FIX TagValue Encoding Technical Specification

Version 1.0 – Technical Standard – June 2020

THIS DOCUMENT IS THE FINAL VERSION OF A FIX TECHNICAL STANDARD. THIS VERSION HAS BEEN APPROVED BY THE GLOBAL TECHNICAL COMMITTEE AS THE FINAL STEP IN CREATING A NEW FIX TECHNICAL STANDARD OR A NEW VERSION OF AN EXISTING FIX TECHNICAL STANDARD. POTENTIAL ADOPTERS ARE STRONGLY ENCOURAGED TO USE ONLY THE FINAL VERSION. EXISTING ADOPTERS ARE STRONGLY ENCOURAGED TO UPGRADE TO THE FINAL VERSION.

© Copyright, 2011-2020, FIX Protocol, Limited

Table of Contents

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 FIX 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 FIX WORKING DRAFTS AS REFERENCE MATERIAL OR TO CITE THEM AS OTHER THAN “WORKS IN PROGRESS”. THE FIX GLOBAL TECHNICAL COMMITTEE WILL ISSUE, UPON COMPLETION OF REVIEW AND RATIFICATION, AN OFFICIAL STATUS (“APPROVED”) OF/FOR 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), except as expressly set out in FIX Protocol Limited’s Copyright and Acceptable Use Policy.

© Copyright 2003-2020 FIX Protocol Limited, all rights reserved

FIX Technical Standard Specifications by FIX Protocol Ltd. are licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License. Based on a work at https://github.com/FIXTradingCommunity/.

1 Scope

The Financial Information eXchange tagvalue encoding is the original encoding used for FIX messages. The tagvalue encoding is the encoding used by the FIX session layer; it corresponds to the Presentation Layer of the ISO Open Systems Interconnection model. The encoding uses an integer number known as a *tag* to identify the field, followed by the "=" character (hexadecimal 0x3D), then the value of that field encoded in the ISO 8859-1 character set. Each tagvalue pair is separated by the *Start of Heading* control character `<SOH>` (hexadecimal value 0x01), which is defined by ISO 6429:1992. The tagvalue encoding also supports the encoding of binary and multibyte character data in certain encoded data fields that are preceded by a Length field.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

— ISO/IEC 8859-1:1998 *Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1*

— ISO/IEC 6429:1992 *Information technology -- Control functions for coded character sets*

— ISO/IEC 11404:2007 *Information technology -- General-Purpose Datatypes (GPD)*

— *Extensible Markup Language (XML) 1.0 (Fifth Edition)* W3C Recommendation 26 November 2008

— IETF RFC 2119 -- *Key words for use in RFCs to Indicate Requirement Levels* March 1997

— IETF RFC 2978 -- *IANA Charset Registration Procedures* October 2000

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 11404:2007 *Information technology -- General-Purpose Datatypes (GPD)* and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— IEC Electropedia: available at http://www.electropedia.org/

— ISO Online browsing platform: available at https://www.iso.org/obp

3.1 General terms and definitions

3.1.1 field presence

The existence or use of a field within a message. FIX specifications and rules of engagement based on FIX should refer to a field as being present in a message.

3.1.2 component presence

The existence or use of a component within a message. FIX specifications and rules of engagement based on FIX should refer to a component as being present in a message.

3.1.3 repeating group instance

A specific record, as defined in ISO/IEC 11404:2007, of the group within a repeating group.

3.1.4 character digit

The character representation of a number, 0 through 9, in the character set used for encoding. Characters 0x30 through 0x39 in the Latin alphabet No. 1 character set (ISO/IEC 8859-1:1998).

4 FIX tagvalue message syntax

4.1 Character encoding

With the exception of datatype *data*, tagvalue encoding uses a single-byte character set. By default, the encoding is ISO/IEC 8859-1:1998 Latin alphabet No. 1.

By counterparty agreement, a different single-byte character set may be used.

Note that the Latin-1 alphabet is an 8-bit code but reserves two ranges for control codes. Message structure is supplemented by ISO/IEC 6429:1992 control character set C0.

4.2 Field syntax

4.2.1 Tag (field identifier)

Each field is uniquely identified by an integer, known as a tag. Tags must be unique among both session and application message fields. (Fields in the standard header and standard trailer components are shared by session and application messages.)

Tags are serialized according to the syntax of the *TagNum* datatype.

