FIX PROTOCOL
INDUSTRY-DRIVEN MESSAGING STANDARD℠

FIX Over FAST℠

Implementor's Guide

Version 1.01

2009-08-13

Status of this Document

This document provides recommendations and guidelines for the FIX community, and requests discussion and suggestions for improvements.

Distribution

Distribution of this document is unlimited.

Any references to this document should state that it is work in progress.

Copyright Notice

Disclaimer

Acknowledgements

This document is the result of the work of many members of the FPL Market Data Optimization Working Group, over many months. However, the document (and the FAST enhancements proposed as a result) could not have been accomplished without the extraordinary efforts of certain key contributors, including (in alphabetical order): Rolf Andersson, Dimitry London, Daniel May, Gregory Maynard, Jim Northey, Jacob Northey, Richard Shriver, and Matt Simpson. Special thanks are also extended to Kathleen Callahan, who could not have done a better job coordinating activities and consistently producing order where there might otherwise have been chaos.

Abstract

The FAST^{SM} protocol is one of the newest and most impactful technologies available to FIX messaging participants today. It has been shown to reduce message size by as much as 90%, with relatively low processing overhead.

This document provides best practices for addressing implementation issues relating to the use of FAST^{SM} encoding of FIX messages to enable participants to engage in low-latency message interchange in a standards-compliant environment.

This document relies on features introduced in FAST version 1.2. The guidelines stem from the same working group discussions, and – in concert - are meant to improve the FAST protocol, particularly as it relates to known efficiencies or improvements to FIX-over-FAST implementations.

Table of Contents

Document History

Version	Date	Author	Description
1.0	2009-07-02	BourseTech, Inc.	Final
1.01	2009-08-13	BourseTech, Inc.	Acknowledgements

Introduction

This document puts forth recommendations for the implementation of the Fix Adapted for STreaming^{SM} (FAST^{SM}) protocol[1] encoded FIX messages. These guidelines address version 1.2 of the FAST protocol, with an appendix providing guidelines for backward compatibility and dealing with previous versions. The reader is expected to be familiar with the FIX and FAST^{SM} specifications, as well as the ancillary resources provided on the FIX Protocol Limited (FPL) web site.

This document is the result of design discussions with the FIX Protocol Market Data Optimization Working Group (MDOWG) and test and production implementations performed by its members.

Background

The FAST^{SM} protocol was originally developed to address technological challenges specific to market data environments characterized by explosive message growth. However, the FAST^{SM} protocol may also provide unique advantages to FIX participants in ordinary point-to-point bidirectional message exchange.

As the name suggests, the Fix Adapted for Streaming protocol is meant to provide a concrete, standards-based mechanism for delivering a FIX data stream in a highly efficient manner. FAST^{SM} was an outgrowth of many coordinated efforts to reduce the wire representation of FIX messages without substantially increasing processing overhead or sacrificing any of the rich content or context which had been so carefully developed by the FIX community over a period of many years.

FAST^{SM} was ultimately designed to address (what had been) an overwhelming challenge: high-throughput, real-time market data dissemination. Market data applications are typically characterized by fixed-format message feeds broadcast over multicast UDP networks. Despite this specialized focus, and mainly due to the diligent attention given to retaining full FIX compatibility, it is both possible and practical to take advantage of the efficiencies of FAST^{SM} in many general FIX messaging usage scenarios. This document is meant to provide guidance on the beneficial use of FAST^{SM} in such scenarios, and to broaden understanding of how the FAST^{SM} standards may be applied in bidirectional FIX message sessions.

[1] "FAST^{SM} protocol" refers to the encoding and decoding rules specified in the FAST Specification v1.1, the FAST 1.2 Extension Proposal and the session protocol specified in the FAST^{SM} Session Control Protocol Specification v1.1 document (draft). Please refer to document endnotes for a complete list of references to documents available on the FIX Protocol, Limited (FPL) site, accessible at http://www.fixprotocol.org/fast/

Purpose

The purpose of this document is to

1) furnish implementation guidelines for FIX over FAST^{SM} messaging, and
2) promote seamless interoperability between messaging partners.

The document also summarizes best practices: hints, suggestions, and techniques that have been developed and employed by members of MDOWG. The reader is strongly encouraged to adopt these recommendations whenever possible, both in the interest of developing better software, and in the interest of ensuring FAST^{SM} protocol interoperability across the industry.

