FAST version 1.2 Extension Proposal

2009-01-18

Version 10

Table of Contents

Front Matter 3
Status of this Document 3
Distribution 3
Copyright Notice 3
Abstract 3
Disclaimer 3
New Type Definition Syntax 4
Enumeration (single-value field) 5
Background 5
Proposal 5
Template Syntax 5
Set (multi-value field) 6
Background 6
Proposal 6
Template Syntax 6
Bit Group (packed field) 7
Background 7
Proposal 7
Template Syntax 7
Timestamp Data Type 9
Background 9
Proposal 9
Template Syntax 9
Boolean Data Type 10
Background 10
Proposal 10
Template Syntax 10
Change History 11

Front Matter

Status of this Document

This document specifies proposed extensions to a standards protocol for the FIX community, and requests discussion and suggestions for improvements.

Distribution

Distribution of this document is unlimited.

Copyright Notice

Abstract

This document proposes a number of extensions to FAST 1.1, and aims to serve as a basis for a discussion within mdowg (market data optimization working group) about the pros and cons of different aspects of the proposed extensions.

Disclaimer

New Type Definition Syntax

We propose a new syntax for type definitions that allows naming of types so that a type can be specified once and then be used repeatedly. The other proposed extensions in this document can be used without the new type definition syntax.

We propose to add the constructs <define>, <field> and <type>. Define accepts the normal type elements without the name attribute that is used in type elements when declaring fields in a template.

A plain string is defined as:

```
<define name="MyString"> <string/> </define>
```

A string with copy operator is defined as:

```
<define name="CopyString"> <string> <copy/> </string> </define>
```

An enumeration with two values, Red and Green, is defined as:

```
<define name="Colors">
  <enum>
    <element name="Red"/>
    <element name="Green"/>
  </enum>
</define>
```

To use these definitions in a template you then specify:

```
<template name="MyTpl">
  <field name="Foo"> <type name="MyString"/> </field>
  <field name="Bar"> <type name="CopyString"/> </field>
  <field name="Colors"> <type name="Colors"/> </field>
</template>
```

The content of a field may in itself be a type declaration:

```
<field name="Foo"> <string/> </field>
```

In order to be backwards compatible with the current template syntax we let:

```
<nnn name="xxx"> <zzz/> </nnn>
```

be syntactic sugar for:

```
<field name="xxx"> <nnn> <zzz/> </nnn> </field>
```

So:

```
<string name="Foo"> <copy/> </string>
```

translates to:

```
<field name="Foo"> <string> <copy/> </string> </field>
```

Enumeration (single-value field)

Background

FIX as well as other protocols make use of fields with a fixed set of alternative values. These fields may either be single-value (an enumerated value) or multi-value (a set of values).

Proposal

We propose to add enumerations to the standard in order to support single-value fields with a fixed number of values. The enum proposal does not require any change to the base representation as SBIT-encoded fields are used to encode enum field values.

For example, a field can take one of four values, M1, M2, M3 and M4. The four values are assigned the encoded values 0, 1, 2 and 3 respectively. The field can then have any one of the encoded values 0-3.

If the field is nullable, then NULL is assigned the encoded value 0 (zero) analogously with how nullable integers are defined. In the example above with an enumerated field with a range of four values, the encoded values would then be 1, 2, 3 and 4 respectively with zero reserved for the NULL value.

An enum can be used in a bit group (defined in later in this document) in which case it will be encoded with the minimum number of bits needed to represent all elements in the enum definition.

Template Syntax

Variant 1 – using an anonymous enum

```
<template name="t1">
  <enum name="MatchType">
    <element name="M1"/>        assigned the encoded value 0
    <element name="M2"/>        assigned the encoded value 1
    <copy/>
  </enum>
</template>
```

Variant 2 – with a separate type definition of a named enum data type

```
<define name="MatchTypeEnum">
  <enum>
    <element name="M1"/>
    <element name="M2"/>
    <copy/>
  </enum>
</define>

<template name="t3">
  <field name="MatchType"> <type name="MatchTypeEnum"/> </field>
</template>
```

Set (multi-value field)

Background

FIX as well as other protocols make use of fields with a fixed set of alternative values. These fields may either be single-value (an enumerated value) or multi-value (a set of values).

Proposal

We propose to add sets to the standard in order to support multi-value fields with a fixed number of values. The set proposal does not require any change to the base representation as SBIT-encoded fields are used to encode multi-value field values.

For example, a field can have one or more of four values A, B, C and D. The four values are assigned values of 1, 2, 4 and 8, i.e. each value is represented by a different bit. The values can then be added together to form combinations of the four values. For Example, A and C will yield 1 + 4 = 5 as the combined encoded field value.

If the field is nullable, then NULL is assigned the encoded value 0 (zero) analogously with how nullable integers are defined. A set can be used in a bit group in which case it will be encoded with the minimum number of bits needed to represent all elements in the set definition.

Template Syntax

Variant 1 – using an anonymous set

```
<template name="t1">
  <set name="TradeCond">
    <element name="A"/>     assigned the encoded value 1
    <element name="B"/>     assigned the encoded value 2
    <copy/>
  </set>
</template>
```

Variant 2 – using a separate type definition of a named set data type

```
<define name="TradeCondSet">
  <set>
    <element name="A"/>
    <element name="B"/>
    <copy/>
  </set>
</define>

<template name="t3">
  <field name="TradeCond"> <type name="TradeCondSet"/> </field>
</template>
```

Bit Group (packed field)

Background

FIX as well as other protocols frequently include fields with small value ranges. The minimum standard length of a field in FAST 1.1 is 8 bits including the stop bit.