4.2.2 Tag delimiter

A tag is delimited from its field value by the equals sign (=), character value 61 (decimal).

4.2.3 Field value

Field values are serialized according to their FIX datatype syntax.

4.2.4 Field delimiter

All fields in a FIX message, including those of datatype *data*, must be terminated by a delimiter character. The *Start of Heading* control character, value 0x01, referred to in this document as `<SOH>`, is used for field termination.

There must be no embedded `<SOH>` characters within field values except for those of datatype *data*.

4.2.5 Well-formed field

A well-formed field has the form:

tag=value`<SOH>`

A field shall be considered malformed if any of the following occurs as a result of encoding:

- the tag is empty
- the tag delimiter is missing
- the value is empty
- the value contains an `<SOH>` character and the datatype of the field is not *data* or *XMLdata*
- the datatype of the field is *data* and the field is not immediately preceded by its associated Length field.

4.2.6 Example of a FIX tag=value message

The following is a FIX 4.2 NewOrderSingle(35=D) message in classic tagvalue pair format:

```
8=FIX.4.2<SOH>9=251<SOH>35=D<SOH>49=AFUNDMGR<SOH>56=ABROKER<SOH>34=2<SOH>
52=2003061501:14:49<SOH>11=12345<SOH>1=111111<SOH>63=0<SOH>64=20030621<SOH>
21=3<SOH>110=1000<SOH>111=50000<SOH>55=IBM<SOH>48=459200101<SOH>22=1<SOH>
54=1<SOH>60=2003061501:14:49<SOH>38=5000<SOH>40=1<SOH>44=15.75<SOH>15=USD<SOH>
59=0<SOH>10=127<SOH>
```

4.3 Message structure

A FIX message is a collection of fields that begins with the BeginString(8) field, followed by the BodyLength(9) field, then the MsgType(35) field, and ends with the Checksum(10) field. The message is identified by the value provided in the MsgType(35) field.

The following section summarizes general specifications for constructing messages in tagvalue syntax.

The general format of a message is a standard header followed by the message body fields and terminated with a standard trailer.

Each message is constructed of a stream of *tag*=*value* fields with a field delimiter between fields in the stream. Messages will be referenced as *message_name*(35=*x*) with *x* representing the message type; fields will be referenced as *field_name*(*tag*).

4.3.1 Message type

The MsgType(35) field is used to identify the type of message encoded. The definition and scope of the message type is provided by the encoder. For example, the FIX session layer standard defines a set of messages to initiate and manage a FIX session. The FIX application layer standard (commonly referred to as *FIX Latest*) defines additional message types for business level processing. There are no message types or reserved values for message types defined at the encoding level.

4.3.2 Field presence

In a message definition, a field must be specified as either required, optional, or conditionally required. If it is conditionally required, the message specification must give a clear rule for when the field must be present.

All fields present in an encoded message must have a value. Optional fields without values must be omitted from the FIX message.

A tag (field) must appear at most once in a message, except when the tag appears within a repeating group.

A tag (field) must appear at most once per repeating group instance.

4.3.3 Field sequence

Except where noted, fields within a message can be defined in any sequence. (Relative position of a field within a message is inconsequential.) The exceptions to this rule are:

- General message format is composed of the standard header, followed by the body, followed by the standard trailer.
- The first three fields in the StandardHeader component must be BeginString(8), followed by BodyLength(9), followed by MsgType(35), in that sequence.
- The last field in the standard trailer must be CheckSum(10).
- Within a repeating group, field sequence is strictly defined by a group definition.

4.3.4 Message delimiter

Messages are effectively delimited by the `<SOH>` character at the end of the CheckSum(10) field.

All messages must begin with the BeginString(8) field and terminate with the CheckSum(10) field.

4.3.5 Components

Application level messages, representing the FIX application layer, can organize a collection of fields into a set commonly referred to as a component or a submessage. These components can contain sub-components.

The FIX tagvalue encoding does not represent component boundaries in the encoding. Any component boundary is lost during encoding. Further, FIX tagvalue encoding does not require the collection of fields to be ordered, and does not enforce component boundaries around the fields in the encoding.

4.3.6 Groups and repeating groups

In ISO/IEC 11404:2007 terminology, a FIX repeating group is an array of records. An instance of a repeated record is called a repeating group instance in this document.