How to Use this Document

This document is meant to augment, not supplant, the body of FIX and FAST^{SM} documentation already available on the FIX Protocol Limited (FPL) web site.

This implementation guide is targeted to individuals tasked with implementing FIX over FAST^{SM}, regardless of whether the solution is strictly in-house, or for public consumption.

The chief focus is on implementation issues which impact performance and interoperability: i.e. exceptional cases with respect to

- session and template management,
- protocol scope,
- compression rates, and
- complex data.

The document is organized in a way that is meant to follow the implementor's thought process. The sections flow from high-level concerns to suggestions for data representation and encoding of specific fields or types such as the FIX Timestamp.

Implementation Concern	Document Section
Should my firm consider using FAST^{SM} on our upcoming _______ interface?	Usage Considerations
When should I use FAST^{SM}, and when should I rule it out?	Usage Considerations – Usage Scenarios
How should I implement FAST^{SM}?	• Rules of Engagement • Session Management and Control • Field Level Mechanics
How can I tune my FAST^{SM} implementation?	Special Topics

FAST℠ Usage Considerations

In order for FAST℠ efficiencies to be achieved, certain conditions must exist:

- Message types and message structure must be known in advance,
- Message content must conform to template-defined value ranges (domains?),
- The protocol must be implemented by partners to "rules of engagement" to ensure complete compatibility of encoding and decoding operations

FAST℠ protocol performance is impacted by several important factors, the major ones being

- data affinity within and across message types, and
- predictability of data in the message stream.

Data Affinity

FAST℠ compression rates and latency reduction potential are primarily derived from patterns within and across a text (ASCII) data stream which – when known to be systematic - can be reduced to a smaller electronic representation.

The reduction in the physical representation of this data is achieved by

1) Reducing ASCII text fields to a smaller binary, sbit-encoded format,
2) Using field operators – functional transforms - to reduce the storage footprint of fields that occur repeatedly in the stream (i.e. contain patterns of functionally reducible content such as delta patterns).
3) Using message templates to drive program logic and storage of the data required to continuously encode or decode the stream.

BeginStr [8]	SeqNum [34]	SenderID [49]	SendingTime [52]	Price [44]	Symbol [55]
FIX4.4	10000	CLIENT1	20080915-11:02:00.100	120000	SYM1
FIX4.4	10001	CLIENT1	20080915-11:02:00.101	120100	SYM1
FIX4.4	10002	CLIENT1	20080915-11:02:00.102	119900	SYM1

The diagram above shows how data affinity comes into play in a message stream of a hypothetical FIX participant. The top row contains the FIX field and tag, and the table contains message data. One can easily imagine how to represent the second and third messages as relatively minor transformations from the first message.

Header Fields

8	BeginString	Template Default – this field is only required if multiple FIX Versions are employed
34	MsgSeqNum	Increment – this field is present in the first message only
49	SenderCompID	Copy - the data in the previous message is copied when the field is not present
52	Sending Time	Delta

Body Fields

44	Price	Delta
55	Symbol	Copy

This illustration of the concept of data affinity shows messages of the same type, containing not only the same fields, but the same or similar content in each field of each message. In this example, each of the fields occurs at the same relative position within the message.

This data stream is comprised of highly systematic patterns. One aspect of FAST^{SM} stream compression is achieved by exploiting patterns like these, to reduce the wire representation of message data.

Predictability

The predictability of message types, message structure, and even field content impact the expected latency reduction potential of FAST^{SM}. When the message types and content on an interface follow predictable patterns, it is usually possible to tune the FAST^{SM} compression with message templates that take advantage of these patterns.

Usage Scenarios

Good Scenarios

In general a "good" usage scenario is characterized by high message volume (or the need for bandwidth conservation), a small number of message types, and relatively small variation in field content.

Examples of this scenario might be

- Member Quote Interface
- Allocation Interface (when bandwidth conservation or message size reduction are essential)
- Market Data Interface
- Crossing Interface

In certain cases – for example when message size reduction is crucial – an implementor may choose FAST℠ for internal data representation.

Questionable Scenarios

The expected benefit of a FAST℠ interface is limited when bandwidth conservation is not an issue, when the message interface lacks predictable message types or message formats, or when the message content fails to conform (e.g. uses non ASCII-7 character set, has widely varying field content across messages, etc.).

