FIX PROTOCOL
INDUSTRY-DRIVEN MESSAGING STANDARD℠

Encoding FIX Using ASN.1
Draft Standard

DRAFT STANDARD REVISION 0.3: MARCH 14, 2014

THIS DOCUMENT IS A DRAFT STANDARD FOR A PROPOSED FIX TECHNICAL STANDARD. A DRAFT STANDARD HAS BEEN APPROVED BY THE GLOBAL TECHNICAL COMMITTEE AS THE FINAL STEP IN CREATING A NEW FIX TECHNICAL STANDARD. POTENTIAL ADOPTERS ARE STRONGLY ENCOURAGED TO BEGIN WORKING WITH THE DRAFT STANDARD AND TO PROVIDE FEEDBACK TO THE GLOBAL TECHNICAL COMMITTEE AND THE WORKING GROUP THAT SUBMITTED THE PROPOSAL. THE FEEDBACK TO THE DRAFT STANDARD WILL DETERMINE WHEN TWO INTEROPERABLE IMPLEMENTATIONS HAVE BEEN ESTABLISHED AND THE DRAFT STANDARD CAN BE PROMOTED TO BECOME A NEW FIX TECHNICAL STANDARD.

CONTENTS

Document History

Revision	Date	Author	Revision comments
0.1	December 5, 2013	Alessandro Triglia OSS Nokalva	Revision submitted to GTC
0.2	March 10, 2014	Alessandro Triglia OSS Nokalva	Added Message Encoding Header
0.3	March 14, 2014	Hanno Klein, Deutsche Börse Group	Minor changes for public review

1 Introduction

This technical specification contains provisions for the mapping of the content of the FIX Unified Repository to ASN.1 for the purpose of enabling the exchange of FIX messages between two endpoints in an efficient binary encoding in the presence of high-performance requirements.

ASN.1 is a family of International Standards for the definition and encoding of messages, jointly developed and published by the International Organization for Standardization and the International Telecommunication Union. The mapping specified in this technical specification can be used for any FIX message and generates a set of definitions in the ASN.1 notation. Any standard set of encoding rules such as OER, BER, or PER can then be applied to those ASN.1 definitions to produce efficient binary encodings. A variety of software tools are available that facilitate the development of applications that handle ASN.1 messages. For example, an ASN.1 compiler may take an ASN.1 schema as input and generate source code automatically.

The mapping procedure specified in this technical specification can be applied either to the original FIX Unified Repository or to any other XML document that contains a `<fix>` element with the same syntax as the one in the FIX Unified Repository. For example, a user of FIX could take a subset of the Repository, apply one or more scenarios to some messages, add encoding attributes to a few fields, and finally apply the mapping procedure to the resulting XML document, thus producing an ASN.1 schema.

ASN.1 has several standard encoding rules, each with different characteristics. **BER** is byte-oriented and largely self-describing in that it carries the tag numbers of every field within the encoded message, and adds a length prefix in front of each field and each group of fields. **PER** is bit-oriented and very compact and tries to include in the encoded message only the information strictly necessary to decode it (for example, the length of a string field is not included in the encoded message if the length is constant and specified in the schema). **OER** is byte-oriented and optimized for speed, though not as compact as PER. **XER** is an XML-based encoding and can be used when human-readability is of the greatest concern and encoding/decoding speed is not an issue. In summary, among the three main binary encoding rules of ASN.1, **OER** is the fastest, **PER** is the most compact (but not as fast as OER), and **BER** may be preferred in certain situations, such as when it is important that an encoded message be understandable (i.e., fully or partially decodable) even by someone who does not have the exact ASN.1 schema available.

This technical specification is intended to be used in an algorithmic fashion. Clause 4 contains general provisions and drives the whole mapping procedure. Clause 5 (mapping of FIX datatypes) and clause 6 (mapping of FIX messages) are invoked by subclause 4.1.5. Clause 7 (mapping of a FIX component) is invoked by subclause 6.2.2 and by itself (recursively).

2 References

- ITU-T Recommendation X.680 (2008) | ISO/IEC 8824-1:2008, Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.
- ITU-T Recommendation X.690 (2008) | ISO/IEC 8825-1:2008, Information technology – ASN.1encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER), and Distinguished Encoding Rules (DER).
- ITU-T Recommendation X.691 (2008) | ISO/IEC 8825-2:2008, Information technology – ASN.1encoding rules: Specification of Packed Encoding Rules (PER).
- ITU-T Recommendation X.693 (2008) | ISO/IEC 8825-4:2008, Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)
- NTCIP 1102-2004 v01.15, National Transportation Communications for ITS Protocol, Octet Encoding Rules (OER), Base Protocol, AASHTO, ITE, and NEMA, October 2005

3 Definitions

abstract value: A value whose definition is based only on the type used to carry some semantics, independently of how it is represented in any encoding.

NOTE – Examples of abstract values are the values of the integer type, the boolean type, a character string type, or of a type which is a sequence (or a choice) of an integer and a boolean.

ASN.1 schema: A collection of one or more ASN.1 modules

BER: Basic Encoding Rules, a set of ASN.1 encoding rules where each field or group of fields in the encoded message is preceded by a length and a tag

bit string type: A simple type whose distinguished values are an ordered sequence of zero, one or more bits

boolean type: A simple type with two distinguished values

character string types: Simple types whose values are strings of characters from some defined character set

choice types: Types defined by referencing a list of distinct types; each value of the choice type is derived from the value of one of the component types

constraint: A notation which can be used in association with a type, to define a subtype of that type

DER: Distinguished Encoding Rules, a variant of BER in which each abstract value is encoded in exactly one way

encoding: The bit-pattern resulting from the application of a set of encoding rules to an abstract value

ASN.1 encoding attribute: Each of the attributes of an encoding information element that is present inside an element within the source `<fix>` element and affects the mapping to ASN.1

encoding information element: An element that is present inside an element within the source `<fix>` element and affects the mapping to one or more FIX encoding technologies

(ASN.1) encoding rules: Rules which specify the representation during transfer of the values of ASN.1 types. Encoding rules also enable the values to be recovered from the representation, given knowledge of the type

enumerated types: Simple types whose values are given distinct identifiers as part of the type notation

integer type: A simple type with distinguished values which are the positive and negative whole numbers, including zero (as a single value)

lexical item: A named sequence of characters which is used in forming the ASN.1 notation

module: One or more instances of the use of the ASN.1 notation for type, value, value encapsulated using the ASN.1 module notation

null type: A simple type consisting of a single value, also called null

OER: Octet Encoding Rules, a byte-oriented set of encoding rules optimized for encoding/decoding speed

PER: Packed Encoding Rules, a bit-oriented set of encoding rules optimized for compactness. There are two variants of PER, called PER Aligned and PER Unaligned, respectively.

octet string type: A simple type whose distinguished values are an ordered sequence of zero, one or more octets, each octet being an ordered sequence of eight bits

sequence types: Types defined by referencing a fixed, ordered list of types (some of which may be declared to be optional); each value of the sequence type is an ordered list of values, one from each component type. Where a component type is declared to be optional, a value of the sequence type need not contain a value of that component type

sequence-of types: Types defined by referencing a single component type; each value in the sequence-of type is an ordered list of zero, one or more values of the component type

source `<fix>` element: An XML element named `<fix>` (usually, but not necessarily, located in the FIX Unified Repository) that is used as the source of the mapping to ASN.1

tag: Additional information, separate from the abstract values of the type, which is associated with every ASN.1 type and which can be changed or augmented by a type prefix

tagging: Assigning a new tag to a type, replacing or adding to the existing (possibly the default) tag

transfer syntax: The set of bit strings used to exchange the abstract values in an abstract syntax, usually obtained by application of encoding rules to an abstract syntax

type: A named set of values

type reference name: A name associated uniquely with a type within some context

XER: XML Encoding Rules, a set of encoding rules that encode the values in XML. There are two variants of XER, called Basic XER and Extended XER (E-XER), respectively.

4 General provisions

4.1 Generation of the ASN.1 schema

NOTE – The mapping of the content, or part of the content, of the FIX Unified Repository to ASN.1 generates an ASN.1 schema consisting of one or more ASN.1 modules, each containing one or more ASN.1 type assignments. This technical specification does not restrict the use of white space (including end-of-line characters) and the inclusion of ASN.1 comments between adjacent ASN.1 tokens, other than by requiring conformance to ITU-T Rec. X.680.

4.1.1 The mapping process specified in this technical specification takes as input a `<fix>` element (called the *source* `<fix>` *element*), whose syntax and semantics are identical to those of the `<fix>` element of the FIX Unified Repository v. 5.0 SP2. The mapping process generates one or more ASN.1 modules, each containing one or more ASN.1 type assignments.

4.1.2 The *source* `<fix>` *element* can be one of the following:

a) a `<fix>` element that is part of a FIX Unified Repository instance;
b) a `<fix>` element derived from (a) by excluding part of its content (e.g., keeping only the messages belonging to one or more specified FIX sections and/or FIX categories);
c) a `<fix>` element derived from (a) or (b) by applying one or more specified scenarios to it;
d) any `<fix>` element whose syntax and semantics are identical to those of the `<fix>` element of the FIX Unified Repository v. 5.0 SP2.

4.1.3 The name of each generated ASN.1 module shall consist of a *root* chosen by the implementation of the mapping or by its user concatenated with a *suffix* as specified in subclause 4.1.6. The same *root* shall be used for the names of all the ASN.1 modules generated by the mapping process from the *source* `<fix>` *element*. The concatenation of the *root* and each one of the suffixes specified in subclause 4.1.6 shall be a valid ASN.1 module name according to ITU-T Rec. X.680, 12.5.

NOTE 1 – A valid ASN.1 module name is a string that:

a) begins with an uppercase letter;
b) consists of one or more uppercase letters (A..Z), lowercase letters (a..z), digits (0..9), and hyphens (-);
c) does not contain two or more consecutive hyphens; and
d) does not end with a hyphen.

NOTE 2 – There is no requirement that object identifiers be specified or generated for the generated ASN.1 modules along with the module names.

4.1.4 Every ASN.1 module shall specify `AUTOMATIC TAGS` and shall not specify `EXTENSIBILITY IMPLIED`.

4.1.5 ASN.1 type assignments shall be generated from the *source* `<fix>` *element* as specified in clauses 5 and 6, in this order. Within each of those clauses, ASN.1 type assignments shall be generated strictly in the order specified therein. No other ASN.1 type assignments shall be generated.