It is permissible for fields to be repeated within a repeating group. For example, the following represents a repeating group with two repeating instances delimited by tag 372 (first field in the repeating group):

```
384=2<SOH>372=6<SOH>385=R<SOH>372=7<SOH>385=R<SOH>
```

4.3.6.1 Repeating group name

It is recommended that a repeating group be named XXXGrp, e.g. DividendPeriodGrp.

4.3.6.2 NumInGroup field

In tagvalue encoding, repeating group instances are preceded by a count of the number of instances to follow. The count is serialized as a FIX field with a value of datatype *NumInGroup*, commonly referred to as a NumInGroup field.

It is recommended that NumInGroup fields be named NoXXX, e.g. NoContraBrokers(382).

4.3.6.3 Field sequence within a repeating group

- The NumInGroup field (for example: NoTradingSessions(386), NoAllocs(78)), which specifies the number of repeating group instances, occurs once for a repeating group and must immediately precede the repeating group instances.
- Fields within repeating groups must be specified in the order that the fields are specified in the message definition.

4.3.6.4 Field presence within a repeating group

- The NumInGroup field is required and must be larger than zero if the repeating group is required, or if the repeating group is optional and the message contains one or more instances for that repeating group.
- If a repeating group field is specifed as required, then it must appear in every instance of that repeating group.
- If a repeating group is used in a message, its first field (after the NumInGroup field) must be populated in each instance of the repeating group. This allows implementations of the protocol to use the first field as the indicator for the start of a new instance within the repeating group.
- The first field listed after the NumInGroup field may be a component or nested repeating group. In this case, the first field is defined as the first field of the component or the NumInGroup field of the nested repeating group. The component or nested repeating group becomes required for every instance of the outer repeating group.
- The presence of optional or conditionally required fields may vary across repeating group instances.

4.3.6.5 Nested repeating groups

Repeating groups may be nested within another repeating group. Multiple levels of nesting are allowed. In an encoded message, nested repeating groups are serialized as a depth-first tree traversal. That is, all instances of a nested group of the first top-level group instance are encoded before the second instance of the top-level group, and so forth.

Nesting Level	Tag	Field Name	Notes
Start Level 1	453	**NoPartyIDs**	This repeating group is the Parties component in the FIX Standard.
	448	> PartyID	Must always be the first field in the repeating group, and must be provided if NoPartyIDs(453) > 0.
	447	> PartyIDSource	Required if NoPartyIDs(453) > 0.

Nesting Level	Tag	Field Name	Notes
	452	> PartyRole	Required if NoPartyIDs(453) > 0.
	2376	> PartyRoleQualifier	Optional; not required for each repeating group instance.
Start Level 2	802	**> NoPartySubIDs**	This nested repeating group is the PtysSubGrp component in the FIX Standard.
	523	> > PartySubID	Required if NoPartySubIDs(802) > 0.
	803	> > PartySubIDType	Required if NoPartySubIDs(802) > 0.
End Level 2			
End Level 1			

4.3.6.6 Nested repeating group example

The following is an example of a Parties repeating group with three instances, two of which contain nested PtysSubGrp repeating groups. This example also demonstrates that repeating group instances may be heterogeneous, meaning that the fields present in an instance can vary across instances.

```
NoPartyIDs(453)=3
    PartyID(448)=DEU
    PartyIDSource(447)=B          (Bank Identifier Code (BIC) ISO 9362)
    PartyRole(452)=1              (Executing Firm)
    NoPartySubIDs(802)=1
          PartySubID(523)=A1
          PartySubIDType(803)=10  (Securities account number)
    PartyID(448)=104317
    PartyIDSource(447)=H          (CSD Participant Number)
    PartyRole(452)=83             (Clearing Account)
    PartyID(448)=GSI
    PartyIDSource(447)=B          (Bank Identifier Code (BIC) ISO 9362)
    PartyRole(452)=4              (Clearing Firm)
    PartyRoleQualifier(2376)=23   (Firm or legal entity)
    NoPartySubIDs(802)=1
          PartySubID(523)=C3
          PartySubIDType(803)=10  (Securities account number)
```

This example is encoded in FIX tagvalue format as follows:

```
453=3<SOH>448=DEU<SOH>447=B<SOH>452=1<SOH>802=1<SOH>523=A1<SOH>803=10<SOH>448=104317<SOH>
447=H<SOH>452=83<SOH>448=GSI<SOH>447=B<SOH>452=4<SOH>2376=23<SOH>802=1<SOH>523=C3<SOH>
803=10<SOH>
```

4.3.7 Encoded data fields

Tagvalue encoding provides features for embedding fields in any IANA-registered encoding, possibly using multibyte character sets. Such fields must be preceded by an associated Length field that specifies the number of octets in the encoded data field.

