A Basic User's Guide to Implementing The FAST Protocol℠

Version 1.0

January, 2006

DISCLAIMER

THE INFORMATION CONTAINED HEREIN AND THE FINANCIAL INFORMATION EXCHANGE PROTOCOL (COLLECTIVELY, THE "FIX PROTOCOL") ARE PROVIDED "AS IS" AND NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL MAKES ANY REPRESENTATION OR WARRANTY, EXPRESS OR IMPLIED, AS TO THE FIX PROTOCOL (OR THE RESULTS TO BE OBTAINED BY THE USE THEREOF) OR ANY OTHER MATTER AND EACH SUCH PERSON AND ENTITY SPECIFICALLY DISCLAIMS ANY WARRANTY OF ORIGINALITY, ACCURACY, COMPLETENESS, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. SUCH PERSONS AND ENTITIES DO NOT WARRANT THAT THE FIX PROTOCOL WILL CONFORM TO ANY DESCRIPTION THEREOF OR BE FREE OF ERRORS. THE ENTIRE RISK OF ANY USE OF THE FIX PROTOCOL IS ASSUMED BY THE USER.

NO PERSON OR ENTITY ASSOCIATED WITH THE FIX PROTOCOL SHALL HAVE ANY LIABILITY FOR DAMAGES OF ANY KIND ARISING IN ANY MANNER OUT OF OR IN CONNECTION WITH ANY USER'S USE OF (OR ANY INABILITY TO USE) THE FIX PROTOCOL, WHETHER DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL (INCLUDING, WITHOUT LIMITATION, LOSS OF DATA, LOSS OF USE, CLAIMS OF THIRD PARTIES OR LOST PROFITS OR REVENUES OR OTHER ECONOMIC LOSS), WHETHER IN TORT (INCLUDING NEGLIGENCE AND STRICT LIABILITY), CONTRACT OR OTHERWISE, WHETHER OR NOT ANY SUCH PERSON OR ENTITY HAS BEEN ADVISED OF, OR OTHERWISE MIGHT HAVE ANTICIPATED THE POSSIBILITY OF, SUCH DAMAGES.

DRAFT OR NOT RATIFIED PROPOSALS (REFER TO PROPOSAL STATUS AND/OR SUBMISSION STATUS ON COVER PAGE) ARE PROVIDED "AS-IS" TO INTERESTED PARTIES FOR DISCUSSION ONLY. PARTIES THAT CHOOSE TO IMPLEMENT THIS DRAFT PROPOSAL DO SO AT THEIR OWN RISK. IT IS A DRAFT DOCUMENT AND MAY BE UPDATED, REPLACED, OR MADE OBSOLETE BY OTHER DOCUMENTS AT ANY TIME. THE FPL GLOBAL TECHNICAL COMMITTEE WILL NOT ALLOW EARLY IMPLEMENTATION TO CONSTRAIN ITS ABILITY TO MAKE CHANGES TO THIS SPECIFICATION PRIOR TO FINAL RELEASE. IT IS INAPPROPRIATE TO USE FPL WORKING DRAFTS AS REFERENCE MATERIAL OR TO CITE THEM AS OTHER THAN "WORKS IN PROGRESS". THE FPL GLOBAL TECHNICAL COMMITTEE WILL ISSUE, UPON COMPLETION OF REVIEW AND RATIFICATION, AN OFFICIAL STATUS ("APPROVED") TO THE PROPOSAL AND A RELEASE NUMBER.

No proprietary or ownership interest of any kind is granted with respect to the FIX Protocol (or any rights therein).