NOTE – The ASN.1 name generation rules specified in subclause 4.3 and invoked in multiple subclauses of this technical specification are sensitive to the order in which they are invoked. Therefore, if the mapping process is performed without

following the prescribed order of type assignment generation, some of the names generated within the ASN.1 schema will be different from those generated by a conforming implementation of the mapping.

4.1.6 Each generated ASN.1 type assignments shall be placed in an ASN.1 module determined as follows:

- all type assignments generated from `<datatype>` and `<field>` elements according to clause 5 and its subclauses shall be placed in one ASN.1 module whose name shall have the suffix "`-DATATYPES`";
- all type assignments generated from `<message>` elements according to clause 6 shall be placed in one ASN.1 module whose name shall have the suffix "`-MESSAGES`";
- all type assignments generated from `<component>` elements according to clause 7 shall be placed in one ASN.1 module whose name shall have the suffix "`-COMPONENTS`".

4.1.7 An `IMPORT` clause shall be present in each ASN.1 module containing references to type assignments in other modules, enabling those references as required by ITU-T Rec. X.680, clause 13.

EXAMPLE

The following ASN.1 module could be generated by the mapping process.

```
FIX-5-0-SP2-DATATYPES DEFINITIONS AUTOMATIC TAGS ::=
BEGIN

Int ::= INTEGER
            --generated from a <datatype> element

Length ::= INTEGER (0..MAX)
            --generated from a <datatype> element

AdvSide-enum ::= ENUMERATED {
      buy,
      sell,
      trade,
      cross,
      ...
}
            --generated from a <field> element
            --containing a sequence of <enum> elements

-- ... many other type assignments ...

END
```

4.2 ASN.1 encoding attributes

4.2.1 Within the *source* `<fix>` *element*, *encoding information* is carried by a set of *encoding attributes* within *encoding information* elements. The *encoding information* elements that affect the mapping to ASN.1 are the elements `<encodingInfo>`, `<ASN1>`, and `<XML>`.

4.2.2 The <encodingInfo> element may contain common *encoding attributes* applicable to all standard FIX encodings. The <ASN1> element may contain *encoding attributes* specific to ASN.1 as well as common *encoding attributes*. The <XML> element may contain *encoding attributes* specific to FIXML. All the attributes of an *encoding information* element are optional. An *encoding information* element may be empty.

4.2.3 Each <datatype>, <field>, <fieldRef>, <component>, <componentRef>, or <message> element in the *source <fix> element* may contain zero or more *encoding information* elements, but never contains two *encoding information* elements with the same name. A common *encoding attribute* may be present in both an <encodingInfo> element and in an <ASN1> element within the same <datatype>, <field>, <fieldRef>, <component>, <componentRef>, or <message> element.

4.2.4 The attributes that affect the mapping to ASN.1 are called *ASN.1 encoding attributes* and are specified in table 1.

Table 1 – ASN.1 encoding attributes

Name	Type	In Repository elements[a]	Applicable to FIX datatypes[b]	Description	Default value[c]	ASN.1-specific[d]
minValue	integer	<datatype> <field> <fieldRef>	int Pattern	minimum permitted value of the integer datatype	negative infinity	no
maxValue	integer	<datatype> <field> <fieldRef>	int Pattern	maximum permitted value of the integer datatype	positive infinity	no
isUTF8	boolean	<datatype> <field> <fieldRef>	data	whether the data consists of Unicode characters encoded in UTF-8	false	no
isNumeric	boolean	<datatype>, <field>, <fieldRef>	String	whether all the permitted strings consist of one or more digits ('0' to '9') and can be regarded as nonnegative numbers in character string form	false	no
minLength	integer	<datatype> <field> <fieldRef>	String Pattern	minimum permitted length of the string datatype (minimum number of characters)	zero	no
maxLength	integer	<datatype>, <field>, <fieldRef>	String Pattern	maximum permitted length of the string datatype (maximum number of characters)	positive infinity	no
numBits	integer	<datatype> <field> <fieldRef>	float UTCTimeStamp TZTimeStamp	size in bits	64	no
exponent	integer	<datatype> <field>	float	exponent (either a fixed exponent for a fixed-point decimal number	0	no

		<fieldRef>		or the default exponent for a variable-point decimal number)		
isFixedPoint	boolean	<datatype> <field> <fieldRef>	float	whether the exponent is fixed (true) or variable (false)	false	no
timeUnit	integer	<datatype> <field> <fieldRef>	UTCTimeOnly UTCTimeStamp TZTimeOnly TZTimeStamp	time unit (0=second, 3=millisecond, 6=microsecond, 9=nanosecond,12=picosecond)	9 (nano-second)	no
epoch	string	<datatype> <field> <fieldRef>	UTCDateOnly UTCTimeStamp LocalMktDate TZTimeStamp	reference epoch (either date or date and time) in ISO 8601 format	19700101	no

[a] This column indicates the FIX Repository elements in which the encoding information element (<ASN1> or <encodingInfo>) containing the attribute may appear.

[b] This column indicates the FIX datatypes to which the attribute applies. An attribute is ignored when used with a FIX datatype to which it does not apply.

[c] The default value is used when an attribute is applicable but is absent.

[d] This column indicates whether the *ASN.1 encoding attribute* is an *encoding attribute* specific to ASN.1 (i.e., it may only occur in an <ASN1> element) or an *encoding attribute* common to all the standard FIX encodings (i.e., it may occur either in an <encodingInfo> element or in an <ASN1> element).

EXAMPLE 1

The following <datatype> element describes an integer datatype which, when mapped to ASN.1, will generate an ASN.1 INTEGER type with a permitted value range of 2 to 50.

```
<datatype name="integer-with-bounds-ABC" ...>
      <XML base="xs:integer" .../>
      <ASN1 minValue="2" maxValue="50"/>
</datatype>
```

EXAMPLE 2

The following <datatype> element describes a price datatype which, when mapped to ASN.1, will generate a fixed-point decimal ASN.1 type, consisting of a mantissa with an implicit exponent. The total precision of the mantissa will be 32 bits, and the exponent will be fixed to -4.

```
<datatype name="Price-fixed-ABC" base="float">
      <XML base="xs:decimal" .../>
      <ASN1 numBits="32" isFixedPoint="true" exponent="-4"/>
</datatype>
```

EXAMPLE 3

The following <datatype> element describes a price datatype which, when mapped to ASN.1, will generate a variable-point decimal ASN.1 type, consisting of a mantissa and an exponent. The total precision of the mantissa will be 64 bits, and the exponent will have a default value of -4.

```
<datatype name="Price-floating-ABC" base="float">
      <XML base="xs:decimal" .../>
      <ASN1 numBits="64" isFixedPoint="false" exponent="-4"/>
</datatype>
```

EXAMPLE 4

The following <datatype> element describes a UTC timestamp datatype which, when mapped to ASN.1, will generate a UTC timestamp ASN.1 type, consisting of a single integer. The integer will indicate the number of microseconds from the reference epoch, and its total precision will be 64 bits.

```
<datatype name="Timestamp-ABC" base="UTCTimeStamp">
      <XML base="xs:dateTime" .../>
      <ASN1 numBits="64" timeUnit="6"/>
</datatype>
```

4.2.5 The following subclauses define the terms *textual ASN.1 encoding attributes* and *effective ASN.1 encoding attributes*, which are used in the remaining clauses of this technical specification.

4.2.6 The *textual ASN.1 encoding attributes* of a <datatype>, <field>, or <fieldRef> element are the union of the *ASN.1 encoding attributes* present in its <ASN1> child element (if any) and the *ASN.1 encoding attributes* present in its <encodingInfo> child element (if any), where an attribute present in the former child element overrides any attribute with the same name present in the latter.

4.2.7 The *effective ASN.1 encoding attributes* of a <datatype> element are determined as follows:

a) if the <datatype> element does not have a base attribute, the *effective ASN.1 encoding attributes* are its *textual ASN.1 encoding attributes*, completed by the default values specified in table 1 for any *ASN.1 encoding attributes* that are not present;
b) otherwise, the *effective ASN.1 encoding attributes* are the union of its *textual ASN.1 encoding attributes* and the *effective ASN.1 encoding attributes* of the base <datatype> element, where a *textual ASN.1 encoding attribute* of this <datatype> element overrides any *effective ASN.1 encoding attribute* with the same name of the base <datatype> element.

NOTE – This recursive definition assumes that the *source <fix> element* does not contain any circular references to <datatype> elements via their base attribute.

4.2.8 The *effective ASN.1 encoding attributes* of a `<field>` element are the union of its *textual ASN.1 encoding attributes* and the *effective ASN.1 encoding attributes* of the `<datatype>` element referenced by the `type` attribute of the `<field>` element, where a *textual ASN.1 encoding attribute* of the `<field>` element overrides any *effective ASN.1 encoding attribute* with the same name of the `<datatype>` element.

4.2.9 The *effective ASN.1 encoding attributes* of a `<fieldRef>` element are the union of its *textual ASN.1 encoding attributes* and the *effective ASN.1 encoding attributes* of the `<field>` element referenced by the `name` attribute of the `<fieldRef>` element, where a *textual ASN.1 encoding attribute* of the `<fieldRef>` element overrides any *effective ASN.1 encoding attribute* with the same name of the `<field>` element.

4.3 Generation of ASN.1 names

4.3.1 This subclause specifies rules for the generation of various kinds of ASN.1 names (type names, component identifiers of `SEQUENCE` types, enumerator identifiers of `ENUMERATED` types, and bit position identifiers of `BIT STRING` types) from character strings.

NOTE – These name generation rules are similar to those specified in ITU-T Rec. X.694, 10.3, for the ASN.1 names generated in the mapping from XML Schema to ASN.1.