4.3.7.1 MessageEncoding field

MessageEncoding(347) is a field in the StandardHeader component that gives the name of an encoding used in a message.

4.3.7.2 Examples of using encoded data fields for Japanese language support

Example 1 – Specify the English value as Issuer plus Japanese character set as EncodedIssuer(349)

Tag	Field Name	Value
…Other standard header fields		
347	MessageEncoding	Shift_JIS
…Other standard header fields		
…Other message body fields		
106	Issuer	HITACHI
348	EncodedIssuerLen	10
349	EncodedIssuer	日立製作所
…Other message body fields		

Example 2 – Specify the English value as Issuer(106) plus Japanese character set as EncodedIssuer(349). Specify the English value as Text(58) plus Japanese character set as EncodedText(357).

Tag	Field Name	Value
…Other standard header fields		
347	MessageEncoding	Shift_JIS
…Other standard header fields		
…Other message body fields		
106	Issuer	HITACHI
348	EncodedIssuerLen	10
349	EncodedIssuer	日立製作所
…Other message body fields		
58	Text	This is a test
356	EncodedTextLen	17
357	EncodedText	これはテストです。
…Other message body fields		

4.3.7.3 Precaution when using multibyte encodings

FIX tagvalue encoding processors must use the Length field associated with the encoded data field when parsing encoded data fields to avoid field truncation and subsequent decoding errors. There is the possibility that one of the octets in a multibyte encoded data field contains the 0x01 value, which can be interpreted by message parsers as the `<SOH>` field delimiter if the associated Length field is not honored.

5 Standard header and trailer

Fields specified in the standard header and trailer serve as delimiters of a message. These fields provide features for message integrity, a body length in the header, and a checksum in the trailer.

5.1 Standard header

Each session or application layer message is preceded by a standard header. The header identifies the message type and length. Higher layers (such as a session layer or an application layer) may extend the definitions of the standard header and trailer by using additional fields, groups, or components.

5.1.1 Body length calculation

The message length must be specified in the BodyLength(9) field. The length must be calculated by counting the number of octets in the message following the end of field delimiter (<SOH>) of BodyLength(9), up to and including the end of field delimiter (<SOH>) of the field immediately preceding the CheckSum(10) field.

5.1.2 Standard header definition

Tag	Field Name	Datatype	Req'd	Comments
8	BeginString	String	Y	FIX.4.2 | FIX.4.4 | FIXT.1.1 BeginString(8) must be the first field in the message.
9	BodyLength	Length	Y	Message length, in octets. BodyLength(9) must be the second field in the message.
35	MsgType	String	Y	Defines message type. MsgType(35) must be the third field in the message.
90	SecureDataLen	Length	N	Length field for SecureData(91). SecureDataLen(90) must be present and unencrypted if SecureData(91) is present in the message.
91	SecureData	Data	N	Encrypted message content. Tag number, separator ("="), and delimiter (<SOH>) must be unencrypted. If present in the message, SecureData(91) must be immediately preceded by SecureDataLen(90).
347	MessageEncoding	String	N	Type of message encoding used in a message's encoded data fields. MessageEncoding(347) must be specified if any fields of datatype *data* are present in the message.

5.2 Standard trailer

5.2.1 Standard trailer definition

Tag	Field Name	Datatype	Req'd	Comments
93	SignatureLength	Length	N	Length field for Signature(89). SignatureLength(93) must be present and unencrypted if Signature(89) is present in the message. SignatureLength(93) must not be included as part of the encrypted content in SecureData(91).
89	Signature	Data	N	If Signature(89) is present, it must be immediately preceded by SignatureLength(93). Signature(89) must not be included as part of the encrypted content in SecureData(91).
10	CheckSum	String	Y	Three-octet character representation of the modulo 256 checksum. Checksum(10) must be the last field in the message. The end of field

Tag	Field Name	Datatype	Req'd	Comments
				delimiter (`<SOH>`) of Checksum(10) serves as the end of message delimiter.