Table of Contents

Document History

Revision	Date	Author	Revision Comments
Version 0.1	Nov 19, 2005	Matt Simpson	Initial Draft
Version 0.2	Nov 21, 2005	Matt Simpson	Second Draft
Version 0.3	Jan 6, 2006	Matt Simpson	
Version 0.4	Jan 19, 2006	Rich Shriver	Recommendations for changes to the content and external references
Version 0.5	Jan 23, 2006	Matt Simpson	Additional notes and changes
Version 0.6	Jan 24, 2006	Matt Simpson	Added comments on decoding errors

Purpose

The purpose of this document is to describe the proper use of the FAST Protocol in a *one-way exchange of data*, from a sender to one or more receivers as commonly found in market data applications. The document will address both broadcast and point-to-point configurations. This document is not a tutorial or a technical specification. Please see the following links for this information.
FAST Tutorial: pending
FAST Field Encoding Specification:
http://fixprotocol.org/documents/2254/FAST%20Field%20Encoding%20Specification%201.0.doc

FAST Transfer Encoding Specification:
http://fixprotocol.org/documents/2255/FAST%20Transfer%20Encoding%20Specification%201.0.doc

Later versions will describe the proper use of FAST in a two-way exchange of data and the proper application in a FIX Session.

Abstract

The FAST Protocol has been developed as part of the FIX Market Data Optimization Working Group. FAST is an acronym for FIX Adapted for STreaming and is designed to optimize communication in the electronic exchange of financial data. FAST, at its core, is a data compression algorithm which when properly implemented will significantly reduce bandwidth requirements and latency between sender and receiver. FAST works especially well at improving performance during periods of peak message rates. For more information on the benefits and real-world results from FAST Protocol implementations, please refer to the Proof of Concept Test Results.

Phase 1A Test Results
Phase 1B Test Results

FAST is an extension of the base FIX specification and any discussion of data structures will assume FIX message formats. However, a basic implementation of FAST does not require or assume a FIX implementation. FAST exists as a stand alone specification which can be used within either broadcast or point-to-point transports.

FAST is also intended to be used as an extension of the FIX session layer, but this document does not currently address this usage. The following discussion will be concerned with the all the practical aspects of implementing FAST in a one-way broadcast or point-to-point configuration.

How FAST Works

The FAST Protocol has been designed to be a flexible and highly extensible solution for high volume, low latency data dissemination. The FAST Protocol is an encoding algorithm which reduces the size of a data stream on two-levels. First, a technique referred to as Field Encoding allows data affinities of a stream to be leveraged and redundant data to be removed. Second, Transfer Encoding of the remaining data is accomplished through binary encoding which also draws on self-describing field lengths and bit maps indicating the presence or absence of fields.

The FAST Protocol is predicated on familiarity with the content of a data feed. This is done through the use of templates which describe the content and characteristics of the data to be encoded or decoded. Templates inform the encoder/decoder of the operations to be used in the encoding and decoding process. Templates will be discussed in greater detail later in this document.

