U32(MESSAGE_CATEGORY)

        switch(msgCategory)
        {
            case 'k':
                dMsg = decode_equity_index_quote_with_size(fast);
                break
            .....
            default:
                dMsg = decode_opra_default(fast)
                break
        }

        // append decoded msg
        asciiMsg.append(dMsg)

        // append Unit Separator
        asciiMsg.append(US)

        fast_decode_end_msg(OPRA_BASE_TID)

        pkt_buf += msgLen // point to next msg
        readLen -= msgLen
    } // end of while (readLen > 1)

    // append ETX
    asciiMsg.replace(asciiMsg.size() - 1, ETX)      // replace last US with ETX
} // end of main
```

```

decode_equity_index_quote_with_size(fast)
{
    // use fast codec

    kMsg.category = 'k'
    kMsg.type = decode_u32(MESSAGE_TYPE);
    kMsg.participantId = decode_u32(PARTICIPANT_ID);
    kMsg.retran = decode_u32(RETRANSMISSION_REQUESTER)
    u32_to_ascii(kMsg.seqNumber, sizeof(kMsg.seqNumber),
                 decode_u32(MESSAGE_SEQUENCE_NUMBER))
    u32_to_ascii(kMsg.time, sizeof(kMsg.time), decode_u32(TIME))

    memset(kMsg.symbol,' ',sizeof(kMsg.symbol)) // left justified
    decode_str(SECURITY_SYMBOL, kMsg.symbol, sizeof(kMsg.symbol))

    decode_str(RESERVED_FIELD_1, kMsg.rf1, sizeof(kMsg.rf1)) //reserved

    kMsg.expirationDate = decode_u32(EXPIRATION_DATE)
    kMsg.year = decode_u32(YEAR)
    kMsg.strikePriceCode = decode_u32(STRIKE_PRICE_CODE)
    kMsg.strikePriceDenomCode = decode_u32(STRIKE_PRICE_DENOM_CODE)
    u32_to_ascii(kMsg.explicitStrike, sizeof(kMsg.explicitStrike),
                 decode_u32(STRIKE_PRICE_CODE))
    kMsg.premiumPriceDenomCode = decode_u32(PREMIUM_PRICE_DENOM_CODE)
    u32_to_ascii(kMsg.bidQuote, sizeof(kMsg.bidQuote), decode_u32(BID_PRICE))
    u32_to_ascii(kMsg.bidSize, sizeof(kMsg.bidSize), decode_u32(BID_SIZE))
    u32_to_ascii(kMsg.askQuote, sizeof(kMsg.askQuote), decode_u32(ASK_PRICE))
    u32_to_ascii(kMsg.askSize, sizeof(kMsg.askSize), decode_u32(ASK_SIZE))
    kMsg.sessionIndicator = decode_u32(SESSION_INDICATOR)
    kMsg.bboIndicator = decode_u32(BBO_INDICATOR)
    switch(kMsg.bboIndicator)
    {
        case 'A': // No Best Bid Change, No Best Offer Change
        case 'B': // No Best Bid Change, Quote Contains Best Offer
        case 'D': // No Best Bid Change, No Best Offer
        case 'E': // Quote Contains Best Bid, No Best Offer Change
        case 'F': // Quote Contains Best Bid, Quote Contains Best Offer
        case 'H': // Quote Contains Best Bid, No Best Offer
        case 'I': // No Best Bid, No Best Offer Change
        case 'J': // No Best Bid, Quote Contains Best Offer
        case 'L': // No Best Bid, No Best Offer
        case ' ': // Ineligible
        break;

        case 'C': // No Best Bid Change, Best Offer
        case 'G': // Quote Contains Best Bid, Best Offer
        case 'K': // No Best Bid, Best Offer
            kMsg.bbo.bestOffer.partId = decode_u32(BEST_OFFER_PART_ID)
            kMsg.bbo.bestOffer.denominator =
                decode_u32(BEST_OFFER_DENOM_CODE)
            u32_to_ascii(kMsg.bbo.bestOffer.price, sizeof(kMsg.bbo.bestOffer.price),
                         decode_u32(BEST_OFFER_PRICE))
            u32_to_ascii(kMsg.bbo.bestOffer.size, sizeof(kMsg.bbo.bestOffer.size),
                         decode_u32(BEST_OFFER_SIZE))
            decode_str(RESERVED_FIELD_2, kMsg.bbo.bestOffer.reserved, sizeof(kMsg.bbo.bestOffer.reserved))