5.2.2 Checksum

The checksum must be calculated by summing the binary value of each octet from the start of the BeginString(8) field up to and including the end of field delimiter (`<SOH>`) of the field immediately preceding the CheckSum(10) field, then transforming this value using a modulo 256.

The calculated modulo 256 checksum must then be encoded as an ISO 8859-1 three-octet representation of the decimal value. For example, if the result of the modulo 256 of the sum of the value of the fields is 23, the CheckSum(10) field will be encoded as the ISO 8859-1 string "10=023". See Annex A for details.

6 FIX tagvalue datatypes

Each field in FIX has a datatype. A datatype is defined as a combination of a value space and a lexical space. Value space is the range of its possible values while lexical space is how those values are represented in a message encoding.

FIX datatypes are shared by session and application messages in tagvalue encoding.

6.1 Value space

The value space of FIX datatypes is shared among all FIX message encodings. Value space of datatypes is defined using the vocabulary in ISO/IEC 11404:2007 *Information technology – General-Purpose Datatypes (GPD)*.

6.2 Lexical space

This specification defines the lexical rules specific to FIX tagvalue encoding. FIX implementations must follow these lexical rules to achieve interoperability.

6.2.1 Character encoding

With exception of fields of datatype *data*, tagvalue encoding uses a single-byte character set. By default, the encoding is ISO/IEC 8859-1:1998 Latin alphabet No. 1. Note that the Latin-1 alphabet is an 8-bit code but reserves two ranges for control codes.

By counterparty agreement, a different single-byte character set may be used.

6.2.2 Lexical encoding for FIX datatypes

Table 1 — FIX datatypes tagvalue encoding

Data Type	**Semantics**	**Value space (ISO/IEC 11404:2007)**	**Tagvalue lexical space**
int	integer number	integer	Sequence of character digits without commas or decimals and optional sign character (characters “-” and “0” - “9”). The sign character utilizes one octet (i.e., positive int is “99999” while negative int is “-99999”). Note that int values may contain leading zeros (e.g. “00023” = “23”).
TagNum	A field’s tag number	ordinal	Sequence of character digits without commas or decimals. Value must be positive and may not contain leading zeros.
SeqNum	A message sequence number.	ordinal	Sequence of character digits without commas or decimals. Value must be positive.
NumInGroup	The number of entries in a repeating group	size	Sequence of character digits without commas or decimals. Value must be positive. Fields of datatype *NumInGroup* are referred to as NumInGroup fields.
DayOfMonth	Day number within a month (values 1 to 31)	integer range 1..31	Sequence of character digits without commas or decimals (values 1 to 31).
float	All float fields must accommodate up to fifteen significant digits. The number	real	Sequence of character digits with optional decimal point and sign character (characters “-”, “0” - “9” and “.”); the absence of the decimal point within the string will be

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
	of decimal places used should be a factor of business/market needs and mutual agreement between counterparties.		interpreted as the float representation of an integer value. Note that float values may contain leading zeros (e.g. “00023.23” = “23.23”) and may contain or omit trailing zeros after the decimal point (e.g. “23.0” = “23.0000” = “23” = “23.”).
Qty	Either a whole number (no decimal places) of “shares” (securities denominated in whole units) or a decimal value containing decimal places for non-share quantity asset classes (securities denominated in fractional units).	Scaled radix=10	Same as float
Price	A price. Note the number of decimal places may vary. For certain asset classes prices may be negative values. For example, prices for options strategies can be negative under certain market conditions.	Scaled radix=10	Same as float
PriceOffset	A price offset, which can be mathematically added to a Price. Note the number of decimal places may vary and some fields such as LastForwardPoints may be negative.	Scaled radix=10	Same as float
Amt	Typically representing a Price times a Qty	Scaled radix=10	Same as float
Percentage	A percentage (e.g. 0.05 represents 5% and 0.9525 represents 95.25%). Note the number of decimal places may	real	Same as float