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

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

Increase in processing latency minus decrease in transfer latency

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

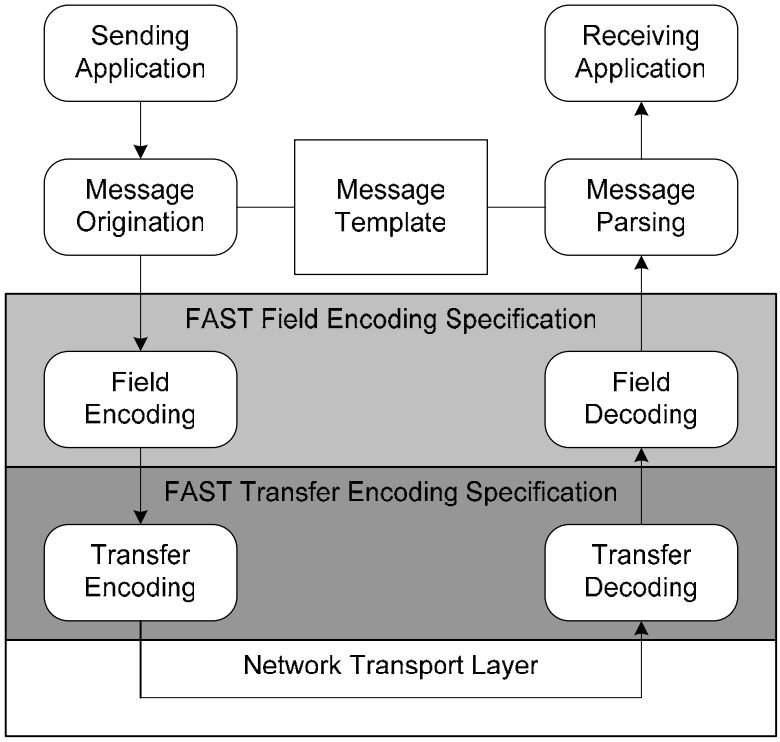


Figure 1 - FAST Protocol Components

Templates

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

Field Encoding

Compression of data relies on removing redundancy in the encoding of information. Traditionally, FIX encodes data by using a tag=value construct. In some applications, this format creates a large overhead.
This specification documents opportunities to implement efficient field level encoding by taking advantage of knowledge about the values of successive instances of a field within a message and between messages.

For more information on field encoding, please refer to the FAST Field Encoding Specifications at http://fixprotocol.org/documents/2254/FAST%20Field%20Encoding%20Specification%201.0.doc

Transfer Encoding (Serialization)

The use of binary representation in the form of numbers and strings with an embedded stop bit in each data byte and a presence map that indicates presence or absence of individual fields in a message. The encoding and decoding process is assumed to have access to a message template specification.

For more information on Transfer Encoding, please refer to the FAST Transfer Encoding Specifications at http://fixprotocol.org/documents/2255/FAST%20Transfer%20Encoding%20Specification%201.0.doc

Templates in FAST

What are Templates?

Templates are used to represent the structure of the data that is to be encoded. A template represents a logical unit of data as it is transmitted from sender to receiver. In other words, a template is used to represent a specific message type. *When planning to encode a data feed a user should begin by converting standard message formats to templates*.

All message types must be expressed in the proper template format. This will be discussed below. A template specifies the all fields included in a message as well as the sequence of those fields. Additionally, a template may also support repeating groups which allows a single message to efficiently convey multiple instructions; trades, quotes, etc.

Template Definition Approaches
There are two primary approaches used when defining templates for a market data feed. The first approach defines one template per message type. Each template is specifically tailored for a given message type. This has the advantage of maximizing data compression since each message type can be optimally encoded. However, it also requires the added complexity of managing multiple templates per feed as well as potentially expanding the scope of the "operator context" to a "global" level such that a field encoding can occur across multiple message types. There will be more discussion on this concept below.

The second approach is to define a generic template which can be used to describe multiple message types. In this approach all the fields from a set of message types are included in the template. When a message does not contain a field in the template it will use a reserved value to indicate that no data value for that field is present. This generic template approach is one that works well with FIX market data messages which supports multiple entry types (bids, asks, trades, hig/lows, etc) in a single message. This will also be discussed in greater detail below.

Messages that do not conform to the specified template will be considered invalid and be unreadable by FAST. An invalid message is one that has additional fields, missing fields, or provides fields out of sequence.

How FAST Uses Templates

The FAST Protocol encodes and decodes on a field by field basis and cannot function without a template to provide the necessary information for each field. Templates provide content-awareness with respect to a data feed and instruct FAST how to encode or decode each field within a message.

Templates provide critical information for both field encoding and serialization operations. Field Encoding instructions are conveyed by the template to the FAST Codec which performs the appropriate field encoding operations. Data type descriptions are also specified in the template and inform the Serializer as to whether a field is a string, an integer, or a decimal value. The Serializer will then perform the appropriate serialization operation based on that data type.

It is important to note that FAST functions as a state machine and must know which field values to keep in memory. FAST compares the current value of a field to the prior value of that field and determines if the new value can be copy coded, delta valued, or incremented. Field Encoding instructions carried in the template convey this information.

