



Securities Industry Automation Corporation
P.O. Box 24270, Brooklyn, NY 11202-4270

March 26, 2007

To: OPRA Multicast Data Recipients

Subject: OPRA FAST Protocol

Attached you will find a 'C' language OPRA FAST decoder supporting all current OPRA message formats. This decoder may be used to reconstitute the original OPRA formatted packets. Data recipients are welcome to use any FAST decoder program that uses the FAST API described in [FAST Specification Version 1.1](#).

Also attached is a revised version of the SIAC FAST for OPRA technical document (Version 1.00.02 dated March 26, 2007) which is FAST version 1.1 compliant. A revision table has been included in this document that tracks updates provided. Of note is a new one byte field preceding every encoded OPRA FAST message. This new field indicates the length of each message eliminating the need for Unit Separators. The sample code previously provided has also been updated to represent all changes.

Also provided is a list of OPRA FAST questions submitted by Data Recipients, along with responses.

Testing

Testing with FAST is now available. Data Recipient test scheduling should be coordinated through Joe Loughran at (212) 383-4908 or Cheryl Boamah at (212) 383-4920.

Technical Support

During the phase-in period, technical support to assist Data Recipients with migration questions will be available. Please email your questions to both Laura Guzzy (lguzzy@siac.com, 212-383-8373) and Melissa Carannante (mcaranna@siac.com, 212-383-2278).

Sincerely,

Michael Collazo
Director
National Market Systems (NMS)
Product Planning & Management

cc:

A. Bach	R. Jakob
B. Faughnan	OPRA Policy Committee
D. Mrakovcic	OPRA Technical Committee

FAST for OPRA

SIAC Technical Information for
OPRA Data Recipients

Document Version: 01.00.02

Date: March 26, 2007

FAST for OPRA

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	- Updated URL link for FAST 1.1 specification
3	- Updated the Template (Field Operator) definition table to reflect OPRA Field Name, Field Operator and Data Type
3	- 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 **F**IX **A**dapted for **S**Treaming. 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.

FAST for OPRA

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

```

```

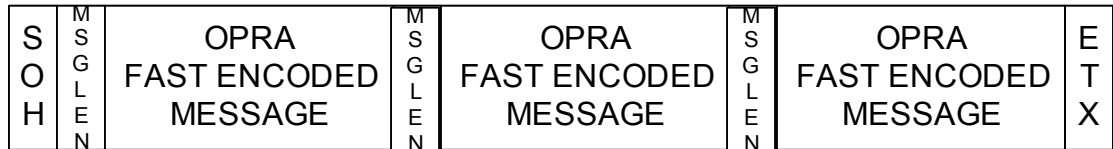
kMsg.bbo.bestBidOffer.bestBid.partId = decode_u32(BEST_BID_PART_ID)
kMsg.bbo.bestBidOffer.bestBid.denominator =
    decode_u32(BEST_BID_DENOM_CODE)
u32_to_ascii(kMsg.bbo.bestBidOffer.bestBid.price,
    sizeof(kMsg.bbo.bestBidOffer.bestBid.price),
    decode_u32(BEST_BID_PRICE))
u32_to_ascii(kMsg.bbo.bestBidOffer.bestBid.size,
    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:

1. Encoded packet format:



2. A new one byte field that indicates message length (msgLen) precedes each encoded OPRA FAST message.
3. 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.
4. Unit Separators will not be part of the OPRA encoded message. The decoder will add Unit Separators to comply with OPRA specifications.
5. The Message Category field will be located at the beginning of the encoded message as opposed to its position in the OPRA ASCII format.
6. 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.
7. In the event an invalid message category is received by OPRA, the entire message body would be encoded as a string.
8. 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.
9. u32_to_ascii is a utility function that converts integers to ASCII.
10. 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.

FAST For OPRA Questions/Responses As of 3/26/07

QUESTION	RESPONSE
How do we track transmission gaps if every message received simply increments previously processed sequence numbers?	The first message for each packet is the reference. There are no gaps within a packet.
Since the FAST protocol relies on increment and copy attributes of OPRA fields this implies that a receiver cannot miss one single packet of data otherwise they may have missed a value that had changed in the missed packet. Is this true?	
Will the first occurrence of the sequence number in a message block will be used for increment.	
Is the FAST context carried across from one packet to the next, or is it totally self contained for all the messages within a multicast packet?	
With the FAST protocol it appears that we will have to do a recovery to ensure that no packet is missed otherwise the FAST context will be lost.	
Will the decoder accept FAST for OPRA input and generate exactly the same ASCII fixed format of the OPRA message we receive today?	Yes
Will the current order of fields be identical to the FAST order for category 'k' and 'H' messages? If not what will the order be?	With the exception of the Message Category field, the FAST format will be in the same order as the current messages.
For Category 'k' messages, if the BBO Indicator is C,G,K... what will the message look like? Will the appendage follow the body immediately or will there be a 16 byte gap (for bid appendage) embedded in the message?	The Category 'k' message is encoded without gaps as per the current message.
OPRA Packets are currently limited to 1004 bytes per packet; will that remain true once we move to FAST?	The current ASCII packets and the new FAST packets will contain the same messages; the packet maximum will still be 1004 bytes.
What is the OPRA FAST bandwidth rate expected during the initiation/testing period?	As per the FAST notice dated February 27, 2007, the FAST bandwidth will be approximately 30% of the current bandwidth (130% if reading/processing both the current ASCII and new FAST feeds simultaneously).
Do the 2007 Projections, distributed on 1/11/2007 take both Regular and FAST protocols into account?	No, the bandwidth reflects just the regular or current feed. An additional 30% bandwidth is required if you receive both the current and FAST feeds.

```
#####  
#  
# Copyright (c) :      SIAC 2007  
# Date          :      March 26, 2007  
# File          :      readme.txt  
#  
#####
```

This package contains source code and test data files for the Opra FAST Specification.
The code and sample programs have been tested on HP-UX using CC and Linux using GCC.

NOTE: Please use a specific compiler for the operating system being used.

The encoded file contains the encoded packet size followed by the encoded packet. Each message in the encoded packet is preceded by its one byte message length. This will help to skip to the next message in the packet, if needed. The UNIT SEPARATOR is not sent between messages in the encoded packet.

The SIAC provided OPRA FAST decoder will append a UNIT SEPARATOR at the end of each message to comply with OPRA specifications.

The FASTforOPRA_Decode.tar.gz contains the following files

```
- readme.txt          # Explanation of package content and usage  
- Makefile           # Makefile used to compile and link software  
(HP-UX and LINUX)  
- types.h            # header file  
- common.h           # header file  
- opra_fast.h        # Definition of opra messages  
- fast_process.h     # Header for decode API  
- fast_wrapper.h     # Declaration of fast_wrapper structure  
- fast_api.h         # FAST API definition  
- fast_main.c        # Main function is written to simulate the UDP  
packet  
- fast_process.c     # definition of the main decode function  
- fast_wrapper.c     # Function definitions to interact with the  
FAST CODEC  
- fast_decode.c      # Function definitions to decode various  
fields of the encoded OPRA message.  
- fast_api.c         # FAST CODEC implementation  
- testmsgs.base      # Ascii OPRA file  
- testmsgs.encoded  # Encoded file
```

COMPILATION:

Build the executable using the following command for HP-UX and LINUX OS:

```
make -f Makefile
```

This will create executable with name "opra_c_decoder".

NOTE: Currently, the makefile supports both "cc" and "gcc" compilers.
So, please uncomment the required compilation statement inside
the makefile. Also, please feel free to edit the makefile to
add new
compiler if neither "cc" nor "gcc" is supported.

TESTING:

To test, please compile the source code with the DEBUG flag ON
and run the executable as follows:

```
<executableName> <encodedfile> <decodefile>  
e.g. opra_c_decoder testmsgs.encoded decoded.new.packet  
Where:  
opra_c_decoder      -> executable name  
testmsg.encoded     -> encoded test file (provided in the package)  
decoded.new.packet  -> output OPRA ASCII packet file
```

After running the executable, a new output file "testmsgs.new.base" will be
generated.

To verify the test results please run the diff command as follows:

```
diff testmsgs.base testmsgs.new.base
```

There should be no difference between the two files. This test will ensure
that the
executable is not corrupted and is running as expected.

NOTE: The encoded file is for the format length of binary data followed by
actual binary
data. The test program is made to work in the same way. But in actual
production,
there will be no length field.