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
	vary.		
char	A single character. All char fields are case sensitive (i.e. m != M).	character repertoire=8859-1 (Latin-1)[1]	Any character except control characters. By default, ISO/IEC 8859-1 (Latin-1). By counterparty agreement, a different character set may be used.
Boolean	Boolean	boolean	'Y' = True/Yes 'N' = False/No
String	Text. All String fields are case sensitive (i.e., "morstatt" != "Morstatt").	characterstring repertoire=8859-1 (Latin-1)[2]	Alphanumeric free-format strings can include any character except control characters.
MultipleCharValue[3]	Set of character codes	set element = character repertoire=8859-1 (Latin-1)[4]	String containing one or more space-delimited single character values, e.g. "2 A F".
MultipleStringValue[5]	Set of string codes	set element = character string repertoire=8859-1 (Latin-1)[6]	String containing one or more space-delimited multiple character values, e.g. "AV AN A".
Country	External code set ISO 3166-1:2013 Codes for the representation of names of countries and their subdivisions - Part 1: Country codes	array element= character index-lowerbound=1 index-upperbound=2	2-character code
Currency	External code set ISO 4217:2015 Codes for the representation of currencies and funds	array element = character index-lowerbound=1 index-upperbound=3	3-character code
Exchange	External code set ISO 10383:2012 Securities and related financial instruments - Codes for exchanges and	array element = character index-lowerbound=1 index-upperbound=4	4-character code

[1] By counterparty agreement, a different single-byte character set may be used.
[2] By counterparty agreement, a different single-byte character set may be used.
[3] The use of datatype *MultipleCharValue* is no longer permitted for enhancements or additions to the FIX protocol. New message designs should use repeating groups for multiple values instead.
[4] By counterparty agreement, a different single-byte character set may be used.
[5] The use of datatype *MultipleStringValue* is no longer permitted for enhancements or additions to the FIX protocol. New message designs should use repeating groups for multiple values instead.
[6] By counterparty agreement, a different single-byte character set may be used.

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
	market identification (MIC)		
MonthYear	Month and year of instrument maturity or expiration	characterstring	String representing month of a year. An optional day of the month can be appended or an optional week code. **Valid formats:** YYYYMM YYYYMMDD YYYYMMWW **Valid values:** YYYY = 0000-9999; MM = 01-12; DD = 01-31; WW = w1, w2, w3, w4, w5.
UTCTimestamp	UTC date/time	time time-unit = millisecond or up to picosecond by bilateral agreement	String representing time/date combination represented in UTC (Universal Time Coordinated) in either YYYYMMDD-HH:MM:SS (whole seconds) or YYYYMMDD-HH:MM:SS.sss* format, colons, dash, and period required. **Valid values:** YYYY = 0000-9999, MM = 01-12, DD = 01-31, HH = 00-23, MM = 0059, SS = 00-60 (60 only if UTC leap second), sss* fractions of seconds. The fractions of seconds may be empty when no fractions of seconds are conveyed (in such a case the period is not conveyed), it may include 3 digits to convey milliseconds, 6 digits to convey microseconds, 9 digits to convey nanoseconds, 12 digits to convey picoseconds; Other number of digits may be used with bilateral agreement. **Leap Seconds:** Note that UTC includes corrections for leap seconds, which are inserted to account for slowing of the rotation of the earth. Leap second insertion is declared by the International Earth Rotation Service (IERS) and has, since 1972, only occurred on the night of Dec. 31 or Jun 30. The IERS considers March 31 and September 30 as secondary dates for leap second insertion, but has never utilized these dates. During a leap second insertion, a UTCTimestamp field may read "19981231-23:59:59", "19981231-23:59:60", "19990101-00:00:00". (see http://tycho.usno.navy.mil/leapsec.html) **Examples:**

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
			"20011217-09:30:47.123" milliseconds "20011217-09:30:47.123456" microseconds "20011217-09:30:47.123456789" nanoseconds "20011217-09:30:47.123456789123" picoseconds
UTCTimeOnly	UTC time of day	time time-unit = millisecond or up to picosecond by bilateral agreement	String representing time-only represented in UTC (Universal Time Coordinated) in either HH:MM:SS (whole seconds) or HH:MM:SS.sss* (milliseconds) format, colons, and period required. This special-purpose field is paired with UTCDateOnly to form a proper UTCTimestamp for bandwidth-sensitive messages. **Valid values:** HH = 00-23, MM = 00-59, SS = 00-60 (60 only if UTC leap second), sss* fractions of seconds. The fractions of seconds may be empty when no fractions of seconds are conveyed (in such a case the period is not conveyed), it may include 3 digits to convey milliseconds, 6 digits to convey microseconds, 9 digits to convey nanoseconds, 12 digits to convey picoseconds; Other number of digits may be used with bilateral agreement. **Examples:** "13:20:00.123"milliseconds "13:20:00.123456" microseconds "13:20:00.123456789" nanoseconds "13:20:00.123456789123" picoseconds
UTCDateOnly	UTC date	time time-unit = day	Date represented in UTC (Universal Time Coordinated) in YYYYMMDD format. **Valid values:** YYYY = 0000-9999, MM = 01-12, DD = 01-31.
LocalMktDate	Local date	time time-unit = day	Date of local market (as opposed to UTC) in YYYYMMDD format. **Valid values:** YYYY = 0000-9999, MM = 01-12, DD = 01-31.
TZTimeOnly	Time of day with timezone	time time-unit = millisecond or up to picosecond by bilateral agreement	Time represented based on ISO 8601. This is the time with a UTC offset to allow identification of local time and time zone of that time. Format is HH:MM[:SS][Z | [+ | - hh[:mm]]] where HH = 00-23 hours, MM = 00-59 minutes, SS = 00-59 seconds, hh = 01-12