Template Definition and Syntax

Templates are defined using a standard syntax that can be interpreted by FAST. In general, it is recommended that an API layer be used to load templates and make them available to the core encoder. Templates should be defined using a standard set of symbols that are used to express the characteristics of the data. These are:

- *fields*
- *field encoding instructions*
- *data types*

The recommendation and assumption is that templates will be based on a FIX message format, although templates may also be used to describe any proprietary message format.

In a standard FIX implementation fields are represented using tag numbers. In a non-FIX implementation field names may be used instead of tag numbers although it is recommended that custom tag numbers be used.

Template Definition Using Compact Notation

Compact notation is straightforward way to define a template using tag number or field name, a field encoding operation symbol, and data type symbol. It allows information regarding a FAST data structure to be conveyed in easy to understand meta data. While compact notation provides benefits due to its simplicity it is not an extensible solution and has known limitations. XML Template Definition described below is a more complete solution but requires knowledge of XML documents and schemas.

The field encoding operators that are valid for use in a template are shown in Table1 below using compact notation

Table 1 Compact Notation - Field Encoding Operators

Entry	Description
!	Default Coding – default value per template
=	Copy Coding – copy prior value
+	Increment Coding – increment prior value
-	Delta Coding – numeric or string differential
@	Constant Value Coding - constant value specified in template
*****	Derived Value Coding – implies field values

The data type descriptors that are valid for use in a template are shown in Table 2 below using compact notation

Table 2 Compact Notations - Data Type Descriptors

Entry	Description
s	String data type
u	Unsigned integer data type

U	Unsigned integer data type supporting a NULL value
i	Signed integer data type
I	Signed integer data type supporting a NULL value
F	Scaled number data type
n	Composite scaled number
v	Byte vector data type

Template Structure

The proper structure for forming tags, field encoding operators and data type descriptors into a template field entry is shown below:

((tag number)(data type)(field encoding operator)**)**

Several of these field occurrences strung together form a template. A delimiter symbol of "|" is used to separate the occurrences

Market Data Template

Now let's use these symbols in an actual template. The example below shows a typical template that has been defined based on the *FIX Market Data Incremental Refresh* message (35=X). Note that the "|" symbol represents a basic field separator, "<" represents the start of a repeating group, and ">" represents the end of a repeating group.

8s!FIX.4.4|9u|35s!X|49s=|34u+1|268u
<279u=|269s=|55s=|167s=|270F-|271F-|346u-|276s=|277s=>

The following table provides a field-by-field account of the template:

Table 3 – Sample Template Definition

Template Entry	FIX Field Name	Field Encode Operation	Data Type	Description of Operation
8s!FIX4.4	BeginString	Default Value	String	Default BeginString to the string value of "FIX4.4"
9u	BodyLength	None	Unsigned integer	Always explicitly specify the value of BodyLength
35s!X	MessageType	Default Value	string	Default MessageType to the string value of "X"
49s=	SenderCompID	Copy Code	String	Copy SenderCompID from the prior occurrence
34u+1	SequenceNumber	Increment	Unsigned integer	Increment SequenceNumber by +1 from the prior occurrence
268u	NoMDEntries	None	Unsigned integer	Always explicitly specify the value of NoMDEntries
279u=	MDUpdateAction	Copy Code	Unsigned Integer	Copy MDUpdateAction from the prior occurrence
269s=	MDEntryType	CopyCode	String	Copy MDEntryType from the prior occurrence
55s=	Symbol	CopyCode	String	Copy Symbol from the prior occurrence
167s=	SecurityType	CopyCode	String	Copy SecurityType from the prior occurrence
270F-	MDEntryPrice	Delta Value	Scaled Number	Calculate the difference from the prior occurrence of MDEntryPrice
271F-	MDEntrySize	Delta Value	Scaled Number	Calculate the difference from the prior occurrence of MDEntryPrice
346u-	NumberOfOrders	Delta Value	Unsigned integer	Calculate the difference from the prior occurrence of NumberOfOrders
276s=	QuoteCondition	CopyCode	String	Copy QuoteCondition from the prior occurrence
277s=	TradeCondition	CopyCode	String	Copy TradeCondition from the prior occurrence