        break;

        case 'M': // Best Bid , No Best Offer Change
        case 'P': // Best Bid , No Best Offer
        case 'N': // Best Bid , Quote Contains Best Offer
            kMsg.bbo.bestBid.partId = decode_u32(BEST_BID_PART_ID)
            kMsg.bbo.bestBid.denominator = decode_u32(BEST_BID_DENOM_CODE)
            u32_to_ascii(kMsg.bbo.bestBid.price, sizeof(kMsg.bbo.bestBid.price),
                         decode_u32(BEST_BID_PRICE))
            u32_to_ascii(kMsg.bbo.bestBid.size, sizeof(kMsg.bbo.bestBid.size),
                         decode_u32(BEST_BID_SIZE))
            kMsg.bbo.bestBid.reserved = '' // reserved
        break;

        case 'O': // Best Bid , Best Offer
    }
}

```

```

kMsg.bbo.bestBidOffer.bestBid.partId = decode_u32(BEST_BID_PART_ID)
kMsg.bbo.bestBidOffer.bestBid.denominator =
    decode_u32(BEST_BID_DENOM_CODE)
u32_to_ascii(kMsg.bbo.bestBidOffer.bestBid.price,
    sizeof(kMsg.bbo.bestBidOffer.bestBid.price),
    decode_u32(BEST_BID_PRICE))
u32_to_ascii(kMsg.bbo.bestBidOffer.bestBid.size,
    sizeof(kMsg.bbo.bestBidOffer.bestBid.size),
    decode_u32(BEST_BID_SIZE))
kMsg.bbo.bestBidOffer.bestBid.reserved = '' // reserved
kMsg.bbo.bestBidOffer.bestOffer.partId =
    decode_u32(BEST_OFFER_PART_ID)
kMsg.bbo.bestBidOffer.bestOffer.denominator =
    decode_u32(BEST_OFFER_DENOM_CODE)
u32_to_ascii(kMsg.bbo.bestBidOffer.bestOffer.price,
    sizeof(kMsg.bbo.bestBidOffer.bestOffer.price),
    decode_u32(BEST_OFFER_PRICE))
u32_to_ascii(kMsg.bbo.bestBidOffer.bestOffer.size,
    sizeof(kMsg.bbo.bestBidOffer.bestOffer.size),
    decode_u32(BEST_OFFER_SIZE))
decode_str(RESERVED_FIELD_3, kMsg.bbo.bestBidOffer.bestOffer.reserved,
    sizeof(kMsg.bbo.bestBidOffer.bestOffer.reserved))

kMsg.bbo.bestBidOffer.bestOffer.reserved = '' // reserved
break;
}
}

```

Notes:

- Encoded packet format:

S	M S G L E N	OPRA FAST ENCODED MESSAGE	M S G L E N	OPRA FAST ENCODED MESSAGE	M S G L E N	OPRA FAST ENCODED MESSAGE	E T X
---	----------------------------	---------------------------------	----------------------------	---------------------------------	----------------------------	---------------------------------	-------------

- A new one byte field that indicates message length (msgLen) precedes each encoded OPRA FAST message.
- If the encoded message length is longer than 254 bytes, then the new field will contain the binary value of 255 (0xFF). In this situation, the end of message will be determined by the ETX delimiter following the message.
- Unit Separators will not be part of the OPRA encoded message. The decoder will add Unit Separators to comply with OPRA specifications.
- The Message Category field will be located at the beginning of the encoded message as opposed to its position in the OPRA ASCII format.
- The Start Of Header (SOH) and End Of Text (ETX) are converted into the binary format, 0x01 and 0x03 respectively. These two fields are not encoded into the FAST format.
- In the event an invalid message category is received by OPRA, the entire message body would be encoded as a string.
- Refer to the current OPRA specifications for OPRA field descriptions. Any OPRA field defined as numeric will be decoded into integer values. If non-numeric data is received for a numeric field, field contents will be decoded into zeroes.
- u32_to_ascii is a utility function that converts integers to ASCII.
- A ‘C’ language OPRA FAST decoder supporting all current OPRA message formats has been provided. This decoder may be used to reconstitute the original OPRA formatted packets. Data recipients are welcome to utilize any FAST decoder program that uses the FAST API described in [FAST Specification Version 1.1](#), however SIAC support will be limited.