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
			offset hours, mm = 00-59 offset minutes.
TZTimestamp	Date/time with timezone	time time-unit = millisecond or up to picosecond by bilateral agreement	String representing a time/date combination representing local time with an offset to UTC to allow identification of local time and time zone offset of that time. The representation is based on ISO 8601. Format is YYYYMMDD-HH:MM:SS.sss*[Z | [+ | - hh[:mm]]] where YYYY = 0000 to 9999, MM = 01-12, DD = 01-31 HH = 00-23 hours, MM = 00-59 minutes, SS = 00-59 seconds, hh = 01-12 offset hours, mm = 00-59 offset minutes, sss* fractions of seconds. The fractions of seconds may be empty when no fractions of seconds are conveyed (in such a case the period is not conveyed), it may include 3 digits to convey milliseconds, 6 digits to convey microseconds, 9 digits to convey nanoseconds, 12 digits to convey picoseconds; Other number of digits may be used with bilateral agreement. **Examples:** “20060901-07:39Z” is 07:39 UTC on 1st of September 2006 “20060901-02:39-05” is five hours behind UTC, thus Eastern Time on 1st of September 2006 “20060901-15:39+08” is eight hours ahead of UTC, thus Hong Kong/Singapore time on 1st of September 2006 “20060901-13:09+05:30” is 5.5 hours ahead of UTC, thus India time on 1st of September 2006 Using decimal seconds: “20060901-13:09.123+05:30” milliseconds “20060901-13:09.123456+05:30” microseconds “20060901-13:09.123456789+05:30” nanoseconds “20060901-13:09.123456789123+05:30” picoseconds “20060901-13:09.123456789Z” nanoseconds (UTC time zone)
Length[7]	Length of a data field in octets	size	Sequence of character digits without commas or decimals. Value must be positive. Fields of datatype *Length* are referred to as Length fields.

[7] Note that the number of octets in the encoding may be different than the number of characters, such as in the case of multibyte character sets.

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
			The Length field must be associated with a field of datatype data. The Length field must specify the number of octets of the value contained in the associated data field up to but not including the terminating `<SOH>`.
data[8]	Opaque data or variable-length string	A union of two datatypes: octetstring and characterstring repertoire=(value encoded using the encoding specified in the MessageEncoding(347) field	Raw data with no format or content restrictions, or a character string encoded as specified by MessageEncoding(347). Fields of datatype data must have an associated field of type Length. Fields of datatype data must be immediately preceded by their associated Length field.

Tenor		characterstring	Used to allow the expression of FX standard tenors in addition to the base valid enumerations defined for the field that uses this pattern data type. This pattern data type is defined as follows:
			
			Dx = tenor expression for “days”, e.g. “D5”, where “x” is any integer > 0
			Mx = tenor expression for “months”, e.g. “M3”, where “x” is any integer > 0
			Wx = tenor expression for “weeks”, e.g. “W13”, where “x” is any integer > 0
			Yx = tenor expression for “years”, e.g. “Y1”, where “x” is any integer > 0

Reserved100Plus	Values 100 and above are reserved for bilaterally agreed upon user defined enumerations.	integer range minInclusive=100	Sequence of character digits without commas or decimals.
Reserved1000Plus	Values 1000 and above are reserved for bilaterally agreed upon user defined enumerations.	integer range minInclusive=1000	Sequence of character digits without commas or decimals.
Reserved4000Plus	Values 4000 and above are reserved for bilaterally agreed upon user	integer range minInclusive=4000	Sequence of character digits without commas or decimals.