XML Template Notation

A FAST Template may also be defined as an XML document using an XML schema. This XML format is intended to be both human and machine readable and can be used for authoring, storing and interchanging FAST templates. The format is not intended to be used on the wire when two end points exchange template definitions over a FAST session. For wire transfers, the FAST Session Control Protocol [SCP] provides a FAST serialization of the structures defined by this document.

Sample XML Template Document

The xml document below describes the same template as that shown above using compact notation. For complete information on how to define templates in XML please reference the detailed FAST Template Definition Specification found at
http://fixprotocol.org/documents/2374

```
<templates xmlns="http://www.fixprotocol.org/ns/template-definition"
ns="http://www.fixprotocol.org/ns/templates/sample">

 <template name="MDRefreshSample">
  <messageRef name="MDIncrementalRefresh"/>
  <string name="8"> <constant value="FIX4.4"/> </string>
  <u32 name="9"> <implicit/> </u32>
  <string name="35"> <constant value="X"/> </string>
  <string name="49"> <constant value="CME"/> </string>
  <u32 name="34"> <increment value="1"/> </u32>
  <string name="52"> <delta/> </string>
  <u32 name="75"> <copy/> </u32>
  <sequence name="MDEntries">
   <length name="268"/>
   <decimal name="270"> <delta/> </decimal>
   <i32 name="271"> <delta/> </i32>
   <u32 name="273"> <delta/> </u32>
   <u32 name="346" presence="optional"/>
   <u32 name="1023"> <increment value="1"/> </u32>
   <string name="279"> <copy/> </string>
   <string name="269"> <copy/> </string>
   <string name="107"> <copy/> </string>
   <string name="48"> <delta/> </string>
   <string name="276"> <copy/> </string>
   <string name="274"> <copy/> </string>
   <decimal name="451"> <copy/> </decimal>
   <string name="277"> <default value="F"/> </string>
   <u32 name="1020" presence="optional"/>
   <i32 name="537"> <default value="1"/> </i32>
   <string name="1024"> <default value="0"/> </string>
   <string name="336"> <default value="2"/> </string>
  </sequence>
  <string name="10"/>
 </template>
</templates>
```

Template Operator Contexts

There are two Template Operator contexts that can be expressed in a template definition: Dictionary Context and Scope Context. Note that Template Operator Contexts are only supported in XML Notation, not Compact Notation.

Dictionary Context

The Dictionary Operator determines the range of a field encoding operation for a given field. There are 3 levels at which the Dictionary Context can be specified;

- *template level* – the dictionary is local to the current template. This means that a field with name *N* in template *T1* will share the same dictionary as a field with name *N* in template *T2* if[1] *T1* = *T2*
- *message level* – the dictionary is local to the current message type. This means that a field with name *N* in template *T1* that is a template of message type *M* will share the same dictionary as a field with name *N* in template *T2* that is a template of message type *M*.
- *global level* – the dictionary is global. All fields with name *N* share the same dictionary regardless of the template and message type.

[1] if and only if

Scope Reset Context

The Scope Operator specifies when a dictionary is automatically reset. When a dictionary is reset, the entry for a field is removed. This means that the next time a field is decoded, there is no previous value. There are four scopes:

- *group* – the dictionary is reset for each group. Note that in this case, messages and sequences are also counted as groups.
- *message* – the dictionary is reset for each message.
- *frame* – the dictionary is reset for each frame. A frame is a logical unit that is not explicitly encoded in a FAST stream. Thus the start of a frame must be signaled to the encoder and decoder by other means not defined by this specification.
- *session* – the dictionary is reset at the start of a session. A session is a logical unit that is not explicitly encoded in a FAST stream. Thus the start of a frame must be signaled to the encoder and decoder by other means not defined by this specification.