Questionable Scenarios

- Unmanaged Order Flow Interface
- Allocation Interface (absent capacity issues)
- Multi-Asset Type Security Definition Interface

Rules of Engagement

One of the key purposes for clearly defining Rules of Engagement between interfacing parties, is to ensure that there is agreement about session management and control. This section discusses some of the main concerns in selecting and implementing various session management frameworks.

The sections below discuss how the Rules of Engagement for FIX over FAST℠ message interchange should – at the minimum - describe

- the FIX application version,
- the FIX transport interface,
- the FAST version, and
- the mode and mechanism of template exchange.

Session Management and Control

Template exchange

In theory, a FIX session may contain any FIX message type defined in a given FIX protocol version (with or without custom message types or tags), while a FAST℠ session cannot reasonably take place without some prior exchange of information to enable protocol negotiation and message exchange.

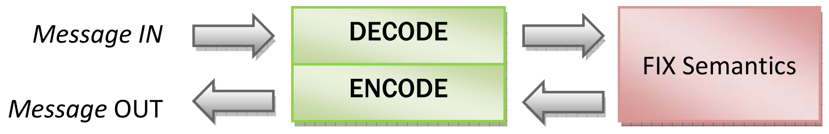
In practice, FAST℠ messages cannot "exist" without some form of pre-session template exchange to describe message type and message content, whether message templates are hard-coded, exchanged out-of-band, or exchanged dynamically.

The ordinary mechanism for FAST^{SM} message template exchange is expected to be either out-of-band, or pre-session. FAST^{SM} implementations are normally over UDP or other unreliable delivery protocol, which makes dynamic template exchange a difficult option. If a message with a new template is missed, subsequent messages of that type cannot be decoded.

Appendix 2 presents an example of a FIX-FAST message template.

SCP Conformance Level

Session Control Protocol (SCP) will continue to be utilized as appropriate as an independent protocol layer.



The decode/encode processes will provide communication errors related to message receipt or sending; in the event the encoder or decoder process receives a message it cannot process, an error must be reported, and the message dropped.

In FIX-over- FAST^{SM} messaging, business logic errors are handled by FIX semantics. This includes dealing with sequence number gaps, making retransmission requests, and handling issues with FIX message content or format.

FIX vs. FAST - Message Constructs

Several FAST^{SM} message constructs were created to represent FIX message constructs which might otherwise be difficult to translate directly:

- FIX component blocks may be represented as static reference templates in FAST
- FAST^{SM} sequences are used to represent FIX repeating groups

These constructs are useful both with respect to bandwidth savings and template management.

Additionally, FAST^{SM} groups may be useful for compressing a number of optional fields at once.

Field Level Mechanics

General Guidelines for Converting

The matrix below provides recommended guidelines for encoding FIX Data Types when using the FAST Protocol. These recommendations are meant to embody best practices to the extent possible under various scenarios. Guidelines notwithstanding, FAST implementors should consistently strive for a system solution that maximizes interoperability as well as efficiency.

The table is arranged with FIX data types down the left side of the matrix. The major data types are in the grey header row and the associated minor data types following. The FAST Date Types are across the top of the matrix. For each FIX minor data type, FAST^{SM} data type columns are encoded with:

R	Recommended Format
Blank	Not Recommended

Technically, a FIX message may be encoded entirely as a String or Byte Vector. This type of encoding format could be used for low volume messages where greater efficiencies would not be realized with a more complex encoding schema. However, implementers are expected to avail themselves and their interfacing partners of the well-known advantages and opportunities provided by the FAST Protocol, and resist any inclination to select data mappings or field encoding rules which are inconsistent with the spirit of the underlying protocols.

When identifying the FIX format of a field, the latest definition of the field should be used. Fields that exist in FIX 4.0 are, in some cases, redefined in later FIX versions. This generally applies to strings and enumerated lists, but this rule should be applied in all cases.

The information in this table is based on FAST^{SM} v1.2 specifications. Please refer to the FAST Version 1.2 Extension document for guidelines in interpreting the latest FAST^{SM} data types and their representation in FAST^{SM} templates.