[8] Datatype *data* is a union of opaque data and character data in a different encoding. It is recommended that the latter case be distinguished by naming such a field as EncodedXXX, e.g. EncodedSecurityDesc(351).

Data Type	Semantics	Value space (ISO/IEC 11404:2007)	Tagvalue lexical space
	defined enumerations.		
XMLData	XML document	characterstring repertoire=(value of XML encoding declaration)	A field of datatype XMLData must contain a well-formed document, as defined by the W3C XML recommendation. Fields of datatype XMLData must have an associated field of type Length. Fields of datatype XMLData must be immediately preceded by their associated Length field.
Language	External code set ISO 639-1:2002 Codes for the representation of names of languages – Part 1: Alpha-2 code	array element = character index-lowerbound=1 index-upperbound=2	2-character code
LocalMktTime	Time local to a market center.	time time-unit=second	Used where offset to UTC varies throughout the year and the defining market center is identified in a corresponding field. Format is HH:MM:SS where HH = 00-23 hours, MM = 00-59 minutes, SS = 00-59 seconds. In general only the hour token is non-zero.

6.2.3 XML data

An XMLData field may be accompanied by a field of datatype *String*, giving the XML schema used to validate the field of datatype *XMLData*. A schema field, if provided, should contain the URI of a governing XML schema.

Example – XML Definition of a Security

Tag	Field name	FIX Datatype
1184	SecurityXMLLen	Length
1185	SecurityXML	XMLData
1186	SecurityXMLSchema	String

7 Code sets

A code set is a finite, unordered set of valid values that may be applied to a field. In ISO/IEC 11404:2007, a code set is known as *state* datatype. (*state* is distinguished from *enumeration* by being unordered, while an *enumeration* is ordered.) FIX fields may have their value space constrained to a code set. Multiple fields may share a common code set.

7.1 Underlying value type

Code set literals are constrained to an underlying value type. In FIX, code sets may be of *int*, *char*, or *String* type.

7.1.1 Internal code sets

For an internal code set, the valid values are listed in the FIX message standard.

7.1.2 External code sets

External code sets are governed and maintained by other standards organizations. The valid values are not explicitly listed in the FIX message standard.

Annex A
(informative)

Checksum calculation

The checksum of a FIX message is calculated by summing every octet of the message up to and including the `<SOH>` character of the field preceding the CheckSum(10) field. This checksum is then transformed into a modulo 256 number.

For example, if the message length sum of character values has been calculated to be 274 then the modulo 256 value is 18 (256 + 18 = 274). This value would be encoded in the CheckSum(10) field as "10=018".

A sample code fragment to generate the checksum field is as follows:

```
char *GenerateCheckSum( char *buf, long bufLen )
{
    static char tmpBuf[ 4 ];
    long idx;
    unsigned int cks;

    for( idx = 0L, cks = 0; idx < bufLen; cks += (unsigned int)buf[ idx++ ] );
    sprintf( tmpBuf, "%03d", (unsigned int)( cks % 256 ) );
    return( tmpBuf );

}
```

Bibliography

[1] Financial Information eXchange – FIX Session Layer Technical Specification

[2] FIX 4.2 Specification with 20010501 Errata https://www.fixtrading.org/standards/fix-4-2/

[3] FIX 4.4 Specification with 20030618 Errata https://www.fixtrading.org/standards/fix-4-4/

[4] FIX 5.0 Specification Service Pack 2 with 20131209 Errata https://www.fixtrading.org/standards/fix-5-0-sp-2/

[5] FpML 5.11 Recommendation https://www.fpml.org/spec/fpml-5-11-8-rec-1/

[6] FIX Orchestra Technical Specification Draft Standard v1.0 https://www.fixtrading.org/packages/fix-orchestra-technical-specification-draft-standard-v1-0/

Unsupported FIX application versions:

[7] FIX 4.3 Specifications https://www.fixtrading.org/standards/unsupported/fix-4-3/

[8] FIX 5.0 Specifications https://www.fixtrading.org/standards/unsupported/fix-5-0/

[9] FIX 5.0 SP1 Specifications https://www.fixtrading.org/standards/unsupported/fix-5-0-sp1/