NULL Value Usage

Null values are useful when a generic template is used across multiple message types. In this situation, there may be fields which are present in one occurrence of the message but not in another.

For example, The MDEntry repeating group that is present in the FIX Market Data Incremental Refresh template shown above can be used to express a trade, bid, ask, high, low, etc within a single message. A trade entry will not use tag 346, NumberofOrders. However, FAST requires that this field be accounted for in the encoded message if specified in the template. A NULL Value would be used in this case to indicate that the field was not present in the data. When expressed in Compact Notation tag 346 would appear as:

346U-

In the serialization layer, FAST reserves binary zeros to indicate a NULL value. For more information, see the FAST Transfer Encoding Specification which can be found at http://fixprotocol.org/documents/2255/FAST%20Transfer%20Encoding%20Specification%201.0.doc

Considerations in Specifying a Template

It is important to understand the characteristics of the data to be encoded when defining a template. Some considerations to take into account when are:

- *Analyze and identify the statistical patterns that occur in a data feed and incorporate this in the template definition*. This will be necessary in determining which field encoding operations to apply. For example, if a depth book feed is being produced with consecutive "bid" entries followed by consecutive "ask" entries it will be more efficient to use Copy Coding with the MDEntryType field. However, if that same feed is being produced as a "bid" followed by the corresponding "ask", then Default Value coding will

be more efficient – as it will allow MDEntryType to be defaulted to the proper value 50% of the time

- *Sequence the fields properly within a template.* Fields which are consistently Copy Coded away should be placed at the end of the template in order to reduce the size of the presence map. Fields which are no longer physically present in a message can be dropped from the presence map if they are placed at the end of the template.
- *Determine which numeric fields carry values that would benefit from scaling.* Fields which can carry large numeric numbers are good candidates for downward scaling. Fields which convey decimal precision are good candidates for upward scaling.

Template ID

Each template is assigned a Template ID that can be used to uniquely describe the format of an encoded message. A Template ID will be carried in every encoded message which will provide a link to the correct template for decoding.

Template ID should be a simple unsigned integer carried as the first data field of every message that allows the decoding system to apply the correct template to the message upon receiving it.

Template ID is encoded and carried as the first data field of the message and will follow the first presence map of each message. It is imperative that all implementations conform to this rule in order to engender interoperability across platforms. A receiver cannot decode the data until the Template ID is located within the message.

The example that follows shows the same template from above with Template ID as tag 999. In this case, Template ID is being defaulted to the value "1234", and copy coded to this value in any subsequent messages. The combination of tag number 999 and the specified value of "1234" tells the system that the ID for this template is "1234".

999u=1234|8s!FIX.4.4|9u|35s!X|49s=|34u+1|268u
<279u=|269s=|55s=|167s=|270F-|271F-|346u-|276s=|277s=>

The sequence, then, for processing a message should be as followed:

- Locate and read the first presence map field
- Treat the next field as the Template ID – this is the first logical field of the message
- Retrieve the correct template once the Template ID has been decoded
- Decode the remainder of the message using the template

Template Exchange

Templates need to be communicated from sender to receiver. The originator of the data is responsible for distributing the templates once they have been defined. This can be accomplished through several means;

Out-of-band

Out-of-band template exchange between template sender and receiver may be accomplished through a use of a basic text file. This file may be a file downloaded from a website, or emailed from the sender to receiver. The key is that the sender and receiver agree on the format of templates and acknowledge that the exchange has occurred.

Side-band

Side-band template exchange may be accomplished through either a broadcast or point-to-point connection. In this approach, the sending party may transmit the templates either on a continual basis or in response to a request by the receiving parties. Generally, templates will be in ASCII form unless the parties agree to encode the templates. If templates are encoded, a "bootstrap" template is necessary to decode the templates. Parties need to agree on the format of the bootstrap template in advance of template transmission.