Proposal

We propose to add a bit group type to the FAST standard. A bit group packs two or more fields into one SBIT-encoded field. Each of the bit fields will have a fixed size that is less than 8 bits. Supported bit field types are enums, sets, and 2-7 bit signed integers and 1-7 bit unsigned integers. The new integer types are named int2-int7 and uInt1-Uint7. The one bit version of signed int is excluded as it makes no sense to have an integer quantity with a sign bit only.

A bit group will use the ordinary wire representation of an SBIT-encoded field. Fields within the group will be assigned space from left to right. For example; a bit group with three fields, A (2 bits), B (1 bit) and C (1 bit), will occupy a total of four bits or one SBIT-encoded byte. The fields will be placed within the byte as: 'SAABCxxx', where S is the SBIT and xxx are three unassigned bits that must be set to zero. Enum and Set fields in a bit group will be encoded with the minimum number of bits needed to represent all values of the type.

A bit group can use multiple bytes and an individual field in a bit group may span the boundary between two bytes. Let's assume that a bit group contains three fields; A (5 bits), B (6 bits) and C (3 bits). The first two bits of field B will be assigned to the two rightmost data bits in the first byte while the remaining four bits will be assigned to the leftmost four data bits in the second byte of the bit group wire representation.

Template Syntax

Variant 1 – using an anonymous bit group

```
<template name="t1">
  <bitGroup name="LevelInfo">
    <enum name="UpdateAction">     will use 2 bits
      <element name="0"/>
      <element name="1"/>
      <element name="2"/>
    </enum>
    <enum name=EntryType">         will use 1 bit
      <element name="0"/>
      <element name="1"/>
    </enum>
    <enum name="CFICode">          will use 1 bit
      <element name="OC"/>
      <element name="OP"/>
    </enum>
    <uInt3 name="PriceLevel"/>     will use 3 bits - range is 0-7
    <copy/>
  </bitGroup>
</template>
```

Variant 2 – using a separate type definition of a named bit group data type

```
<define name="UpdateActionEnum">
  <enum>
    <element name="0"/>
    <element name="1"/>
    <element name="2"/>
```

```
  </enum>
</define>
<define name="EntryTypeEnum">
  <enum>
    <element name="0"/>
    <element name="1"/>
  </enum>
</define>
<define name="CFICodeEnum">
  <enum>
    <element name="OC"/>
    <element name="OP"/>
  </enum>
</define>
<define name="LevelInfoBitGroup">
  <bitGroup>
    <field name="UpdateAction"> <type name="UpdateActionEnum"/> </field>
    <field name="EntryType">    <type name="EntryTypeEnum"/>    </field>
    <field name="CFICode">      <type name="CFICodeEnum"/>      </field>
    <field name="PriceLevel">   <uInt3/>                        </field>
    <copy/>
  </bitGroup>
</define>
<template name="t2">
  <field name="LevelInfo"> <type name="LevelInfoBitGroup"/> </field>
</template>
```

Timestamp Data Type

Background

FIX has several time related data types. The FAST specification should therefore provide native support for timestamp fields.

Proposal

We propose to add a timestamp type to the FAST standard. A timestamp is an integer that represents a number of time units since an epoch. Negative numbers can be used to define timestamps before the chosen epoch.

The default epoch is the UNIX epoch (1970-01-01 00:00:00 GMT). It is possible to specify a different epoch. As a special case, an epoch of `"today"` yields a timestamp value that represents time units since midnight, providing a convenient representation of FIX UTCTimeOnly.

The supported time units are; day, second, millisecond, microsecond and nanosecond.
The default time unit is milliseconds.

By supporting different time units and different epochs, we can efficiently represent timestamps with a granularity that meets different application level needs. For example; FIX UTCTimeOnly and UTCDateOnly can be efficiently represented when a full timestamp is not needed.

No new wire format is needed. The existing SBIT-encoded integer type is used to represent all variants of the timestamp field type.

Template Syntax

A list of the supported units

```
<define name="t0"> <timestamp unit="day"/> </define>
<define name="t1"> <timestamp unit="second"/> </define>
<define name="t2"> <timestamp unit="millisecond"/> </define>
<define name="t3"> <timestamp unit="microsecond"/> </define>
<define name="t4"> <timestamp unit="nanosecond"/> </define>
```

A unit of "day" corresponds to UTCDateOnly

```
<define name="t5"> <timestamp unit="day"/> </define>
```

An epoch of "today" corresponds to UTCTimeOnly

```
<define name="t6"> <timestamp unit="second" epoch="today"/> </define>
```

The default epoch is the UNIX epoch and the default unit is millisecond

```
<define name="t7"> <timestamp/> </define>
```

Boolean Data Type

Background

The FIX specification contains a number of boolean fields. The FAST specification should therefore provide native support for boolean fields.

Proposal

We propose to add a boolean data type which has one of two possible values, true or false. A boolean field is represented as an unsigned integer with the value 0 meaning false and the value 1 meaning true. A boolean value can be a member of a bit group in which case it will use one bit (0/1). A boolean field with optional presence will use a nullable representation analogously with a nullable integer representation. NULL will be represented as a 0 (zero), false as 1 (one) and true as 2 (two). A nullable boolean field will use 2 bits in a bit group.

Template Syntax

Variant 1 – using a boolean directly

```
<template name=”t1”>
   <boolean name=”UpdatesLastPaid” presence=”optional”/>
</template>
```

Variant 2 – using a defined boolean type

```
<define name=”UpdatesLastPaidFlag”> <boolean/> </define>

<template name=”t2”>
  <field name=”UpdatesLastPaid” presence=”optional”>
    <type name=UpdatesLastPaidFlag”/>
  </field>
</template>
```

Change History

Date	Version	Author	Description
2009-01-17	9	Rolf Andersson	Changes following review by the mdowg members
2009-01-18	10	Rolf Andersson	Release version for submission to the GTC