FIX DATA TYPES		FAST DATA TYPES									
Data Type	Format Comments	Integer		Scaled Decimal	ASCII String	Byte Vector	Enum	Set	Time Stamp	Boolean
		signed	unsigned							
Integer - One byte field with optional sign field in the front										
Int	Integer Fields	R								
Length	must be positive		R							
NumInGroup	Number of repeating values in a group		R							
TagNum	FIX field tag numbers – must be positive and not contain leading zeroes		R							
SeqNum	Message sequence number – value must be positive		R							
Char Fields - any single character or punctuation, except for the delimiter										
Char							R			
Boolean	Values 'Y' or 'N'									R
String - Alpha-numeric free format fields										
String					R					
Data	Encoded Fields					R				

FIX DATA TYPES		FAST DATA TYPES								
Data Type	Format Comments	Integer		Scaled Decimal	ASCII String	Byte Vector	Enum	Set	Time Stamp	Boolean
		signed	unsigned							
MultipleCharValue	One or more single char space delimited values							R		
MultipleStringValue	One or more space delimited string values							R		
Country	ISO 3166 2-char code						R			
Currency	ISO 4217 3-char codes						R			
Exchange	ISO 10383 4-char code						R			
Day-of-month	Format: DD		R							
Month-year[2]	Valid Formats YYYYMM YYYYMMDD YYYYMMWW YYYY = 0000-9999 MM = 01-12 DD = 01-31 WW = w1-w5								R	
LocalMktDate	Format: YYYYMMDD								R	
TZTimeOnly	HH:MM[:SS][Z | [+ | - hh[:mm]]] Examples: 07:39Z 07:39 UTC 02:39-05 five hours behind UTC 15:39+08 eight hours ahead of UTC								R	
TZTimestamp	Based on ISO 8601, format is YYYYMMDD-HH:MM:SS[Z|+/- hh:mm]]]								R	
UTCTimeOnly	Format: [GMT][+/-][OFFSET] Where: GMT HH:MM +/- Required if offset from GMT Offset HH:MM is the offset from GMT								R	
UTCDateOnly	YYYYMMDD								R	

[2] Open Issue - Confirm the existing timestamp routines are able to support the MM, DD, YYYY, and WW formats.

FIX DATA TYPES		FAST DATA TYPES								
Data Type	Format Comments	Integer		Scaled Decimal	ASCII String	Byte Vector	Enum	Set	Time Stamp	Boolean
		signed	unsigned							
UTCTimestamp	YYYYMMDD-HH:MM:SS (whole seconds) or YYYYMMDD-HH:MM:SS.sss								R	
Float - Sequence of digits with optional decimal point and sign character Note about Quantity Fields Select the smallest format that accommodates the data; Signed Int if the value can be negative, Scaled Decimal if there are fractional units, or Unsigned Int if the values are whole positive units.										
Amt	Value typically represents Price*Qty			R						
Qty or Quantity	Float value – either a whole number or decimal value	R	R	R						
Price	May be negative			R						
PriceOffset	May be negative			R						
Percentage	Float			R						
Pattern – Fields that use a base FIX data type and a pattern data type Note about TENOR Fields When possible, ENUMs should be used; if the base value is static for the field, this should be identified in the template and the field coded as a Scaled Decimal for the variable portion. For encoding YEARS, a SCALED DECIMAL can be used if the field cannot be contained in an ENUM.										
Tenor[i]	Used to allow the expression of FX standard tenors in addition to the base valid enumerators. Dx Days D5 Mx Months M3 Wx Weeks W13 Yx Years Y1						R			
Reserved100Plus										
Reserved1000Plus										
Reserved4000Plus										

Special Topics

This section surveys protocol extension proposals and various optimization approaches that have been used by FPL members and participants.

Optimization

- Dynamic vs. Static Template exchange
- Arbitrary field order in FIX messages
- Implicit values (FIX body length, checksum, etc)

Template Exchange

The FAST^{SM} protocol specification provides a great deal of flexibility with respect to message template exchange. Templates may be exchanged in-band, out-of-band, in-stream, or even by a prior agreement between interfacing parties.

For most purposes, participants have chosen out-of-band template exchange. From a practical standpoint, this method is the least risky method to use for all parties. It permits the message encoder and the recipient to plan ahead, test the interface extensively, and forecast encoding and decoding performance reliably. Out-of-band message template exchange also permits parties to mitigate the risk of an unreliable transmission medium such as UDP.

That said, there are still many proponents of dynamic template exchange, and many approaches to optimization methods.

Support for dynamic template exchange may – in some cases – reduce the need for complex or massive template sets. Dynamic templates also provide interfacing parties with an unparalleled flexibility to adapt to unexpected or unforeseen messaging scenarios.

Obviously the flexibility comes at a cost – both in feature development resource consumption and operational risk.

Members have taken several approaches to processing dynamic templates from lightweight script pre-processing, to generating runnable byte code on-the-fly.

Before electing to support dynamic template exchange, implementers should discuss and contrast available options.

Arbitrary Field Ordering in FIX messages

There is general consensus amongst practitioners that – in reality – FIX-over-FAST^{SM} encoded messages are not usually decoded as FIX messages on the receiving end. Even though FIX protocol semantics still apply (for example, a POSSDUP flag would trigger specific business logic at the receiving end), some or all of the efficiencies gained with FAST would be lost if the decoding process also included a process to reformat the message back into a fully formed FIX message.

However, on the encoding side, the original message payload may well be extracted from a FIX message stream, generated by a FIX engine of some sort.

Despite the fact that certain tags or message components must precede or follow other tags or components, FIX field ordering is not predetermined. In general, the FIX protocol permits arbitrary field ordering in FIX messages. Even though FIX engine vendors use specific logic to compose FIX messages, field ordering within each message may vary – for the same message type – for a number of reasons.

Since FAST^{SM} requires a pre-determined field order, practitioners have taken one of two approaches to dealing with FIX field ordering issues:

1) Develop an optimized message format for each message type and pre-process each message to coerce it into the expected format for subsequent FAST^{SM} encoding.
2) Permit message variations through multiple templates for a single message, or by employing dynamic templates to make format adjustments in-stream.

Regardless of the solution selected, the implementor should be aware that decisions must be made early on with respect to message format definitions and template strategy.

Implicit values (FIX body length, checksum, etc.)

As discussed in the preceding section, there is a chance that implementors processing messages which come directly from a FIX engine may wish to reconstitute the FIX message at the receiving end. Should this be the case, thery will also need to re-create the message envelope and “signature” (header, trailer, checksum, etc.) for verification by the recipient.

Message Checksum

All other considerations aside, a message checksum may be a piece of information that implementers wish to use for verification purposes. However, implementers should first agree on how the checksum will be computed and used. The FIX checksum is uniquely associated with the original message.

Appendix 1 – Backwards Compatibility with Older Versions of FAST^{SM}

The following data types were added in FAST 1.2 to support FIX over FAST:

- Enumeration
- Set
- Timestamp
- Boolean

All four data types above can be mapped to the unsigned integer data type. They can therefore be emulated in a 1.1 implementation with some extra encoding and decoding logic above the FAST encoding layer. It is thus possible to support FIX over FAST with a FAST 1.1 implementation.

The Set data type may result in values that are larger than the 64 bit integer data type defined in FAST 1.1. This can be viewed as an implementation restriction when using a FAST 1.1 implementation to interact with a FAST 1.2 implementation.

The bitgroup data type can also be mapped directly to the unsigned integer data type. As with the Set data type, a Bitgroup may be larger than the 64 bit integer data type.

APPENDIX 2 – SAMPLE FIX OVER FAST℠ TEMPLATE

```xml
<?xml version="1.0"?>
<templates xmlns="http://www.fixprotocol.org/ns/template-definition">

<!--                                                  -->
<!-- Convenience defines as per FAST 1.2 spec         -->
<!--                                                  -->

  <define name="MDEntryType" id="269">
     <element name="0"/>
     <element name="1"/>
     <copy/>
  </define>

  <define name="MDEntryBitGroup">
    <bitGroup>
       <enum name="MDUpdateAction" id="279">
         <element name="0"/>
         <element name="1"/>
         <element name="2"/>
         <copy/>
       </enum>
       <field name="MDEntryType"/>
     </bitGroup>
  </define>

  <define name="NoMDEntries" id="268"> <length/> </define>

<!-- Reference templates to be included by other      -->
<!-- templates. From the FIX perspective, reference   -->
<!-- templates are roughly equivalent to component    -->
<!-- blocks.                                          -->
<!--                                                  -->

<!-- Standard Header contains common information      -->
<!-- shared accross all templates. Dictionary scope   -->
<!-- of this template is global.                      -->
<!-- Note that MsgType field is NOT defined within    -->
<!-- StandardHeader template; instead, it is defined  -->
<!-- as a constant field with an appropriate value    -->
<!-- within a corresponding standalone template thus  -->
<!-- yielding performance benefits (MsgType is never  -->
<!-- sent on the wire nor it is encoded/decoded) and  -->
<!-- over network nor is it encoded/decoded) while    -->
<!-- still conforming to the FIX spec as the MsgType  -->
<!-- is defined as the very first field preceding the -->
<!-- rest of the Header fields.                       -->

  <template name="StandardHeader">
    <string name="AppVerID"     id="8">  <constant value="FIX.5.0.SP1"/> </string>
    <string name="SenderCompID" id="49"> <constant value="Company Name"/> </string>
    <uInt64 name="MDEntryID"    id="278" presence="optional"/>
    <uInt64 name="MsgSeqNum"    id="34">  </uInt64>
    <timestamp name="Receive Time" unit="day" id="273"> </timestamp>
  </template>
```

```
<!--                                                 -->
<!-- Instrument template contains instrument related  -->
<!-- information.                                     -->
<!--                                                 -->
  <template name="Instrument">
    <string name="SecurityIDSource"  id="22">  <constant value="122"/> </string>
    <string name="SecurityID"        id="48">  <copy/> </string>
    <string name="MDMkt"             id="275"> <copy/> </string>
  </template>

<!--                                                 -->
<!-- Instrument Sequence Template contains a sequence -->
<!-- of Instruments.                                  -->
<!--                                                 -->
   <template name="InstrumentSeq">
     <sequence name="Instruments">
       <length name="NoInstruments" id="146"/>
       <group name="Instrument">
         <templateRef name="Instrument"/>
       </group>
     </sequence>
   </template>

<!--                                                 -->
<!-- Common Market Data Entry Reference Templates     -->
<!--                                                 -->
<!--                                                 -->
<!-- Sequence defining a repeating group of Market    -->
<!-- entries for Market Data Request.                 -->
<!--                                                 -->
   <template name="MDEntryReqSeq">
     <sequence name="MDEntries">
        <field name="NoMDEntries"/>
        <field name="MDEntryType"/>
     </sequence>
   </template>

<!--                                                 -->
<!-- Market Data Entry Response Template contains     -->
<!-- common Market Data Entry fields for all types.   -->
<!--                                                 -->
   <template name="MDEntryResp">
     <decimal name="MDEntryPX"  id="270"   presence="optional"> <copy/> </decimal>
     <uInt32 name="MDEntrySize" id="271"   presence="optional"> <copy/> </uInt32>
   </template>

<!--                                                 -->
<!-- Market Data Entry Response Level2 contains       -->
<!-- extra fields for Level2 data.                    -->
<!--                                                 -->
  <template name="MDEntryRespLevel2Extension">
    <uInt32 name="MDEntryPositionNo" id="290" presence="optional"> <copy/> </uInt32>
    <uInt32 name="NumberOfOrders"    id="346" presence="optional"> <copy/> </uInt32>
  </template>
```

```
<!--                                                 -->
<!-- Market Data Entry for Incremental Refresh       -->
<!-- contains common MDEntry fields and, optionally, -->
<!-- additional data applicable only to Level1 or    -->
<!-- Level2. Since Level1 and Level2 are specified   -->
<!-- as FAST group entities, and these groups are    -->
<!-- mutually exclusive, using this template for both -->
<!-- level yields both performance and flexibility   -->
<!-- benefits.                                       -->
<!--                                                 -->
  <template name="MDEntryInc">
    <field name="MDEntryBitGroup"/>
    <group name="Level2" presence="optional">
      <templateRef name="MDEntryRespLevel2Extension"/>
    </group>
    <templateRef name="MDEntryResp"/>
    <group name="Instrument" presence="optional">
      <templateRef name="Instrument"/>
    </group>
  </template>

<!--                                                 -->
<!-- Sequence of MDEntryInc entries                  -->
<!--                                                 -->
  <template name="MDEntryIncSeq">
    <sequence name="MDEntries">
       <field name="NoMDEntries"/>
       <templateRef name="MDEntryInc"/>
    </sequence>
  </template>

<!--                                                 -->
<!-- Market Data Entry for Snaphot Full Refresh      -->
<!-- is used for both Level1 and Level2 data.        -->
<!--                                                 -->
  <template name="MDEntryFull">
    <field name="MDEntryType"/>
    <templateRef name="MDEntryResp"/>
    <group name="Level2" presence="optional">
      <templateRef name="MDEntryRespLevel2Extension"/>
    </group>
  </template>

<!--                                                 -->
<!-- Sequence of MDEntryFull entries                 -->
<!--                                                 -->
  <template name="MDEntryFullSeq">
    <sequence name="MDEntries">
      <field name="NoMDEntries"/>
      <templateRef name="MDEntryFull"/>
    </sequence>
  </template>
```

```
<!--                                                  -->
<!-- Standalone data templates are defined below.     -->
<!-- Unlike a reference template, each template is    -->
<!-- identified by additional "id" attribute which    -->
<!-- will be used to encode/decode FAST data.         -->
<!-- Each template is used for all asset types and    -->
<!-- market depth levels.                             -->
<!--                                                  -->

<!--                                                  -->
<!-- Market Data Incremental Refresh template         -->
<!-- MsgType="X"                                      -->
<!--                                                  -->
  <template name="MDIncRefresh" id="30" dictionary="template">
    <string name="MsgType" id="35">  <constant value="X"/> </string>
    <templateRef name="StandardHeader"/>
    <templateRef name="MDEntryIncSeq"/>
  </template>

<!--                                                  -->
<!-- Market Data Snapshot Full Refresh template       -->
<!-- MsgType="W"                                      -->
<!--                                                  -->
  <template name="MDFullRefresh" id="32" dictionary="template">
    <string name="MsgType" id="35"> <constant value="W"/> </string>
    <templateRef name="StandardHeader"/>
    <string name="MDReqID" id="262" presence="optional"/>
    <group name="Instrument">
       <templateRef name="Instrument"/>
    </group>
    <templateRef name="MDEntryFullSeq"/>
  </template>

<!--                                                 -->
<!-- Heartbeat - used for monitoring connection      -->
<!-- status.                                         -->
<!-- MsgType="0".                                    -->
<!--                                                 -->
  <template name="Heartbeat" id="99" dictionary="template">
    <string name="MsgType" id="35">   <constant value="0"/> </string>
    <templateRef name="StandardHeader"/>
    <uInt32 name="HB Interval" id="108" presence="optional"/>
    <timestamp name="Timestamp" id="779" presence="optional"> <delta/> </timestamp>
  </template>

</templates>
```

[i] **DISCUSSION NOTE ABOUT TENOR FIELDS**

(*Lisa Taikitsadaporn*, BrookPath Partners)

The "tenor" concept is represented in the SettlType (63) field of FIX.
This field has what we call a union data type right now of a defined set of enum values:
0 - Regular / FX Spot settlement (T+1 or T+2 depending on currency)
1 - Cash (TOD / T+0)
2 - Next Day (TOM / T+1)
3 - T+2
4 - T+3
5 - T+4
6 - Future
7 - When And If Issued
8 - Sellers Option
9 - T+5
B - Broken date - for FX expressing non-standard tenor, SettlDate (64) must be specified C - FX Spot Next settlement (Spot+1, aka next day)

or the pattern:
Dx = FX tenor expression for "days", e.g. "D5", where "x" is any integer > 0 Mx = FX tenor expression for "months", e.g. "M3", where "x" is any integer > 0 Wx = FX tenor expression for "weeks", e.g. "W13", where "x" is any integer > 0 Yx = FX tenor expression for "years", e.g. "Y1", where "x" is any integer > 0

When we added the pattern to support FX tenors we thought about defined enums list as well, but like you said, years can be infinite in theory.

There are no other valid base for tenors other than the ones already described, therefore I do not foresee expansion beyond days, weeks, months and years.

It is definitely used in quoting or streaming prices, particularly for forward contracts. I am not sure what you mean by "how it might be used in the future". The "tenor" concept is a well established business concept that's not going to change and I believe that the way it is supported now in FIX accommodates what the business needs.

I think you will have to look at the Dx, Mx, Wx, and Yx pattern as data values in the same way you'd treat Symbol field for example, in addition to the defined enums.