In-band
In-band template exchange may be accomplished through either a broadcast or point-to-point connection. In this approach, the sending party transmits the templates as part of the primary data feed and the receiver must pick them out of the feed. Generally, templates will be in ASCII form unless the parties agree to encode the templates. In this case, a "bootstrap" template is necessary to decode the templates. Parties need to agree on the format of the bootstrap template in advance of template transmission.

Currently, it is recommended that the first option be used – out-of-band template exchange - in which the sending party makes the templates available via text file using the conventions for template definition discussed earlier. A request-based model is still being worked out by the Market Data Optimization Work Group.

Using FAST in a Broadcast Environment

FAST can be effectively implemented in a broadcast environment in which a single sender is broadcasting a data feed to one or more receivers. A broadcast model takes place over a multicast transport which uses the User Datagram Protocol (UDP). "UDP is a simple, datagram-oriented transport layer protocol: each output operation by a process produces exactly one UDP datagram which causes one IP datagram to be sent[2]". UDP is unreliable - meaning that packets can be delivered out of sequence or even dropped.

Sending Data via UDP

The UDP payload carries the FAST encoded data and is usually 1000 to 1400 bytes in size. This payload is also known as the packet or frame. *A FAST implementation should be configured to process no more than a single frame due to the unreliable nature of UDP*. Performing encode operations across multiple UDP frames is not recommended since data may be undecipherable should one of the packets become lost or out-of-sequence.

It is recommended that the Sender Process perform the encode operation on a frame-by-frame basis. The process should count the bytes as they are loaded into a frame. When the frame is full, the UDP packet should be transmitted.

Optionally, a **frame length delimiter** can be used to indicate the amount of encoded data to be read from a UDP frame. A frame length delimiter would be specified as the first field in message, preceding even the initial presence map field. Please see **Frame Length** below for more information.

Receiving Data via UDP

Receiving and decoding a FAST data feed in a broadcast environment is relatively straightforward. The basic operation requires the decoding of data within a discreet packet as well as the ability to determine the end of one message and beginning of another within that packet.

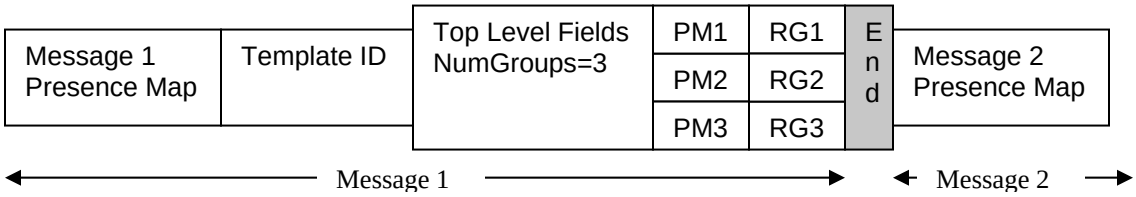
[2] TCP/IP Illustrated, Volume1 – W. Richard Stevens

Decoding a Packet
Decoding of the datagram should be conducted as a unit of work in which all data in the packet participates in the scope of the decoding process. In order to determine when the process has fully decoded a discreet message it is important to use the presence map/s and the template to step through a message field-by-field.

Decoding Messages in a Packet
For a message that uses repeating groups, the decoder must keep track of the number of repeating groups processed verses the number of repeating groups as specified in the NumberOfRepeatingGroups tag. Each repeating group will have its own presence map which will be used to in decoding the group.