4.3.2 The following transformations shall be applied, in order, to each character string being mapped to an ASN.1 name, where each transformation except the first is applied to the result of the previous transformation:

1) the characters ' ' (SPACE), '.' (FULL STOP), and '_' (LOW LINE) shall be replaced by a '-' (HYPHEN-MINUS);
2) any character except 'A' to 'Z' (LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z), 'a' to 'z' (LATIN SMALL LETTER A to LATIN SMALL LETTER Z), '0' to '9' (DIGIT ZERO to DIGIT NINE), and '-' (HYPHEN-MINUS) shall be removed;
3) a sequence of two or more HYPHEN-MINUS characters shall be replaced with a single HYPHEN-MINUS;
4) any HYPHEN-MINUS characters occurring at the beginning or at the end of the character string shall be removed;
5) if a character string that is to be used as a type name starts with a lower-case letter, the first letter shall be converted to upper case;
6) if a character string that is to be used as a type name starts with a digit (DIGIT ZERO to DIGIT NINE), it shall be prefixed with an 'X' (LATIN CAPITAL LETTER X);
7) if a character string that is to be used as an identifier starts with an upper-case letter, the first letter shall be converted to lower case;
8) if a character string that is to be used as an identifier starts with a digit (DIGIT ZERO to DIGIT NINE), it shall be prefixed with an 'x' (LATIN SMALL LETTER X);
9) if a character string that is to be used as a type name is empty, it shall be replaced by 'X' (LATIN CAPITAL LETTER X);
10) if a character string that is to be used as an identifier is empty, it shall be replaced by 'x' (LATIN SMALL LETTER X).

4.3.3 Depending on the kind of name being generated, exactly one of the subclauses 4.3.3.1 to 4.3.3.3 applies.

4.3.3.1 If the name being generated is the type name on the left side of an ASN.1 type assignment and the character string produced by subclause 4.3.2 either:

- is one of the reserved words specified in ITU-T Rec. X.680, 11.27; or
- matches the name of one of the supporting ASN.1 types specified in subclause 5.4; or
- is identical to the type name on the left side of another ASN.1 type assignment previously generated within the same or any other ASN.1 module;

then a suffix shall be appended to the character string produced by subclause 4.3.2. The suffix shall consist of a HYPHEN-MINUS followed by the least positive integer (with no leading zeros) such that the resulting name differs from the type name on the left side of any other ASN.1 type assignment previously generated within the same or any other ASN.1 module.

4.3.3.2 If the name being generated is the identifier of a component of a `SEQUENCE` type and the character string produced by subclause 4.3.2 is identical to the identifier of a previously generated component of the same `SEQUENCE` type, then a suffix shall be appended to the character string produced by subclause 4.3.2. The suffix shall consist of a HYPHEN-MINUS followed by the least positive integer (with no leading zeros) such that the resulting identifier differs from the identifier of any previously generated component of that `SEQUENCE` type.

4.3.3.3 If the name being generated is the identifier of an item of an `ENUMERATED` or `BIT STRING` type and the character string produced by subclause 4.3.2 is identical to the identifier of a previously generated item of the same `ENUMERATED` or `BIT STRING` type, then a suffix shall be appended to the character string produced by subclause 4.3.2. The suffix shall consist of a HYPHEN-MINUS followed by the least positive integer (with no leading zeros) such that the resulting identifier differs from the identifier of any previously generated item of that `ENUMERATED` or `BIT STRING` type.

5 Mapping of FIX datatypes

5.1 Datatypes explicitly defined via <datatype> elements

5.1.6 For each <datatype> element in the <datatypes> element of the *source <fix> element*, in order, an ASN.1 type assignment shall be generated as specified in subclauses 5.1.6.1 to 5.1.6.2.

5.1.6.1 The type name on the left side of the type assignment shall be generated from the name indicated in the <datatype> element in accordance with subclause 4.3.

5.1.6.2 The type expression on the right side of the type assignment shall be the *target ASN.1 type expression* determined from the <datatype> element as specified in subclause 5.3.1.

EXAMPLE 1

The following <datatype> element:

```
<datatype name="int" ...>
      <XML base="xs:integer" .../>
</datatype>
```

will be mapped to ASN.1 as follows:

Source FIX datatype	int
XSD datatype	integer
ASN.1 type name	Int
ASN.1 type expression	INTEGER
Generated ASN.1 type assignment	Int ::= INTEGER

EXAMPLE 2

The following <datatype> element:

```
<datatype name="Length" baseType="int" ...>
      <XML base="xs:nonNegativeInteger" .../>
</datatype>
```

will be mapped to ASN.1 as follows:

Source FIX datatype	Length (derived from int)
XSD datatype	nonNegativeInteger
ASN.1 type name	Length

ASN.1 type expression	`INTEGER (0..MAX)`
Generated ASN.1 type assignment	`Length ::= INTEGER (0..MAX)`

5.2 Datatypes implicitly defined via <field> elements

5.2.1 The type of a FIX field in the *source* <fix> *element* is the FIX datatype determined from a <field> element, its attributes, and its child elements, as specified in table 2.

Table 2 – Determination of the type of a FIX field from a <field> element

Case	Multiple-valued type[a]	<enum> child elements[b]	enum DataType attribute[c]	Different encoding attributes[d]	union DataType attribute[e]	Type of the FIX field
1	no	no	no	no	no	the FIX datatype indicated in the type attribute
2	no	no	no	**yes**	no	the FIX datatype indicated in the type attribute, modified by the *effective ASN.1 encoding attributes* of the <field> element
3	no	**yes**	no	no	no	the FIX datatype indicated in the type attribute, restricted by the enumeration specified in the <enum> child elements of this <field> element
4	no	no	**yes**	no	no	the FIX datatype indicated in the type attribute, restricted by the enumeration specified in the <enum> child elements of the <field> element referenced by the enumDataType attribute
5	**yes**	no	no	no	no	the FIX datatype (either MultipleCharValue or MultipleStringValue) with no restrictions on the items of the space-separated list
6	**yes**	**yes**	no	no	no	the FIX datatype (either MultipleCharValue or MultipleStringValue) where each item of the space-separated list is required to belong to the enumeration specified in the <enum> child elements of this <field> element
7	**yes**	no	**yes**	no	no	the FIX datatype (either MultipleCharValue or MultipleStringValue) where each item of the space-separated list is required to belong to the enumeration specified in the <enum> child elements of the <field> element referenced by the enumDataType attribute
8	no	no	no	no	**yes**	a union between the type determined in case 1 above and the FIX datatype indicated in the unionDataType attribute

9	no	no	no	**yes**	**yes**	a union between the type determined in case 2 above and the FIX datatype indicated in the `unionDataType` attribute
10	no	**yes**	no	no	**yes**	a union between the type determined in case 3 above and the FIX datatype indicated in the `unionDataType` attribute
11	no	no	**yes**	no	**yes**	a union between the type determined in case 4 above and the FIX datatype indicated in the `unionDataType` attribute

NOTE – Other combinations of column values do not occur

[a] This column indicates whether the `type` attribute of the `<field>` element is a multiple-valued datatype (either `MultipleCharValue` or `MultipleStringValue`)

[b] This column indicates whether the `<field>` element has one or more `<enum>` child elements

[c] This column indicates whether the `<field>` element has an `enumDataType` attribute

[d] This column indicates whether the *effective ASN.1 encoding attributes* of the `<field>` element differ from the *effective ASN.1 encoding attributes* of the `<datatype>` element referenced by the `type` attribute

[e] This column indicates whether the `<field>` element has a `unionDataType` attribute

5.2.2 For each `<field>` element in the `<fields>` element of the *source* `<fix>` *element*, in order, that has one or more `<enum>` child elements and either:

a) it has a `type` other than `MultipleCharValue` or `MultipleStringValue` (cases 3 and 10 of table 2); or
b) it is referenced by the `enumDataType` attribute of a `<field>` element having a `type` other than `MultipleCharValue` or `MultipleStringValue` (cases 4 and 11 of table 2),

an ASN.1 type assignment shall be generated as specified in subclauses 5.2.2.1 to 5.2.2.3.

5.2.2.1 The type name on the left side of the type assignment shall be generated from the name of the FIX field with the "`-enum`" suffix appended, in accordance with subclause 4.3.

5.2.2.2 The type expression on the right side of the type assignment shall be an `ENUMERATED` type expression containing one enumerator identifier for each `<enum>` child element of the `<field>` element, in order. Each identifier shall be generated from the value of the `symbolicName` attribute of the corresponding `<enum>` element in accordance with subclause 4.3. If the `type` of the `<field>` element is `int` or a datatype derived from `int`, each identifier shall be followed by the value of the `value` attribute of the corresponding `<enum>` element within round brackets.

NOTE – For a field having the FIX datatype `Currency`, if a list of `<enum>` child elements specifing a limited set of currency codes is added to the `<field>` element, the type of the field will be mapped to an `ENUMERATED` type. The default mapping of the `Currency` datatype (a `IA5String` type with a fixed size) will not be used for this particular field. The

same considerations apply to other FIX datatypes that rely on an externally defined codelist, such as `Exchange`, `Country`, or `Language`.

5.2.2.3 An extension marker (`...`) shall be added after the last enumerator of the `ENUMERATED` type.

EXAMPLE 1

The following `<field>` element:

```
<field ... name="AdvSide" type="char" ... abbrName="AdvSide"
notReqXML="0">
      <enum value="B" symbolicName="Buy" ... />
      <enum value="S" symbolicName="Sell" .../>
      <enum value="T" symbolicName="Trade" .../>
      <enum value="X" symbolicName="Cross" .../>
</field>
```

will generate the following ASN.1 type assignment:

```
AdvSide-enum ::= ENUMERATED {
      buy,
      sell,
      trade,
      cross,
      ...
}
```

EXAMPLE 2

The following `<field>` element:

```
<field ... name="AllocStatus" type="int" ... abbrName="Stat"
notReqXML="0">
      <enum value="0" symbolicName="Accepted" .../>
      <enum value="1" symbolicName="BlockLevelReject" .../>
      <enum value="2" symbolicName="AccountLevelReject" .../>
      <enum value="3" symbolicName="Received" .../>
      <enum value="4" symbolicName="Incomplete" .../>
      <enum value="5" symbolicName="RejectedByIntermediary" .../>
      <enum value="6" symbolicName="AllocationPending" .../>
      <enum value="7" symbolicName="Reversed" .../>
      <enum value="8" symbolicName="CancelledByIntermediary" .../>
      <enum value="9" symbolicName="Claimed" .../>
      <enum value="10" symbolicName="Refused" .../>
      <enum value="11" symbolicName="PendingGiveUpApproval" .../>
      <enum value="12" symbolicName="Cancelled" .../>
      <enum value="13" symbolicName="PendingTakeUpApproval" .../>
      <enum value="14" symbolicName="ReversalPending" .../>
```

```
</field>
```

will generate the following ASN.1 type assignment:

```
AllocStatus-enum ::= ENUMERATED {
      accepted                      (0),
      blockLevelReject              (1),
      accountLevelReject            (2),
      received                      (3),
      incomplete                    (4),
      rejectedByIntermediary        (5),
      allocationPending             (6),
      reversed                      (7),
      cancelledByIntermediary       (8),
      claimed                       (9),
      refused                       (10),
      pendingGiveUpApproval         (11),
      cancelled                     (12),
      pendingTakeUpApproval         (13),
      reversalPending               (14),
      ...
}
```

5.2.3 For each <field> element in the <fields> element of the *source* <fix> *element*, in order, that has one or more <enum> child elements and either:

a) it has a type of MultipleCharValue or MultipleStringValue (case 6 of table 2); or
a) it is referenced by the enumDataType attribute of a <field> element having a type of MultipleCharValue or MultipleStringValue (case 7 of table 2),

an ASN.1 type assignment shall be generated as specified in subclauses 5.2.3.1 to 5.2.3.2.

5.2.3.1 The type name on the left side of the type assignment shall be generated from the name of the FIX field with the "-bitmap" suffix appended, in accordance with subclause 4.3.

5.2.3.2 The type expression on the right side of the type assignment shall be a BIT STRING type expression with a named bit list containing one bit position identifier for each <enum> child element of the <field> element. Each identifier shall be generated from the value of the symbolicName attribute of the <enum> element in accordance with subclause 4.3. The identifiers shall occur in the same order as the <enum> elements, and the bit positions shall be consecutively numbered starting from zero.

EXAMPLE

The following <field> element:

```
<field ... id="276" name="QuoteCondition" type="MultipleStringValue"
...>
```

```
      <enum value="A" symbolicName="Open" .../>
      <enum value="B" symbolicName="Closed" ... />
      <enum value="C" symbolicName="ExchangeBest" ... />
      <enum value="D" symbolicName="ConsolidatedBest" .../>
</field>
```

will generate the following ASN.1 type assignment:

```
QuoteCondition-bitmap ::= BIT STRING {
      open              (0),
      closed            (1),
      exchangeBest      (2),
      consolidateBest   (3)
} (SIZE (4))
```

5.2.4 For each <field> element in the <fields> element of the *source* <fix> *element*, in order, that has a unionDataType attribute (cases 8, 9, 10, and 11 of table 2) an ASN.1 type assignment shall be generated as specified in subclauses 5.2.4.1 to 5.2.4.4.

5.2.4.1 The type name on the left side of the type assignment shall be generated from the name of the FIX field with the "-union" suffix appended, in accordance with subclause 4.3.

5.2.4.2 The type expression on the right side of the type assignment shall be a CHOICE type expression containing two alternatives.

5.2.4.3 The identifier of the first alternative of the CHOICE type shall be basic, and its type expression shall be determined as specified in table 3.

Table 3 – Determination of the ASN.1 type of the first alternative of a CHOICE type

Case of table 2	Type expression of the first alternative	Reference
8	the type name on the left side of the ASN.1 type assignment generated from the FIX datatype indicated in the type attribute of the <field> element	subclause 5.1.6
9	the *target ASN.1 type expression* determined from the <field> element	subclause 5.3.2
10	the type name on the left side of the ENUMERATED type assignment generated from the <enum> child elements of this <field> element	subclause 5.2.2
11	the type name on the left side of the ENUMERATED type assignment generated from the <enum> child elements of the <field> element whose id is indicated in the enumDataType attribute of this <field> element	subclause 5.2.2

NOTE – Since the ASN.1 type of the first alternative of the CHOICE type is determined at this stage of the mapping from the <field> element, any *textual ASN.1 encoding attribute* of a <fieldRef> element referencing this <field> element will have no effect on the first alternative of the CHOICE type.

5.2.4.4 The identifier of the second alternative of the CHOICE type shall be ext, and its type expression shall be the type name on the left side of the ASN.1 type assignment generated from the FIX datatype indicated in the unionDataType attribute as specified in subclause 5.1.6.

EXAMPLE 1

The following <field> element:

```
<field ... id="1778" name="EntitlementAttribType" type="int" ...
      unionDataType="Reserved4000Plus"/>
```

will generate the following ASN.1 type assignment:

```
EntitlementAttribType-union ::= CHOICE {
      basic     Int,
      ext       Reserved4000Plus
}
```

EXAMPLE 2

The following <field> element:

```
<field ... id="723" name="PosMaintResult" type="int" ...
            abbrName="Rslt" notReqXML="0"
unionDataType="Reserved100Plus">
      <enum value="0" symbolicName="SuccessfulCompletion" ... />
      <enum value="1" symbolicName="Rejected" .../>
      <enum value="99" symbolicName="Other" ... />
</field>
```

will generate the following ASN.1 type assignments:

```
PosMaintResult-enum ::= ENUMERATED {
      successfulCompletion   (0),
      rejected               (1),
      other                  (99),
      ...
}

PosMaintResult-union ::= CHOICE {
      basic     PosMaintResult-enum,
      ext       Reserved100Plus
}
```

EXAMPLE 3

The following <field> element:

```
<field ... id="368" name="QuoteEntryRejectReason" type="int" ...
      enumDatatype="300" unionDataType="Reserved100Plus"/>
```

will generate the following ASN.1 type assignment:

```
QuoteEntryRejectReason-union ::= CHOICE {
      basic      QuoteRejectReason-enum,
      ext        Reserved100Plus
}
```

5.3 Datatype mapping to ASN.1 types

5.3.1 Determination of the target ASN.1 type expression corresponding to a <datatype> element

The *target ASN.1 type expression* corresponding to a <datatype> element shall be determined according the following procedure:

1) Call *source FIX datatype* the FIX datatype described by the <datatype> element.
2) Scan table 4 from the top down, and select the first row that satisfies both of the following conditions:
   a) the first cell of the row references either the *source FIX datatype* or one of its ancestor FIX datatypes; and
   b) the second cell of the row is either empty, or it references either the XSD datatype described by the <XML> child element (if any) of the <datatype> element or one of its ancestor XSD datatypes.
3) Get the ASN.1 type expression from the third cell of the selected row.
4) If the ASN.1 type expression contains one or more names of ASN.1 encoding attributes, replace each of them with the value of the *effective ASN.1 encoding attribute* of the <datatype> element with that name.
5) If the ASN.1 type expression references one of the *supporting ASN.1 types* specified in a subclause of subclause 5.4, the subclause that specifies the supporting ASN.1 type shall be invoked.

Table 4 – Datatype mapping rules

FIX datatype	XSD datatype	ASN.1 type expression
NumInGroup		INTEGER (0..MAX)
DayOfMonth		INTEGER (1..31)
Reserved100Plus		INTEGER (100..MAX)
Reserved1000Plus		INTEGER (1000..MAX)
Reserved4000Plus		INTEGER (4000..MAX)
int	nonNegativeInteger	INTEGER (0..MAX)

int	positiveInteger	INTEGER (1..MAX)
int		when minValue is absent: INTEGER when minValue is present and maxValue is absent: INTEGER (minValue..MAX) when minValue and maxValue are both present: INTEGER (minValue..maxValue)
float		when isFixedPoint is true and minValue is either absent or less than zero: Decimal-fixedexponent-numBits (see subclause 5.4.1) when isFixedPoint is true and minValue is positive or zero: Decimal-fixedexponent-nonneg-numBits (see subclause 5.4.2) when isFixedPoint is false and minValue is either absent or less than zero: Decimal-varexponent-numBits (see subclause 5.4.3) when isFixedPoint is false and minValue is positive or zero: Decimal-varexponent-nonneg-numBits (see subclause 5.4.4) (in all of the above names there is no hyphen before "exponent")
UTCDateOnly		UTCDateOnly-epoch (see subclause 5.4.5)
UTCTimeOnly		UTCTimeOnly-timeUnit (see subclause 5.4.6)
UTCTimestamp		UTCTimestamp-timeUnit-epoch-numBits (see subclause 5.4.7)
LocalMktDate		LocalMktDate-epoch (see subclause 5.4.8)
TZTimeOnly		TZTimeOnly-timeUnit (see subclause 5.4.9)
TZTimestamp		TZTimestamp-timeUnit-epoch-numBits (see subclause 5.4.10)
data		when isUTF8 is false: BinaryString (see subclause 5.4.11) when isUTF8 is true: UTF8String
XMLData		XMLString (see subclause 5.4.12)

char		IA5String (SIZE (1))
Country		IA5String (SIZE (2))
Currency		IA5String (SIZE (3))
String		when isNumeric is false: when both minLength and maxLength are absent: IA5String when minLength is present but maxLength is absent: IA5String (SIZE (minLength..MAX)) when minLength is absent but maxLength is present: IA5String (SIZE (0..maxLength)) when minLength and maxLength are present and different: IA5String (SIZE (minLength..maxLength)) when minLength and maxLength are present and equal: IA5String (SIZE (minLength)) when isNumeric is true: when minValue is absent: INTEGER when minValue is present and maxValue is absent: INTEGER (minValue..MAX) when minValue and maxValue are both present: INTEGER (minValue..maxValue)
Tenor		Duration (see subclause 5.4.13)
MonthYear		YearAndMonth (see subclause 5.4.14)
Pattern	integer	same as for the int datatype above
Pattern		same as for the String datatype above

NOTE In this technical specification, some time-related datatypes are mapped to a single INTEGER type containing the number of time units from a reference epoch, and others are mapped to a SEQUENCE type containing such an INTEGER type along with another INTEGER type representing the time offset from UTC expressed in minutes. In both cases the default time unit is the **nanosecond**, and the default reference epoch is **midnight, January 1, 1970 UTC** (19700101T000000

in ISO 8601 notation). Both the time unit and the reference epoch can be changed by using the *ASN.1 encoding attributes* `timeUnit` and `epoch` in the *source* `<fix>` *element*.

5.3.2 Determination of the target ASN.1 type expression corresponding to a <field> element

The *target ASN.1 type expression* corresponding to a `<field>` element shall be determined according the following procedure:

1) Call *source FIX datatype* the FIX datatype described by the `<datatype>` element referenced by the `type` attribute of the `<field>` element.
2) Scan table 4 from the top down, and select the first row that satisfies both of the following conditions:
   a) the first cell of the row references either the *source FIX datatype* or one of its ancestor FIX datatypes; and
   b) the second cell of the row is either empty, or it references either the XSD datatype described by the `<XML>` child element (if any) of the `<datatype>` element or one of its ancestor XSD datatypes.
3) Get the ASN.1 type expression from the third cell of the selected row.
4) If the ASN.1 type expression contains one or more names of ASN.1 encoding attributes, replace each of them with the value of the *effective ASN.1 encoding attribute* of the `<field>` element with that name.
5) If the ASN.1 type expression references one of the *supporting ASN.1 types* specified in a subclause of subclause 5.4, the subclause that specifies the supporting ASN.1 type shall be invoked.

5.3.3 Determination of the target ASN.1 type expression corresponding to a <fieldref> element

The *target ASN.1 type expression* corresponding to a `<fieldref>` element shall be determined according the following procedure:

1) Call *source FIX datatype* the FIX datatype described by the `<datatype>` element referenced by the `type` attribute of the `<field>` element that is referenced by the `name` attribute of the `<fieldRef>` element.
2) Scan table 4 from the top down, and select the first row that satisfies both of the following conditions:
   a) the first cell of the row references either the *source FIX datatype* or one of its ancestor FIX datatypes; and
   b) the second cell of the row is either empty, or it references either the XSD datatype described by the `<XML>` child element (if any) of the `<datatype>` element or one of its ancestor XSD datatypes.
3) Get the ASN.1 type expression from the third cell of the selected row.
4) If the ASN.1 type expression contains one or more names of ASN.1 encoding attributes, replace each of them with the value of the *effective ASN.1 encoding attribute* of the `<fieldref>` element with that name.
5) If the ASN.1 type expression references one of the *supporting ASN.1 types* specified in a subclause of subclause 5.4, the subclause that specifies the supporting ASN.1 type shall be invoked.

5.4 Supporting ASN.1 types

NOTE – Only the supporting ASN.1 types that are actually needed will be generated, and each of them will be generated at most once.

5.4.1 The `Decimal-fixedN-S` types

5.4.1.1 A `Decimal-fixedN-S` type holds a decimal value with a fixed exponent.

5.4.1.2 "`Decimal-fixedN-S`" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *S* stands for a number indicating the size of the mantissa (either 32 or 64 bits), and *N* stands for the exponent (fixed).

NOTE – There is no hyphen ('-') between "`fixed`" and *N*, because *N* can be a negative number and two consecutive hyphens are not allowed in a name.

5.4.1.3 On the first invocation of this subclause 5.4.1 for given *N* and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *N* replaced with the actual exponent specified in the subclause that invokes this subclause 5.4.1:

- when *S* = 32

```
Decimal-fixedN-32 ::= INTEGER (-2147483648..2147483647)
      --a mantissa with an implicit exponent of N
```

- when *S* = 64

```
Decimal-fixedN-64 ::= INTEGER (-9223372036854775808..9223372036854775807)
      --a mantissa with an implicit exponent of N
```

NOTE – In BER, PER, and OER, these ASN.1 types have an efficient binary encoding, in which the two fields of the `SEQUENCE` type (`mantissa` and `exponent`) are encoded as two separate binary integers. The `exponent` field is not included in the encoding if it has the default value.

EXAMPLE

When *S* is 32 (indicating a size of 32 bits) and *N* is -4 (indicating a fixed exponent of -4), the following ASN.1 type assignment will be generated:

```
Decimal-fixed-4-32 ::= INTEGER (-2147483648..2147483647)
            --a mantissa with an implicit exponent of -4
```

Notice the minus sign (-) before the character '4' in the name of the type, `Decimal-fixed-4-32`.

5.4.2 The `Decimal-fixedN-nonneg-S` types

5.4.2.1 A `Decimal-fixedN-nonneg-S` type holds a non-negative decimal value with a fixed exponent.

5.4.2.2 "`Decimal-fixedN-nonneg-S`" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *S* stands for a number indicating the size of the mantissa (either 32 or 64 bits), and *N* stands for the exponent (fixed).

NOTE – There is no hyphen ('-') between "fixed" and *N*, because *N* can be a negative number and two consecutive hyphens are not allowed in a name.

5.4.2.3 On the first invocation of this subclause 5.4.2 for given *N* and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *N* replaced with the actual exponent specified in the subclause that invokes this subclause 5.4.2:

- when *S* = 32

```
Decimal-fixedN-nonneg-32 ::= INTEGER (0..4294967295)
      --a mantissa with an implicit exponent of N
```

- when *S* = 64

```
Decimal-fixedN-nonneg-64 ::= INTEGER (0..18446744073709551615)
      --a mantissa with an implicit exponent of N
```

NOTE – In BER, PER, and OER, these ASN.1 types have an efficient binary encoding, in which the two fields of the SEQUENCE type (mantissa and exponent) are encoded as two separate binary integers. The exponent field is not included in the encoding if it has the default value.

EXAMPLE

When *S* is 32 (indicating a size of 32 bits) and *N* is -4 (indicating a fixed exponent of -4), the following ASN.1 type assignment will be generated:

```
Decimal-fixed-4-nonneg-32 ::= INTEGER (0..4294967295)
            --a mantissa with an implicit exponent of -4
```

Notice the minus sign (-) before the character '4' in the name of the type, Decimal-fixed-4-nonneg-32.

5.4.3 The Decimal-var*N*-*S* types

5.4.3.1 A Decimal-var*N*-*S* type holds a decimal value with a variable exponent. A default exponent is specified in the ASN.1 schema and can be overridden in an instance of a message over the wire.

5.4.3.2 "Decimal-var*N*-*S*" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *S* stands for a number indicating the size of the mantissa (either 32 or 64 bits), and *N* stands for the default exponent.

NOTE – There is no hyphen (-) between "var" and *N*, because *N* can be a negative number and two consecutive hyphens are not allowed in a name.

5.4.3.3 On the first invocation of this subclause 5.4.3 for given *N* and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *N* replaced with the actual exponent specified in the subclause that invokes this subclause 5.4.3:

- when *S* = 32:

```
Decimal-varN-32 ::= SEQUENCE {
     mantissa   INTEGER (-2147483648..2147483647),
     exponent   INTEGER (-128..127) DEFAULT N
}
```

- when *S* = 64:

```
Decimal-varN-64 ::= SEQUENCE {
     mantissa    INTEGER (-9223372036854775808..9223372036854775807),
     exponent    INTEGER (-128..127) DEFAULT N
}
```

NOTE – In BER, PER, and OER, these ASN.1 types have an efficient binary encoding, in which the two fields of the `SEQUENCE` type (`mantissa` and `exponent`) are encoded as two separate binary integers. The `exponent` field is not included in the encoding if it has the default value.

EXAMPLE

When *S* is 32 (indicating a size of 32 bits) and *N* is -4 (indicating a default exponent of -4), the following ASN.1 type assignment will be generated:

```
Decimal-var-4-32 ::= SEQUENCE {
     mantissa   INTEGER (-2147483648..2147483647),
     exponent   INTEGER (-128..127) DEFAULT -4
}
```

Notice the minus sign (-) before the character '4' in the name of the type, `Decimal-var-4-32`. These two characters represent the negative number -4. If the default exponent were the positive number 4, the name of the type would be `Decimal-var4-32`.

5.4.4 The `Decimal-varN-nonneg-S` types

5.4.4.1 A `Decimal-varN-nonneg-S` type holds a non-negative decimal value with a variable exponent. A default exponent is specified in the ASN.1 schema and can be overridden in an instance of a message over the wire.

5.4.4.2 "`Decimal-varN-nonneg-S` " is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *S* stands for a number indicating the size of the mantissa (either 32 or 64 bits), and *N* stands for the default exponent.

NOTE – There is no hyphen (`-`) between "`var`" and *N*, because *N* can be a negative number and two consecutive hyphens are not allowed in a name.

5.4.4.3 On the first invocation of this subclause 5.4.4 for given *N* and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *N* replaced with the actual exponent specified in the subclause that invokes this subclause 5.4.4:

- when *S* = 32

```
Decimal-varN-nonneg-32 ::= SEQUENCE {
     mantissa   INTEGER (0..4294967295),
     exponent   INTEGER (-128..127) DEFAULT N
}
```

- when *S* = 64

```
Decimal-varN-nonneg-64 ::= SEQUENCE {
     mantissa    INTEGER (0..18446744073709551615),
     exponent    INTEGER (-128..127) DEFAULT N
}
```

NOTE – In BER, PER, and OER, these ASN.1 types have an efficient binary encoding, in which the two fields of the `SEQUENCE` type (`mantissa` and `exponent`) are encoded as two separate binary integers. The `exponent` field is not included in the encoding when it has the default value.

EXAMPLE

When *S* is 32 (indicating a size of 32 bits) and *N* is -4 (indicating a default exponent of -4), the following ASN.1 type assignment will be generated:

```
Decimal-var-4-nonneg-32 ::= SEQUENCE {
     mantissa   INTEGER (0..4294967295),
     exponent   INTEGER (-128..127) DEFAULT -4
}
```

Notice the minus sign (-) before the character '4' in the name of the type, `Decimal-var-4-nonneg-32`. These two characters represent the negative number -4. If the default exponent were the positive number 4, the name of the type would be `Decimal-var4-nonneg-32`.

5.4.5 The `UTCDateOnly-E` types

5.4.5.1 The `UTCDateOnly-E` type holds a UTC date represented as a single integer that is the number of days from a reference epoch.

5.4.5.2 "`UTCDateOnly-E`" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *E* stands for a string indicating the reference epoch in ISO 8601 format.

5.4.5.3 On the first invocation of this subclause 5.4.5 for a given *E*, the following ASN.1 type assignment shall be generated, with the letter *E* replaced with the actual epoch specified in the subclause that invokes this subclause 5.4.5:

```
UTCDateOnly-E ::= INTEGER (0..65535)  --days from epoch E
```

5.4.6 The UTCTimeOnly-*U* types

5.4.6.1 A UTCTimeOnly-*U* type holds a UTC time-of-day represented as a single integer that is the number of time units since midnight. The time unit can be a second, a millisecond, a microsecond, a nanosecond, or a picosecond.

5.4.6.2 "UTCTimeOnly-*U*" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *U* stands for a number denoting the time unit (0=second, 3=millisecond, 6=microsecond, 9=nanosecond, 12=picosecond).

5.4.6.3 On the first invocation of this subclause 5.4.6 for a given *U*, exactly one of the following ASN.1 type assignments shall be generated:

- when *U* = 0

```
UTCTimeOnly-0 ::= INTEGER (0..87839)
      --seconds since midnight
```

- when *U* = 3

```
UTCTimeOnly-3 ::= INTEGER (0..87839999)
      --milliseconds since midnight
```

- when *U* = 6

```
UTCTimeOnly-6 ::= INTEGER (0..87839999999)
      --microseconds since midnight
```

- when *U* = 9

```
UTCTimeOnly-9 ::= INTEGER (0..87839999999999)
      --nanoseconds since midnight
```

- when *U* = 12

```
UTCTimeOnly-12 ::= INTEGER (0..87839999999999999)
      --picoseconds since midnight
```

NOTE – The value ranges in the above type expressions allow for a leap second to be present at the end of a day (24 x 60 x 61 = 87840).

5.4.7 The UTCTimeStamp-*U*-*E*-*S* types

5.4.7.1 A UTCTimeStamp-*U*-*E*-*S* type holds a UTC date-and-time represented as a single integer that is the number of time units from a reference epoch. The time unit can be a second, a millisecond, a microsecond, a nanosecond, or a picosecond.

5.4.7.2 "UTCTimeStamp-*U*-*E*-*S*" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *U* stands for a number denoting the time unit (0=second, 3=millisecond, 6=microsecond, 9=nanosecond,

12=picosecond), *E* stands for a string indicating the reference epoch in ISO 8601 format, and *S* stands for a number indicating the size of the integer (either 32 or 64 bits).

5.4.7.3 On the first invocation of this subclause 5.4.7 for given *U*, *E*, and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *E* replaced with the actual epoch specified in the subclause that invokes this subclause 5.4.7:

- when *U* = 0 and *S* = 32

```
UTCTimeStamp-0-E-32 ::= INTEGER (0..4294967295)
      --seconds from epoch E
```

- when *U* = 0 and *S* = 64

```
UTCTimeStamp-0-E-64 ::= INTEGER (0..18446744073709551615)
      --seconds from epoch E
```

- when *U* = 3 and *S* = 32

```
UTCTimeStamp-3-E-32 ::= INTEGER (0..4294967295)
      --milliseconds from epoch E
```

- when *U* = 3 and *S* = 64

```
UTCTimeStamp-3-E-64 ::= INTEGER (0..18446744073709551615)
      --milliseconds from epoch E
```

- when *U* = 6 and *S* = 32

```
UTCTimeStamp-6-E-32 ::= INTEGER (0..4294967295)
      --microseconds from epoch E
```

- when *U* = 6 and *S* = 64

```
UTCTimeStamp-6-E-64 ::= INTEGER (0..18446744073709551615)
      --microseconds from epoch E
```

- when *U* = 9 and *S* = 32

```
UTCTimeStamp-9-E-32 ::= INTEGER (0..4294967295)
      --nanoseconds from epoch E
```

- when *U* = 9 and *S* = 64

```
UTCTimeStamp-9-E-64 ::= INTEGER (0..18446744073709551615)
      --nanoseconds from epoch E
```

- when *U* = 12 and *S* = 32

```
UTCTimeStamp-12-E-32 ::= INTEGER (0..4294967295)
      --picoseconds from epoch E
```

- when U = 12 and S = 64

```
UTCTimeStamp-12-E-64 ::= INTEGER (0..18446744073709551615)
      --picoseconds from epoch E
```

5.4.8 The LocalMktDate-E types

5.4.8.1 The LocalMktDate-E type holds a local date represented as a single integer that is the number of days from a reference epoch.

5.4.8.2 "LocalMktDate-E" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which E stands for a string indicating the reference epoch in ISO 8601 format.

5.4.8.3 On the first invocation of this subclause 5.4.8 for a given E, the following ASN.1 type assignment shall be generated, with the letter E replaced with the actual epoch specified in the subclause that invokes this subclause 5.4.8:

```
LocalMktDate-E ::= INTEGER (0..65535)  --days from epoch E
```

5.4.9 The TZTimeOnly-U types

5.4.9.1 A TZTimeOnly-U type holds a time-of-day with timezone information. The time is expressed as UTC time with an optional time offset from UTC. The time-of-day is represented as a single integer that is the number of time units since midnight. The time unit can be a second, a millisecond, a microsecond, a nanosecond, or a picosecond. The time offset is expressed in minutes (either positive or negative).

5.4.9.2 "TZTimeOnly-U" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which U stands for a number denoting the time unit (0=second, 3=millisecond, 6=microsecond, 9=nanosecond, 12=picosecond).

5.4.9.3 On the first invocation of this subclause 5.4.9 for a given U, exactly one of the following ASN.1 type assignments shall be generated:

- when U = 0

```
TZTimeOnly-0 ::= SEQUENCE {
      time               INTEGER (0..87839),
            --seconds since midnight
      timeOffset         INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when U = 3

```
TZTimeOnly-3 ::= SEQUENCE {
      time               INTEGER (0..87839999),
```

```
            --milliseconds since midnight
      timeOffset          INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 6

```
TZTimeOnly-6 ::= SEQUENCE {
      time                INTEGER (0..87839999999),
            --microseconds since midnight
      timeOffset          INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 9

```
TZTimeOnly-9 ::= SEQUENCE {
      time                INTEGER (0..87839999999999),
            --nanoseconds since midnight
      timeOffset          INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 12

```
TZTimeOnly-12 ::= SEQUENCE {
      time                INTEGER (0..87839999999999999),
            --picoseconds since midnight
      timeOffset          INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

5.4.10 The TZTimeStamp-*U*-*E*-*S* types

5.4.10.1 A TZTimeStamp-*U*-*E*-*S* type holds a date-and-time with timezone information. The time can be expressed as UTC time with an optional time offset from UTC. The date-and-time is represented as a single integer that is the number of time units from a reference epoch. The time unit can be a second, a millisecond, a microsecond, a nanosecond, or a picosecond. The time offset is expressed in minutes (either positive or negative).

5.4.10.2 "TZTimeStamp-*U*-*E*-*S*" is not an actual ASN.1 type name but a template for an ASN.1 type name, in which *U* stands for a number denoting the time unit (0=second, 3=millisecond, 6=microsecond, 9=nanosecond, 12=picosecond), *E* stands for a string indicating the reference epoch in ISO 8601 format, and *S* stands for a number indicating the size of the integer (either 32 or 64 bits).

5.4.10.3 On the first invocation of this subclause 5.4.10 for given *U*, *E*, and *S*, exactly one of the following ASN.1 type assignments shall be generated, with the letter *E* replaced with the actual string specified in the subclause that invokes this subclause 5.4.10:

- when *U* = 0 and *S* = 32

```
TZTimeStamp-0-E-32 ::= SEQUENCE {
      timeStamp              INTEGER (0..4294967295),
            --seconds from epoch E
      timeOffset             INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 0 and *S* = 64

```
TZTimeStamp-0-E-64 ::= SEQUENCE {
      timeStamp              INTEGER (0..18446744073709551615),
            --seconds from epoch E
      timeOffset             INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 3 and *S* = 32

```
TZTimeStamp-3-E-32 ::= SEQUENCE {
      timeStamp              INTEGER (0..4294967295),
            --milliseconds from epoch E
      timeOffset             INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 3 and *S* = 64

```
TZTimeStamp-3-E-64 ::= SEQUENCE {
      timeStamp              INTEGER (0..18446744073709551615),
            --milliseconds from epoch E
      timeOffset             INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 6 and *S* = 32

```
TZTimeStamp-6-E-32 ::= SEQUENCE {
      timeStamp              INTEGER (0..4294967295),
            --microseconds from epoch E
      timeOffset             INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when *U* = 6 and *S* = 64

```
TZTimeStamp-6-E-64 ::= SEQUENCE {
      timeStamp              INTEGER (0..18446744073709551615),
```

```
            --microseconds from epoch E
      timeOffset              INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when U = 9 and S = 32

```
TZTimeStamp-9-E-32 ::= SEQUENCE {
      timeStamp               INTEGER (0..4294967295),
            --nanoseconds from epoch E
      timeOffset              INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when U = 9 and S = 64

```
TZTimeStamp-9-E-64 ::= SEQUENCE {
      timeStamp               INTEGER (0..18446744073709551615),
            --nanoseconds from epoch E
      timeOffset              INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when U = 12 and S = 32

```
TZTimeStamp-12-E-32 ::= SEQUENCE {
      timeStamp               INTEGER (0..4294967295),
            --picoseconds from epoch E
      timeOffset              INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

- when U = 12 and S = 64

```
TZTimeStamp-12-E-64 ::= SEQUENCE {
      timeStamp               INTEGER (0..18446744073709551615),
            --picoseconds from epoch E
      timeOffset              INTEGER (-900..900) DEFAULT 0
            --minutes from UTC
}
```

5.4.11 The `BinaryString` type

5.4.11.1 The `BinaryString` type holds a string of arbitrary octets.

5.4.11.2 On the first invocation of this subclause 5.4.11, the following ASN.1 type assignment shall be generated:

```
BinaryString ::= OCTET STRING
```

5.4.12 The XMLString type

5.4.12.1 The XMLString type holds an XML fragment, i.e., either an XML element or two or more concatenated XML elements.

5.4.12.2 On the first invocation of this subclause 5.4.12, the following ASN.1 type assignment shall be generated:

```
XMLString ::= UTF8String
```

5.4.13 The Duration type

5.4.13.1 The Duration type holds a number of days, weeks, months, or years.

5.4.13.2 On the first invocation of this subclause 5.4.13, the following ASN.1 type assignment shall be generated:

```
Duration ::= CHOICE {
      days        INTEGER (1..MAX),
      weeks       INTEGER (1..MAX),
      months      INTEGER (1..MAX),
      years       INTEGER (1..MAX)
}
```

5.4.14 The YearAndMonth type

5.4.14.1 The YearAndMonth type holds a year, a month, and optionally either a day or a week.

5.4.14.2 On the first invocation of this subclause 5.4.14, the following ASN.1 type assignment shall be generated:

```
YearAndMonth ::= SEQUENCE {
      year                    INTEGER (0..4095),
      month                   INTEGER (1..12),
      dayOrWeek         CHOICE {
            day               INTEGER (1..31),
            week              INTEGER (1..5)
      } OPTIONAL
}
```

5.5 Datatype mapping summary

Tables 5, 6, and 7 contain a summary of the mapping to ASN.1 of all the datatypes defined in the FIX 5.0 SP2 Unified Repository.

Table 5 – Datatype mapping summary (part 1)

FIX datatype	FIX base datatype	XSD datatype	min Value	max Value	min Length	max Length	is UTF8	ASN.1 type

int		integer	-infinity	+infinity				INTEGER
Length	int	nonNegative Integer						INTEGER (0..MAX)
TagNum	int	nonNegative Integer						INTEGER (0..MAX)
SeqNum	int	positive Integer						INTEGER (1..MAX)
NumInGroup	int	*(none)*						INTEGER (0..MAX)
DayOfMonth	int	*(none)*						INTEGER (1..31)
char		string						IA5String (SIZE (1))
Boolean	char	string						BOOLEAN
String		string			0	+infinity		IA5String
Multiple CharValue	String	string			0	+infinity		IA5String
Multiple StringValue	String	string			0	+infinity		IA5String
Country	String	string						IA5String (SIZE (2))
Currency	String	string						IA5String (SIZE (3))
Exchange	String	string			0	+infinity		IA5String
data	String	string					false	BinaryString
Pattern		*(none)*						IA5String
Tenor	Pattern	string						Duration
MonthYear	String	string						YearAndMonth
Reserved100Plus	Pattern	integer						INTEGER (100..MAX)
Reserved1000Plus	Pattern	integer						INTEGER (1000..MAX)
Reserved4000Plus	Pattern	integer						INTEGER (4000..MAX)
XMLData	String	string						XMLString
Language	String	language			0	+infinity		IA5String
field with enumerated values	*(any)*	*(any)*						an ENUMERATED type and/or a BIT STRING type
field with a union datatype	*(any)*	*(any)*						a CHOICE type

Table 6 – Datatype mapping summary (part 2)

FIX datatype	FIX base datatype	XSD datatype	fixedPoint	exponent	numBits	ASN.1 type
float		decimal	false	0	64	Decimal-var0-64
Qty	float	decimal	false	0	64	Decimal-var0-64
Price	float	decimal	false	0	64	Decimal-var0-64
PriceOffset	float	decimal	false	0	64	Decimal-var0-64
Amt	float	decimal	false	0	64	Decimal-var0-64
Percentage	float	decimal	false	0	64	Decimal-var0-64

Table 7 – Datatype mapping summary (part 3)

FIX datatype	FIX base datatype	XSD datatype	timeUnit	epoch	numBits	ASN.1 type
UTCDateOnly	String	date		19700101		UTCDateOnly-19700101
UTCTimeOnly	String	time	9			UTCTimeOnly-9
UTCTimestamp	String	dateTime	9	19700101	64	UTCTimeStamp-9-19700101-64
LocalMktDate	String	date		19700101		LocalMktDate-19700101
TZTimeOnly	String	time	9			TZTimeOnly-9
TZTimestamp	String	dateTime	9	19700101	64	TZTimeStamp-9-19700101-64

6 Mapping of FIX messages

6.1 The <messages> element in the *source* <fix> *element* contains a list of <message> elements each describing one FIX message. A message consists of a sequence of fields and/or components, each either required or optional.

6.2 For each <message> element in the <messages> element of the *source* <fix> *element*, in order, an ASN.1 type assignment shall be generated as specified in subclauses 6.2.1 to 6.2.2.

6.2.1 The type name on the left side of the type assignment shall be generated from the name of the FIX message with the "-message" suffix appended, in accordance with subclause 4.3.

6.2.2 The type expression on the right side of the type assignment shall be determined as specified in subclauses 7.3.2 to 7.3.9 for the mapping of a <component> element not consisting of a repeating group. The type expression shall be preceded by a textual ASN.1 tag of the *context-specific* class with the tag number equal to the id attribute of the <message> element.

EXAMPLE

The NewOrderList message of the FIX Repository v.5.0 SP2 will generate the following ASN.1 type assignment:

```
NewOrderList-message ::= [15] SEQUENCE {
      standardHeader             [1024]             StandardHeader,
      listID                     [APPLICATION 66]   IA5String,
      bidID                      [APPLICATION 390]  IA5String
                                                      OPTIONAL,
      clientBidID                [APPLICATION 391]  IA5String
                                                      OPTIONAL,
      progRptReqs                [APPLICATION 414]  ProgRptReqs-enum
                                                      OPTIONAL,
      bidType                    [APPLICATION 394]  BidType-enum,
      progPeriodInterval         [APPLICATION 415]  INTEGER
                                                      OPTIONAL,
      cancellationRights         [APPLICATION 480]
                                 CancellationRights-enum OPTIONAL,
      moneyLaunderingStatus      [APPLICATION 481]
                                 MoneyLaunderingStatus-enum OPTIONAL,
      registID                   [APPLICATION 513]  IA5String
                                                      OPTIONAL,
      listExecInstType           [APPLICATION 433]
                                 ListExecInstType-enum      OPTIONAL,
      listExecInst               [APPLICATION 69]   IA5String
                                                      OPTIONAL,
      contingencyType            [APPLICATION 1305]
                                 ContingencyType-union      OPTIONAL,
      encodedListExecInst        [APPLICATION 353]
                                 BinaryString               OPTIONAL,
```

```
      allowableOneSidednessPct   [APPLICATION 765]  Percentage
                                                      OPTIONAL,
      allowableOneSidednessValue [APPLICATION 766]  Amt
                                                      OPTIONAL,
      allowableOneSidednessCurr  [APPLICATION 767]  Currency
                                                      OPTIONAL,
      totNoOrders                [APPLICATION 68]   INTEGER,
      lastFragment               [APPLICATION 893]
                              LastFragment-enum          OPTIONAL,
      rootParties-list           [1031]             RootParties-list
                                                      OPTIONAL,
      listOrdGrp-list            [2030]             ListOrdGrp-list,
      throttleInst               [APPLICATION 1685] ThrottleInst-enum
                                                      OPTIONAL,
      standardTrailer            [1025]             StandardTrailer,
      ...
}
```

which contains references to the following ASN.1 type assignments separately generated from <component>, <field>, and <datatype> elements:

```
StandardHeader ::= SEQUENCE {
      applVerID       [APPLICATION 1128] ApplVerID-enum  OPTIONAL,
      *** etc. ***
}

StandardTrailer ::= SEQUENCE {
      signature       [APPLICATION 89] BinaryString OPTIONAL
}

ApplVerID-enum ::= ENUMERATED {
      fIX27,
      fIX30,
      fIX40,
      fIX41,
      fIX42,
      fIX43,
      fIX44,
      fIX50,
      fIX50SP1,
      fIX50SP2,
      ...
}

ProgRptReqs-enum ::= ENUMERATED {
      buySideRequests            (1),
      sellSideSends              (2),
      realTimeExecutionReports   (3),
      ...
```

```
}

BidType-enum ::= ENUMERATED {
      nonDisclosed      (1),
      disclosed         (2),
      noBiddingProcess  (3),
      ...
}

CancellationRights-enum ::= ENUMERATED {
      yes,
      noExecutionOnly,
      noWaiverAgreement,
      noInstitutional,
      ...
}

MoneyLaunderingStatus-enum ::= ENUMERATED {
      passed,
      notChecked,
      exemptBelowLimit,
      exemptMoneyType,
      exemptAuthorised,
      ...
}

ListExecInstType-enum ::= ENUMERATED {
      immediate,
      waitForInstruction,
      sellDriven,
      buyDrivenCashTopUp,
      buyDrivenCashWithdraw,
      ...
}

ContingencyType-enum ::= ENUMERATED {
      oneCancelsTheOther               (1),
      oneTriggersTheOther              (2),
      oneUpdatesTheOtherAbsolute       (3),
      oneUpdatesTheOtherProportional   (4),
      bidAndOffer                      (5),
      bidAndOfferOCO                   (6),
      ...
}

ContingencyType-union ::= CHOICE {
      basic   ContingencyType-enum,
      ext     Reserved100Plus
}
```

```
ThrottleInst-enum ::= ENUMERATED {
      rejectIfThrottleLimitExceeded  (0),
      queueIfThrottleLimitExceeded   (1),
      ...
}

ListOrdGrp-list ::= SEQUENCE OF ListOrdGrp

ListOrdGrp ::= SEQUENCE {
      clOrdID                [APPLICATION 11]  IA5String,
      secondaryClOrdID       [APPLICATION 526] IA5String OPTIONAL,
      *** etc. ***
      ...
}

LastFragment-enum ::= ENUMERATED {
      notLastMessage,
      lastMessage,
      ...
}

RootParties-list ::= SEQUENCE OF RootParties

RootParties ::= SEQUENCE {
      rootPartyID           [APPLICATION 1117] IA5String
                                                     OPTIONAL,
      rootPartyIDSource     [APPLICATION 1118] PartyIDSource-enum
                                                     OPTIONAL,
      rootPartyRole         [APPLICATION 1119] PartyRole-enum
                                                     OPTIONAL,
      rootSubParties-list   [2097]             RootSubParties-list
                                                     OPTIONAL,
      ...
}

RootSubParties-list ::= SEQUENCE OF RootSubParties

RootSubParties ::= SEQUENCE {
      rootPartySubID        [APPLICATION 1121] IA5String
                                                     OPTIONAL,
      rootPartySubIDType    [APPLICATION 1122] PartySubIDType-union
                                                     OPTIONAL,
      ...
}

PartyIDSource-enum ::= ENUMERATED {
      uKNationalInsuranceOrPensionNumber,
      uSSocialSecurityNumber,
```

```
      uSEmployerOrTaxIDNumber,
      australianBusinessNumber,
      *** etc. ***
      ...
}

PartyRole-enum ::= ENUMERATED {
      executingFirm    (1),
      brokerOfCredit   (2),
      clientID         (3),
      clearingFirm     (4),
      *** etc. ***
      ...
}

PartySubIDType-enum ::= ENUMERATED {
      firm                 (1),
      person               (2),
      system               (3),
      application          (4),
      fullLegalNameOfFirm  (5),
      postalAddress        (6),
      *** etc. ***
      ...
}

PartySubIDType-union ::= CHOICE {
      basic    PartySubIDType-enum,
      ext      Reserved4000Plus
}

Reserved100Plus ::= INTEGER (100..MAX)

BinaryString ::= OCTET STRING

Amt ::= Decimal-var0-64

Currency ::= IA5String (SIZE (3))
```

7 Mapping of a FIX component

7.1 This clause applies as explicitly invoked by other clauses of this technical specification to generate either one or two ASN.1 type assignments corresponding to a FIX component.

7.2 The `<components>` element in the *source* `<fix>` *element* contains a list of `<component>` elements each describing one FIX component. A component consists of:

a) a sequence of fields and/or child components, each either required or optional; or
b) a repeating group, which in turn contains a sequence of fields and/or child components, each either required or optional

7.3 On the first invocation of this clause 7 for a given `<component>` element not consisting of a repeating group (case (a) of subclause 7.2), an ASN.1 type assignment shall be generated as specified in subclauses 7.3.1 to 7.3.9.

7.3.1 The type name on the left side of the type assignment shall be generated from the name of the FIX component in accordance with subclause 4.3.

7.3.2 The type expression on the right side of the type assignment shall be a `SEQUENCE` type expression having one component for each `<fieldRef>` and `<componentRef>` child element of the `<component>` element, in order, with the exceptions specified in subclause 7.3.3.

7.3.3 No components of the `SEQUENCE` type shall be generated for the following FIX fields:

- any FIX field of type `Length` that has a non-empty `associatedDataTag` attribute;
- the `BeginString` field (FIX field id = `8`);
- the `BodyLength` field (FIX field id = `9`);
- the `MsgType` field (FIX field id = `35`);
- the `CheckSum` field (FIX field id = `10`).

7.3.4 The identifier of each component of the `SEQUENCE` type shall be generated from:

a) the `name` attribute of the `<componentRef>` element with the "`-list`" suffix appended, in accordance with subclause 4.3, if the referenced component consists of a repeating group;
b) the `name` attribute of the `<fieldRef>` or `<componentRef>` element in accordance with subclause 4.3, otherwise.

7.3.5 The type expression of each component of the `SEQUENCE` type corresponding to a `<fieldRef>` element shall be determined as follows:

a) if the *effective ASN.1 encoding attributes* of the `<fieldRef>` element differ from the *effective ASN.1 encoding attributes* of the `<datatype>` element referenced by the `type` attribute of the referenced `<field>` element, the type expression shall be the *target ASN.1 type expression* determined from the `<fieldRef>` element as specified in subclause 5.3.3;

b) otherwise, the type expression shall be determined as specified in table 8.

Table 8 – Determination of the ASN.1 type of a SEQUENCE component

Case of table 2	Type expression of the component	Reference
1, 2, 5	the type name on the left side of the ASN.1 type assignment generated from the FIX datatype indicated in the type attribute of the referenced <field> element	subclause 5.1.6
3	the type name on the left side of the ENUMERATED type assignment generated from the referenced <field> element	subclause 5.2.2
4	the type name on the left side of the ENUMERATED type assignment generated from the <field> element whose id is indicated in the enumDataType attribute of the referenced <field> element	subclause 5.2.2
6	the type name on the left side of BIT STRING type assignment generated from the referenced <field> element	subclause 5.2.3
7	the type name on the left side of the BIT STRING type assignment generated from the <field> element whose id is indicated in the enumDataType attribute of the referenced <field> element	subclause 5.2.3
8, 9, 10, 11	the type name on the left side of the CHOICE type assignment generated from a union of two datatypes	subclause 5.2.4

NOTE – The inclusion of case 2 of table 2 in the first row of table 8 accounts for the (possibly rare) case in which:

a) the *effective ASN.1 encoding attributes* of the <field> element differ from the *effective ASN.1 encoding attributes* of the <datatype> element; and

b) the *effective ASN.1 encoding attributes* of the <fieldRef> element differ from the *effective ASN.1 encoding attributes* of the <field> element; but

c) the *effective ASN.1 encoding attributes* of the <fieldRef> element are identical to the *effective ASN.1 encoding attributes* of the <datatype> element.

7.3.6 The type expression of each component of the SEQUENCE type corresponding to a <componentRef> element shall be the type name on the left side of the ASN.1 type assignment generated by invoking this clause 7 for the referenced <component> element.

7.3.7 The type expression of each component of the SEQUENCE type corresponding to a <fieldRef> element shall be preceded by a textual ASN.1 tag of the APPLICATION class with the tag number equal to the id attribute of the referenced <field> element.

7.3.8 The type expression of each component of the SEQUENCE type corresponding to a <componentRef> element shall be preceded by a textual ASN.1 tag of the *context-specific* class with the tag number equal to the id attribute of the referenced <component> element.

NOTE – The purpose of this subclause is to provide a textual ASN.1 tag corresponding to the FIX component that is guaranteed to be unique in the context of the SEQUENCE type, while keeping a clear distinction between textual ASN.1 tags assigned to FIX fields (APPLICATION tags) and textual ASN.1 tags assigned to FIX components (*context-specific* tags).

7.3.9 If the ASN.1 type assignment is being generated from either a <message> element or a <component> element consisting of a repeating group, an extension marker (...) shall be added after the last component of the SEQUENCE type.

EXAMPLE 1

The CommissionData component of the FIX Repository v.5.0 SP2 will generate the following ASN.1 type assignment:

```
CommissionData ::= SEQUENCE {
      commission       [APPLICATION 12]  Amt            OPTIONAL,
      commType         [APPLICATION 13]  CommType-enum  OPTIONAL,
      commCurrency     [APPLICATION 479] Currency       OPTIONAL,
      fundRenewWaiv    [APPLICATION 497] FundRenewWaiv-enum OPTIONAL
}
```

which contains references to the following ASN.1 type assignments separately generated from <field> and <datatype> elements:

```
CommType-enum ::= ENUMERATED {
      perUnit,
      percent,
      absolute,
      percentageWaivedCashDiscount,
      percentageWaivedEnhancedUnits,
      pointsPerBondOrContract,
      ...
}

FundRenewWaiv-enum ::= ENUMERATED { no, yes, ... }

Amt ::= Decimal-var0-64

Currency ::= IA5String (SIZE (3))
```

7.4 On the first invocation of this clause 7 for a given <component> element consisting of a repeating group (case (b) of subclause 7.2), two ASN.1 type assignments shall be generated as specified in subclauses 7.4.1 to 7.4.4.

7.4.1 The first type assignment shall be generated in the same way as is specified in subclause 7.3 for a <component> element not consisting of a repeating group (case (a) of subclause 7.2), except that every mention of a child element of the <component> element within that subclause shall be understood as referring to a child element of the <repeatingGroup> element under the <component> element.

7.4.2 The type name on the left side of the second type assignment shall be generated from the name of the FIX component with the "`-list`" suffix appended, in accordance with subclause 4.3.

7.4.3 The type expression on the right side of the second type assignment shall be a `SEQUENCE OF` type expression.

7.4.4 The type expression of the component of the `SEQUENCE OF` type shall be the type name on the left side of the first ASN.1 type assignment.

EXAMPLE 1

The `NestedParties` component of the FIX Repository v.5.0 SP2 will generate the following ASN.1 type assignments:

```
NestedParties-list ::= SEQUENCE OF NestedParties

NestedParties ::= SEQUENCE {
      nestedPartyID          [APPLICATION 524] IA5String
                                                 OPTIONAL,
      nestedPartyIDSource    [APPLICATION 525] PartyIDSource-enum
                                                 OPTIONAL,
      nestedPartyRole        [APPLICATION 538] PartyRole-enum
                                                 OPTIONAL,
      nstdPtysSubGrp-list    [2078]            NstdPtysSubGrp-list
                                                 OPTIONAL,
      ...
}
```

which contain references to the following ASN.1 type assignments separately generated from `<component>`, `<field>`, and `<datatype>` elements:

```
NstdPtysSubGrp-list ::= SEQUENCE OF NstdPtysSubGrp

NstdPtysSubGrp ::= SEQUENCE {
      nestedPartySubID       [APPLICATION 545] IA5String
                                                 OPTIONAL,
      nestedPartySubIDType   [APPLICATION 805] PartySubIDType-union
                                                 OPTIONAL,
      ...
}

PartyIDSource-enum ::= ENUMERATED {
      uKNationalInsuranceOrPensionNumber,
      uSSocialSecurityNumber,
      uSEmployerOrTaxIDNumber,
      australianBusinessNumber,
      *** etc. ***
      ...
}
```

```
PartyRole-enum ::= ENUMERATED {
      executingFirm    (1),
      brokerOfCredit   (2),
      clientID         (3),
      clearingFirm     (4),
      *** etc. ***
      ...
}

PartySubIDType-enum ::= ENUMERATED {
      firm                 (1),
      person               (2),
      system               (3),
      application          (4),
      fullLegalNameOfFirm  (5),
      postalAddress        (6),
      *** etc. ***
      ...
}

PartySubIDType-union ::= CHOICE {
      basic    PartySubIDType-enum,
      ext      Reserved4000Plus
}

Reserved4000Plus ::= INTEGER (4000..MAX)
```

8 Message Encoding Header for use with ASN.1 Encodings

8.1 When a value of an ASN.1 type generated from a FIX message (see **6.2**) is encoded using PER or OER, the encoded message does not carry any indication of the FIX message type.

8.2 The lack of such an indication in an encoded message is a characteristic of PER and OER. In general, when an ASN.1 schema defines multiple top-level ASN.1 types, it is assumed that the top-level type used in a particular instance of a message will be conveyed either in a message of a higher-level protocol containing the PER- or OER-encoded message, or in a short header preceding the PER- or OER-encoded message.

8.3 A message encoding header for use with ASN.1 encodings of FIX messages is a data structure defined as follows:

Field	Size (bits)	Description
Schema ID	16	A number that identifies an ASN.1 schema
Schema Version	16	The version number of the ASN.1 schema
Top-Level Type ID	32	A number that identifies the top-level ASN.1 type to which the present message conforms

where all the fields are unsigned integers in big-endian order.

8.4 The fields *Schema ID* and *Schema Version*, taken together, identify a particular ASN.1 schema containing the top-level ASN.1 type definition to which the current message conforms. If there are multiple ASN.1 schemas with the same *Schema ID* but different *Schema Version*s, there is no requirement that all those schemas contain a definition for the present message with the same *Top-Level Type ID*.

8.5 The field *Top-Level Type ID* identifies a particular top-level ASN.1 type definition present in the schema identified by *Schema ID* and *Schema Version*.

8.6 The mapping between *Top-Level Type ID* values and top-level ASN.1 types within a schema is not standardized. The following two methods are recommended, under the assumption that the only top-level ASN.1 types of interest have a name ending with "`-message`" (see **6.2**):

a) if all the ASN.1 types whose name ends with "`-message`" present in the schema have distinct textual ASN.1 tag numbers, then the tag number should be used directly as the *Top-Level Type ID* (see the example in clause 6, where the type `NewOrderList-message` has the ASN.1 tag number 15);
b) otherwise, all the ASN.1 types whose names end with "`-message`" present in the schema should be assigned a number (1, 2, 3, etc.) according to the lexicographic order of their names.