When the decoder determines that it has processed all repeating groups in the message, and that the repeating group was the last structure in the template, then it can conclude that the end of the message has been reached. In this way, it is necessary to use the presence maps, Number of Repeating Group field, and template to determine when end of message has been reached. To recap, the steps involved are:

- Decode fields in the non-repeating body of the message using presence map
- If the last field in the template has been reached then end of message has been encountered
- If NumberOfRepeatingGroups tag is present, then process each repeating group using the repeating group presence map.
- When number of repeating groups processed equals NumberOfRepeatingGroups, then end of message has been encountered
- In the example below Message 1 has 3 repeating groups (RG), each repeating group has a presence map (PM), and the last repeating group signifies the logical end of the message



Error Handling in a Broadcast Environment
If FAST encounters an error during the process of decoding the contents of a UDP packet, the process should stop throw an exception and discontinue decoding. In all likelihood, the error is due to an inconsistency between the encoded data and the template being used to decode the data. At this point, it is advised that parties synchronize templates in order to ensure that the same data definitions are being used.

Using FAST in a Point-to-point Environment

FAST can be effectively implemented in a point-to-point environment. In this configuration, a sender transmits data to a single receiver. Generally, point-to-point connections are conducted over a TCP-IP transport which provides greater reliability in data delivery. Every transmission requires a response from the receiver acknowledging that the data was received in good form. This greater reliability allows the scope of a FAST transaction to be extended across a much larger window than UDP which is limited to a transmission packet.

Sending Data via TCP-IP

Because TCP-IP is reliable, FAST can operate on a much larger set of data. As long as the sender is receiving acknowledgements that the data has been successfully delivered, the sender can continue to encode from the point in time at which the TCP session was initiated. This has the potential to produce higher compression across the overall feed.

It is important for the sender to provide message or frame delimitation for the data being sent. Because TCP-IP is essentially a byte-stream the receiver will not know how many bytes must be received before decoding should begin. Because of this it is important for the sender to use a frame length delimiter which will tell the receiver how many bytes are in a logical packet. **This frame length field should be provided as the first field in any encoded message preceding even the presence map and the template id.** It would appear in a message stream as shown below.

Frame	Presence Map	Template ID	Message Body

Receiving Data via TCP-IP

A receiver of a data feed over a TCP-IP connection will need to manage the data being received as a byte stream rather than in discreet packets. Double buffering is an alternative to a byte-oriented approach but involves additional processing and complexity. In order to provide support for logical packets of data in a non-packeted feed, the use of a frame length is recommended.

Frame Length
Frame Length is very useful to a receiver in a point-to-point, byte-oriented feed and provides information with respect to the size of the logical packet that is to be decoded. The Frame Length is used to specify the encoded length of a logical packet of data and tells the receiver how many bytes to read before decoding a specific chunk of data. The sender and receiver need to agree on the scope of the packet ahead of time – that is, how many messages a logical packet should contain.

The Frame Length is always carried as the first field of any logical packet and is encoded in the same manner as all other data. The frame length field is a variable length unsigned integer, so there is not need to define a length for it. It will vary from frame to frame.

As the receiver begins to process a stream of bytes the Frame Length can be used to read the specified number of bytes and begin decoding only once those bytes have been received.

[no need to initialize at the end of the frame]

Error Handling in a TCP Environment
If FAST encounters an error during the process of decoding the contents of a TCP stream, the process should throw an exception and discontinue decoding. If frame lengths are being used the process should skip to the end of the frame before initiating decoding. In all likelihood, the error is due to an inconsistency between the encoded data and the template being used to decode the data. At this point, it is advised that parties synchronize templates in order to ensure that the same data definitions are being used.

FAST Reference Code

FAST Reference Code has been made available which can be licensed through FPL for use in FAST implementations. The reference code is useful in getting started and provides a core encoder/decoder component called the FASTapi Codec that adheres to the FAST specification.

The reference code also provides an Application Codec which contains a hard-coded template representing the FIX Market Data Incremental Refresh Message. A Test Harness is provided to execute the FASTApi Codec with the corresponding Application Codec. The Test Harness allows the implementation to be executed as either an encoder or decoder.

The FAST Reference Code provides support for most of the concepts discussed within this Basic User's Guide.

The FAST Reference Code and User's Guide can be found at the following links:
FAST Reference Code:
FAST Reference Code User's Guide: