INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

X.680 (2002)

Amendment 1
(10/2003)

SERIES X: DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI networking and system aspects – Abstract Syntax Notation One (ASN.1)

Information technology – Abstract Syntax Notation One (ASN.1) –Specification of basic notation

Amendment 1:

to ITU-T Rec. X.680 | ISO/IEC 8824-1

INFORMATION TECHNOLOGY – ABSTRACT SYNTAX NOTATION ONE (ASN.1) – SPECIFICATION OF BASIC NOTATION

DRAFT AMENDMENT 1 (to ITU-T Rec. X.680 | ISO/IEC 8824-1)

1 Summary

An Amendment 1 is provided for ITU-T Rec. X.680 | ISO/IEC 8824-1, ITU-T Rec. X.681 | ISO/IEC 8824-2, ITU-T Rec. X.690 | ISO/IEC 8825-1, ITU-T Rec. X.691 | ISO/IEC 8825-2 and ITU-T Rec. X.693 | ISO/IEC 8825-4. These amendments provide the following:

– Correction of a bug in CXER resulting from allowing white-space between a minus sign and a following INTEGER or REAL value (CXER was not canonical). This is no longer permitted, in value notation, XML Value Notation or in XER and CXER. **This is a change and not an addition**.

– Addition of encoding instructions in an ASN.1 module, using either a type prefix or within an encoding control section, in order to specify variations of the BASIC-XER encodings. These encoding instructions are designed to support mappings from an XSD specification to an ASN.1 specification. This provision has meant a change of terminology, where a type with "`[...]`" in front of it is a prefixed type, and the "`[...]`" notation may or may not be a tag. This change of terminology results in changes to the text (but not the substance) of the BER and PER specifications.

– The addition of NaN (Not-a-Number) and minus zero as new values for REAL (support for encoding these new values is provided in Amendment 1 to ITU-T Rec. X.690 | ISO/IEC 8825-1 and to ITU-T Rec. X.691 | ISO/IEC 8825-2, as well as in Amendment 1 to ITU-T Rec. X.693 | ISO/IEC 8825-4).

– The addition of new XML Value Notations for **`REAL`**, **`BOOLEAN`**, **`ENUMERATED`**, and **`INTEGER`** that use text rather than empty-element tags for the values. These are available in XML Value Notation and in EXTENDED-XER, but not in BASIC-XER (for reasons of backwards-compatibility).

– Changes to the XML Value Notation for sequence-of (and the XER encodings) to provide delimitation of values where they are not XML elements (this occurs with the additional XML Value Notations, and only affects use of those additional XML Value Notations). This change is only concerned with use of XML Value Notations that have been added by this amendment, and these are not allowed in BASIC-XER, which is not affected.

This provides the necessary basic support for EXTENDED-XER.

INFORMATION TECHNOLOGY – ABSTRACT SYNTAX NOTATION ONE (ASN.1) – SPECIFICATION OF BASIC NOTATION

DRAFT AMENDMENT 1 (to ITU-T Rec. X.680 | ISO/IEC 8824-1)

Support for EXTENDED-XER

NOTE: All new or changed text in this document is highlighted in yellow in clauses being replaced. When merging all such text into the base document the highlighting is to be removed.

In the Introduction, insert the following paragraph immediately prior to the paragraph that begins with "An ASN.1 specification will initially be produced with a set of fully defined ASN.1 types.":

It is also possible to assign encoding instructions to a type in order to affect the encoding of that type. This can be done either by a type prefix placed before a type definition or use of a type reference, or by an encoding control section placed at the end of an ASN.1 module. The generic syntax of type prefixes and encoding control sections is specified in this Recommendation | International Standard, and includes an encoding reference to identify the encoding rules that are modified by the encoding instruction. The semantics and detailed syntax of encoding instructions are specified in the encoding rules Recommendation | International Standard identified by the encoding reference.

In the Introduction, insert the following paragraph between the paragraphs beginning with "Annex C" and "Annex D":

Annex Cbis forms an integral part of this Recommendation | International Standard and specifies the currently defined encoding references and the Recommendation | International Standard that defines the semantics and detailed syntax of encoding instructions with those encoding references.

In subclause 2.2, replace the NOTE under "The Unicode Standard" with the following:

NOTE – The above reference is included because it provides names for control characters and specifies categories of characters.

Insert a new subclause 3.6.18bis as follows:

3.6.18bis default encoding reference (for a module): An encoding reference that is specified in the module header and is assumed in all type prefixes which do not contain an encoding reference.

NOTE – If a default encoding reference is not specified in the module header, then all type prefixes which do not contain an encoding reference are assigning tags.

Insert 3 new subclauses 3.6.22bis – quat as follows:

3.6.22bis encoding control section: Part of an ASN.1 module that enables encoding instructions to be assigned to types defined or used within that ASN.1 module.

3.6.22ter encoding instruction: Information which can be associated with a type using a type prefix or an encoding control section, and which affects the encoding of that type by one or more ASN.1 encoding rules.

NOTE – An encoding instruction does not affect the abstract values of a type, and is not expected to be visible to an application.

3.6.22quat encoding reference: A name (see Annex Cbis) that identifies which encoding rules are affected by an encoding instruction in a type prefix or an encoding control section.

NOTE – The encoding reference `TAG` can be used to specify that a type prefix is assigning a tag rather than an encoding instruction (see 30.2).

Replace subclause 3.6.54 with the following:

3.6.54 real type: A simple type whose distinguished values (specified in clause 20) include the set of real numbers (numerical real numbers) together with special values such as **`NOT-A-NUMBER`**.

Replace subclause 6.6.69 with the following:

3.6.69 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.

NOTE – Tag information is used in some encoding rules to ensure that encodings are not ambiguous. Tag information differs from encoding instructions because tag information is associated with all ASN.1 types, even if they do not have a type prefix.

Replace subclause 6.6.71 with the following:

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

Insert a new subclause 3.6.74bis as follows:

3.6.74bis type prefix: Part of the ASN.1 notation that can be used to assign an encoding instruction or a tag to a type.

Replace subclause 8.1 with the following:

8.1 A tag is specified (either within the text of this Recommendation | International Standard or by using a type prefix) by giving a class and a number within the class. The class is one of:

- universal;
- application;
- private;
- context-specific.

*In subclause 8.3 replace "*clause 30*" with "*30.2*", and in the NOTE replace "*Clause 30*" with "*Subclause 30.2*".*

Insert a new clause 8bis as follows:

8bis Encoding instructions

8bis.1 An encoding instruction is assigned to a type using either a type prefix (see 30.3) or an encoding control section (see clause 50).

8bis.2 A type prefix may contain an encoding reference. If it does not, the encoding reference is determined by the default encoding reference for the module (see 12.4bis).

8bis.3 An encoding control section always contains an encoding reference. There may be multiple encoding control sections, but each encoding control section shall have a distinct encoding reference.

8bis.4 An encoding instruction consists of a sequence of lexical items specified in the Recommendation | International Standard determined by the encoding reference (see Annex Cbis).

8bis.5 Multiple encoding instructions with the same or with different encoding references may be assigned to a type (using either or both of type prefixes and an encoding control section). Encoding instructions assigned with a given encoding reference are independent from those assigned with a different encoding reference, and from any use of a type prefix to perform tagging.

8bis.6 The effect of assigning several encoding instructions with the same encoding reference (using either or both of type prefixes and an encoding control section) is specified in the Recommendation | International Standard determined by the encoding reference (see Annex Cbis), and is not specified in this Recommendation | International Standard.

8bis.7 If an encoding instruction is assigned to the "Type" in a "TypeAssignment", it becomes associated with the type, and is applied wherever the "typereference" of the "TypeAssignment" is used. This includes use in other modules through the export and import statements.

Insert a new subclause 11.20bis as follows:

11.20bis Encoding references

Name of item – encodingreference

An "encodingreference" shall consist of a sequence of characters as specified for a "typereference" in 11.2, except that no lower-case letters shall be included.

NOTE – Currently defined encoding references are listed in Annex Cbis with the Recommendation | International Standard that specifies the syntax and semantics of the corresponding encoding instructions. The "encodingreference" shall consist only of the sequences listed in Annex Cbis in this or in future versions of this Recommendation | International Standard.

Replace subclause 11.23.2 with the following:

11.23.2 In analyzing an instance of use of this notation, a "true" is distinguished from a "valuereference" or an "identifier" or an instance of XML boolean "extended-true" by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2).

Insert a new subclause 11.23bis as follows:

11.23bis XML boolean extended-true item

Name of item – extended-true

11.23bis.1 This item shall consist of either the sequence of characters:

true

or of the single character

1 (DIGIT ONE)

11.23bis.2 In analyzing an instance of use of this notation, an "extended-true" is distinguished from a "valuereference" or an "identifier" or an instance of XML boolean "true" by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2).

Replace subclause 11.24.2 with the following:

11.24.2 In analyzing an instance of use of this notation, a "false" is distinguished from a "valuereference" or an "identifier" or an instance of XML boolean "extended-false" by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2).

Insert new subclauses 11.24bis, ter and quat as follows:

11.24bis XML boolean extended-false item

Name of item – extended-false

11.24bis.1 This item shall consist of either the sequence of characters:

false

or of the single character

0 (DIGIT ZERO)

11.24bis.2 In analyzing an instance of use of this notation, a "false" is distinguished from a "valuereference" or an "identifier" or an instance of XML boolean "false" by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2).

11.24ter XML real not-a-number item

Name of item – "NaN"

11.24ter.1 This item shall consist of the sequence of characters:

NaN

11.24ter.2 In analyzing an instance of use of this notation, a "NaN" is distinguished from any other lexical item commencing with an upper-case letter by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2).

11.24quat XML real infinity item

Name of item – "INF"

11.24quat.1 This item shall consist of the sequence of characters:

INF

11.24quat.2 In analyzing an instance of use of this notation, an "INF" is distinguished from any other lexical item commencing with an upper-case letter by the context in which it appears.

NOTE – This sequence does not contain any white-space characters (see 11.1.2)

Replace subclause 11.25.5 with the following:

11.25.5 If the ASN.1 built-in type is a "PrefixedType" then the type which determines the "xmlasn1typename" shall be "Type" in the "PrefixedType" (see 30.1.5). If this is itself a "PrefixedType", then this subclause 11.25.5 shall be recursively applied.

NOTE – The subclauses of 25.11 specify the "Type" to be used for a "SelectionType" and a "ConstrainedType".

In table 4, replace "TaggedType" *with* "PrefixedType".

In subclause 11.27 insert the following 3 new reserved words in new cells in the table in the appropriate alphabetical position:

ENCODING-CONTROL

INSTRUCTIONS

NOT-A-NUMBER

In subclause 12.1, replace the production "ModuleDefinition" with the following:

```
ModuleDefinition ::=
        ModuleIdentifier
        DEFINITIONS
        EncodingReferenceDefault
        TagDefault
        ExtensionDefault
        "::="
        BEGIN
        ModuleBody
        EncodingControlSections
        END
```

and insert the following new production immediately before the "TagDefault" production:

```
EncodingReferenceDefault ::=
        encodingreference INSTRUCTIONS |
        empty
```

In subclause 12.2 in the NOTE, replace "Clause 30" *with* "Subclause 30.2".

Insert a new subclause 12.4bis as follows:

12.4bis The "EncodingReferenceDefault" specifies that the "encodingreference" is the default encoding reference for the module. If the "EncodingReferenceDefault" is "empty", then the default encoding reference for the module is **TAG**.

NOTE – Annex Cbis contains a list of allowed encoding references, together with the Recommendation | International Standard which specifies the form and meaning of the corresponding encoding instructions.

Insert a new subclause 12.21 as follows:

12.21 "EncodingControlSections" is specified in clause 50.

In subclause 13.3 in the NOTE, replace "XML tags" *with* "XML tag names".

In subclause 16.2, replace the 2 occurrences of "TaggedType" *with* "PrefixedType".

In subclause 16.9, replace "TaggedValue" *with* "PrefixedValue".

*In subclause 16.10, replace "*XMLTaggedValue*" with "*XMLPrefixedValue*".*

Replace subclause 17.3 with the following:

17.3 The value of a boolean type (see 3.6.73 and 3.6.38) shall be defined by the notation "BooleanValue", or when used as an "XMLValue", by the notation "XMLBooleanValue". These productions are:

```
BooleanValue ::= TRUE | FALSE

XMLBooleanValue ::=
                    EmptyElementBoolean
     |              TextBoolean

EmptyElementBoolean ::=
                    "<" & "true" "/>"
     |               "<" & "false" "/>"

TextBoolean ::=
                    extended-true
     |              extended-false
```

Insert a new subclause 17.4 as follows:

17.4 If an "EmptyElementBoolean" appears in an "XMLValueAssignment", then there shall be no occurrence of "TextBoolean" in that "XMLValueAssignment".

Replace subclause 18.9 with the following:

18.9 The value of an integer type shall be defined by the notation "IntegerValue", or when used as an "XMLValue", by the notation "XMLIntegerValue". These productions are:

```
IntegerValue ::=
          SignedNumber |
          identifier

XMLIntegerValue ::=
          XMLSignedNumber
     |    EmptyElementInteger
     |    TextInteger

XMLSignedNumber ::=
          number
     |    "-" & number

EmptyElementInteger ::=
          "<" & identifier "/>"

TextInteger ::=
          identifier
```

Add a new subclause 18.9bis as follows:

18.9bis If an "EmptyElementInteger" appears in an "XMLValueAssignment", then there shall be no occurrence of "TextInteger" in that "XMLValueAssignment".

Replace subclause 18.10 and its NOTE with the following:

18.10 The "identifier" in "IntegerValue" and in the last two alternatives for "XMLIntegerValue" shall be one of the "identifier"s in the "IntegerType" with which the value is associated, and shall represent the corresponding number.

NOTE – When referencing an integer value for which an "identifier" has been defined, use of the "identifier" form of "IntegerValue" and one of the "identifier" forms of "XMLIntegerValue" should be preferred.

Add a new subclause 18.12 as follows:

18.12 The second alternative of "XMLSignedNumber" shall not be used if the "number" is zero.

Replace subclause 19.8 with the following:

19.8 The value of an enumerated type shall be defined by the notation "EnumeratedValue", or when used as an "XMLValue", by the notation "XMLEnumeratedValue". These productions are:

```
EnumeratedValue ::= identifier

XMLEnumeratedValue ::=
        EmptyElementEnumerated
    |   TextEnumerated

EmptyElementEnumerated ::= "<" & identifier "/>"

TextEnumerated ::= identifier
```

Add a new subclause 19.8bis as follows:

19.8bis If an "EmptyElementEnumerated" appears in an "XMLValueAssignment", then there shall be no occurrence of "TextEnumerated" in that "XMLValueAssignment".

Replace subclause 19.9 with the following:

19.9 The "identifier" in "EnumeratedValue" and in the two alternatives of "XMLEnumeratedValue" shall be equal to that of an "identifier" in the "EnumeratedType" sequence with which the value is associated.

Replace subclauses, 20.3, 20.4, 20.5 and 20.6 with the following:

20.3 The abstract values of the real type are the special values **`PLUS-INFINITY`**, **`MINUS-INFINITY`**, and **`NOT-A-NUMBER`** together with numeric real numbers consisting of either plus zero or minus zero, or capable of being specified by the following formula involving three integers, M, B and E:

$$M \times B^{E}$$

where M (non-zero) is called the mantissa, B (either 2 or 10) the base, and E the exponent. Values with B = 2 ("**`base`**" 2 abstract values) and B = 10 ("**`base`**" 10 abstract values) are defined as distinct abstract values. Otherwise, values of $M \times B^{E}$ which evaluate to the same numerical value are a single abstract value.

NOTE – Minus zero and plus zero are two distinct abstract values for a mathematical zero, and the "**`base`**" 2 and "**`base`**" 10 abstract values are distinct abstract values for all other numeric real numbers.

20.4 The real type has an associated type which is used to support the value and subtype notations for numeric values of the real type (in addition to the notation for the special values of the real type and for plus zero and minus zero).

NOTE – Encoding rules may define a different type which is used to specify encodings, or may specify encodings without reference to the associated type. In particular, the encoding in BER and PER provides a Binary-Coded Decimal (BCD) encoding if "base" is 10, and an encoding which permits efficient transformation to and from hardware floating point representations if "base" is 2.

20.5 The associated type for value definition (and for subtyping purposes) of the numeric values is (with normative comments):

```
SEQUENCE {
      mantissa   INTEGER (ALL EXCEPT 0),
      base  INTEGER (2|10),
      exponent   INTEGER

      -- The associated mathematical real number is "mantissa"

      -- multiplied by "base" raised to the power "exponent"
}
```

NOTE 1 – Values represented by "**`base`**" 2 and by "**`base`**" 10 are considered to be distinct abstract values even if they evaluate to the same real number value, and may carry different application semantics.

NOTE 2 – The notation **`REAL (WITH COMPONENTS { ... , base (10)})`** can be used to restrict the set of values to the "**`base`**" 10 numeric values (and similarly for "**`base`**" 2 numeric values). This notation does not include the values (special real values and plus and minus zero) that cannot be represented by the associated type. If required, these can be added using set arithmetic.

NOTE 3 – This type is capable of carrying an exact finite representation of any number which can be stored in typical floating point hardware, and of any number with a finite character-decimal representation.

20.6 The value of a real type shall be defined by the notation "RealValue", or when used in an "XMLValue", by the notation "XMLRealValue":

```
RealValue ::=
        NumericRealValue
    |   SpecialRealValue

NumericRealValue ::=
        realnumber
    |   "-" realnumber
    |   SequenceValue        -- Value of the associated sequence type

SpecialRealValue ::=
        PLUS-INFINITY
    |   MINUS-INFINITY
    |   NOT-A-NUMBER
```

NOTE – The third alternative of "NumericRealValue" cannot be used for plus zero or minus zero values. These abstract values are specified using either the first or the second alternative respectively, with a single "**0**" character for "realnumber".

```
XMLRealValue ::=
        XMLNumericRealValue | XMLSpecialRealValue

XMLNumericRealValue ::=
        realnumber
    |   "-" & realnumber

XMLSpecialRealValue ::=
        EmptyElementReal
    |   TextReal

EmptyElementReal ::=
        "<" & PLUS-INFINITY "/>"
    |   "<" & MINUS-INFINITY "/>"
    |   "<" & NOT-A-NUMBER "/>"

TextReal ::=
        "INF"
    |   "-" & "INF"
    |   "NaN"
```

Add a new subclause 20.6bis as follows:

20.6bis If an "EmptyElementReal" appears in an "XMLValueAssignment", then there shall be no occurrence of "TextReal" in that "XMLValueAssignment".

Replace subclause 20.7 with the following:

20.7 When the "realnumber" notation is used it identifies the corresponding "**base**" 10 abstract value, or plus zero. When a "realnumber" value is preceded by "**-**", it identifies the corresponding "**base**" 10 abstract values that are negative numbers, or minus zero. If the "RealType" is constrained to "**base**" 2, the "realnumber" or "**-**" "realnumber" identifies the "**base**" 2 abstract value corresponding either to the decimal value specified by the "realnumber" or to a locally-defined precision if an exact representation is not possible.

Replace subclause 21.9 with the following:

21.9 The value of a bitstring type shall be defined by the notation "BitStringValue", or when used as an "XMLValue", by the notation "XMLBitStringValue". These productions are:

```
BitStringValue ::=
        bstring
    |   hstring
    |   "{" IdentifierList "}"
    |   "{" "}"
    |   CONTAINING Value
```

```
IdentifierList ::=
                identifier
     |          IdentifierList "," identifier

XMLBitStringValue ::=
           XMLTypedValue
     |     xmlbstring
     |     XMLIdentifierList
     |     empty

XMLIdentifierList ::=
           EmptyElementList
     |     TextList

EmptyElementList ::=
           "<" & identifier "/>"
     |     EmptyElementList "<" & identifier "/>"

TextList ::
           identifier
     |     TextList identifier
```

Add a new subclause 21.9bis as follows:

21.9bis If an "EmptyElementList" appears in an "XMLValueAssignment", then there shall be no occurrence of "TextList" in that "XMLValueAssignment".

Replace subclause 21.12 with the following:

21.12 Each "identifier" in "BitStringValue" or in the alternatives of "XMLBitStringValue" shall be the same as an "identifier" in the "BitStringType" production sequence with which the value is associated.

Add a new subclause 24.1bis as follows:

24.1bis For the purposes of the following clauses, a "PrefixedType" is defined to be a textually tagged type if either:

a) the "PrefixedType" is a "TaggedType"; or

b) the "Type" in the "PrefixedType" is a textually tagged type.

Replace the text of subclause 24.2 with the following (retaining the two NOTES):

24.2 When the "ComponentTypeLists" production occurs within the definition of a module for which automatic tagging is selected (see 12.3), and none of the occurrences of "NamedType" in any of the first three alternatives for "ComponentType" is a textually tagged type (see 24.1bis), then the automatic tagging transformation is selected for the entire "ComponentTypeLists", otherwise it is not.

*In the NOTE of subclause 24.3, replace "*explicitly*" with "*textually*".*

*In subclause 24.5.1, replace "*clause 30*" with "*30.2*".*

Replace subclause 24.8 with the following:

24.8 If automatic tagging is in effect and the "ComponentType"s in the extension root have no tags, then no "ComponentType" within the "ExtensionAdditionList" shall be a textually tagged type.

*In NOTE 1 of subclause 24.9, replace "*30.6*" with "*30.2.7*".*

*In subclause 25.3, remove the "XMLSpaceSeparatedList" alternative from the "XMLSequenceOfValue" production, remove the "XMLSpaceSeparatedList" production, remove "*1*" in NOTE 1, and delete NOTE 2.*

Add the following NOTE after subclause 25.4:

NOTE – This occurs only for `SEQUENCE OF NULL`.

Replace Table 5 with the following:

Table 5 – "XMLSequenceOfValue" and "XMLSetOfValue" notation for ASN.1 types

ASN.1 type	XML value notation
BitStringType	XMLDelimitedItemList
BooleanType	*See 25.5bis*
CharacterStringType	XMLDelimitedItemList
ChoiceType	XMLValueList
EmbeddedPDVType	XMLDelimitedItemList
EnumeratedType	*See 25.5ter*
ExternalType	XMLDelimitedItemList
InstanceOfType	*See ITU-T Rec. X.681 | ISO/IEC 8824-2, C.9*
IntegerType	XMLDelimitedItemList
NullType	XMLValueList
ObjectClassFieldType	*See ITU-T Rec. X.681 | ISO/IEC 8824-2, 14.10 and 14.11*
ObjectIdentifierType	XMLDelimitedItemList
OctetStringType	XMLDelimitedItemList
RealType	XMLDelimitedItemList
RelativeOIDType	XMLDelimitedItemList
SequenceType	XMLDelimitedItemList
SequenceOfType	XMLDelimitedItemList
SetType	XMLDelimitedItemList
SetOfType	XMLDelimitedItemList
PrefixedType	*See 25.11.1*
UsefulType (GeneralizedTime)	XMLDelimitedItemList
UsefulType (UTCTime)	XMLDelimitedItemList
UsefulType (ObjectDescriptor)	XMLDelimitedItemList
TypeFromObject	*See ITU-T Rec. X.681 | ISO/IEC 8824-2, 15.6*
ValueSetFromObjects	*See ITU-T Rec. X.681 | ISO/IEC 8824-2, 15.6*

Insert two new subclauses 25.5bis and 25.5ter as follows:

25.5bis If "EmptyElementBoolean" is used for the value of a boolean type then "XMLValueList" shall be used, otherwise "XMLDelimitedItemList" shall be used.

25.5ter If "EmptyElementEnumerated" is used for the value of an enumerated type then "XMLValueList" shall be used, otherwise "XMLDelimitedItemList" shall be used.

Delete the three subclauses 25.6, 25.7 and 25.8. Note that these clauses have been inserted with minor modifications as 25.11.1, 25.11.2 and 25.11.3 respectively.

Replace subclause 25.11 with the following:

25.11 If the first alternative of "XMLDelimitedItem" is used, then if the component of the sequence-of type (after ignoring any occurrences of "TypePrefix") is a "typereference" or an "ExternalTypeReference", then the

"NonParameterizedTypeName" shall be the "typereference" or the "typereference" in the "ExternalTypeReference", respectively, otherwise it shall be the "xmlasn1typename" specified in Table 4 corresponding to the built-in type of the component.

Insert three new subclauses 25.11.1, 25.11.2 and 25.11.3 as follows:

25.11.1 If the "Type" of the component is a "PrefixedType" then the type which determines the "XMLSequenceOfValue" alternative and the "xmlasn1typename" (if required) shall be the "Type" in the "PrefixedType" (see 30.1.5). If this is itself a "PrefixedType", a "ConstrainedType" or a "SelectionType" then these subclauses of 25.11 shall be recursively applied.

25.11.2 If the "Type" of the component is a "ConstrainedType" then the type which determines the "XMLSequenceOfValue" alternative and the "xmlasn1typename" (if required) shall be the "Type" in the "ConstrainedType" (see 45.1). If this is itself a "PrefixedType", a "ConstrainedType" or a "SelectionType" then these subclauses of 25.11 shall be recursively applied.

25.11.3 If the "Type" of the component is a "SelectionType" then the type which determines the "XMLSequenceOfValue" alternative and the "xmlasn1typename" (if required) notation shall be the type referenced by the "SelectionType" (see clause 29). If this is itself a "PrefixedType", a "ConstrainedType" or a "SelectionType" then these subclauses of 25.11 shall be recursively applied.

*In subclause 26.3, replace "*clause 30*" with "*30.2*".*

Replace subclause 27.3 with the following:

27.3 The notation for defining a value of a set-of type shall be the "SetOfValue", or when used as an "XMLValue", "XMLSetOfValue". These productions are:

```
SetOfValue ::=
        "{" ValueList "}"
    |   "{" NamedValueList "}"
    |   "{"  "}"

XMLSetOfValue ::=
        XMLValueList
    |   XMLDelimitedItemList
    |   empty
```

"ValueList", "NamedValueList" and the alternatives of "XMLSetOfValue" are specified in 25.3, and the choice of alternative is the same as if "XMLSequenceOfValue" had been used. The "**{**" "**}**" or "empty" notation is used when the "SetOfValue" or "XMLSetOfValue" is an empty list.

NOTE 1 – Semantic significance should not be placed on the order of these values.

NOTE 2 – Encoding rules are not required to preserve the order of these values.

NOTE 3 – The set-of type is not a mathematical set of values, thus, as an example, for **SET OF INTEGER** the values **{ 1 }** and **{ 1 1 }** are distinct.

Replace subclause 28.2 with the following:

28.2 When the "AlternativeTypeLists" production occurs within the definition of a module for which automatic tagging is selected (see 12.3), and none of the occurrences of "NamedType" in any "AlternativeTypeList" is a textually tagged type (see 24.1bis), the automatic tagging transformation is selected for the entire "AlternativeTypeLists", otherwise it is not.

*In subclause 28.3 replace "*clause 30*" with "*30.2*".*

*In subclause 28.4, replace "*tagged type*" with "*textually tagged type*".*

*In subclause 28.5, NOTE 1, replace "*30.6*" with "*30.2.7*".*

Replace clause 30 and its subclauses with the following:

30 Notation for prefixed types

30.1 General

30.1.1 A prefixed type is a new type which is isomorphic with an old type, but which has a different or additional tag and may have different or additional associated encoding instructions.

30.1.2 A prefixed type is either a "TaggedType" or an "EncodingPrefixedType".

30.1.3 A prefixed type which is a tagged type is mainly of use where this Recommendation | International Standard requires the use of types with distinct tags (see 24.5 to 24.6, 26.3 and 28.3). The use of a "TagDefault" of **`AUTOMATIC TAGS`** in a module allows this to be accomplished without the explicit appearance of "TaggedType" in that module.

NOTE – Where a protocol determines that values from several data types may be transmitted at any moment in time, distinct tags may be needed to enable the recipient to correctly decode the value.

30.1.4 The assignment of an encoding instruction using an "EncodingPrefixedType" is only relevant to the encodings identified by the associated encoding reference and has no effect on the abstract values of the type.

30.1.5 The notation for a prefixed type shall be "PrefixedType":

```
PrefixedType ::=
        TaggedType
    |   EncodingPrefixedType
```

NOTE – Specification of the syntax for "PrefixedType" would be simpler and clearer if tagging was described as the assignment of an encoding instruction. However, historically, tagging was introduced in the earliest versions of the ASN.1 specifications, and can affect the legality of a type definition. Minimum changes to the concepts of tagging (and the associated syntactic descriptions) were made when encoding prefixed types were introduced. Tagging also differs syntactically from assignment of encoding instructions: the specification that tagging is **`EXPLICIT`** or **`IMPLICIT`** occurs following the closing "`]`" of the tag, it is not contained within the paired "`[`" and "`]`" as is the case with normal encoding instructions.

30.1.6 The notation for a value of a "PrefixedType" shall be "PrefixedValue", or when used as an "XMLValue", "XMLPrefixedValue". These productions are:

```
PrefixedValue ::= Value

XMLPrefixedValue ::= XMLValue
```

where "Value" or "XMLValue" is a notation for a value of the "Type" in the "TaggedType" or the "EncodingPrefixedType" of the "PrefixedType".

NOTE 1 – Neither the "Tag" nor any part of the "EncodingPrefix" appears in this notation.

NOTE 2 – Encoding instructions can also be assigned to a type in an encoding control section (see clause 50). Such an assignment has no effect on the value notation for a type.

30.2 The tagged type

30.2.1 The notation for a tagged type shall be "TaggedType":

```
TaggedType ::=
            Tag Type
    |       Tag IMPLICIT Type
    |       Tag EXPLICIT Type

Tag ::= "[" EncodingReference Class ClassNumber "]"

EncodingReference ::=
            encodingreference ":"
    |       empty

ClassNumber ::=
            number
    |       DefinedValue

Class ::=
            UNIVERSAL
    |       APPLICATION
    |        PRIVATE
    |        empty
```

30.2.2 When used in "Tag", the "encodingreference" shall be **`TAG`**. The "EncodingReference" in "Tag" shall not be "empty" unless the default encoding reference for the module is **`TAG`** (see 12.4bis).

30.2.3 The "valuereference" in "DefinedValue" shall be of type integer, and assigned a non-negative value.

30.2.4 The new type is isomorphic with the old type, but has a tag with class "Class" and number "ClassNumber", except when "Class" is "empty", in which case the tag is context-specific class and number is "ClassNumber".

30.2.5 The "Class" shall not be **`UNIVERSAL`** except for types defined in this Recommendation | International Standard.

NOTE 1 – Use of universal class tags are agreed from time to time by ITU-T and ISO.

NOTE 2 – Subclause E.2.12 contains guidance and hints on stylistic use of tag classes.

30.2.6 All application of tags is either implicit tagging or explicit tagging. Implicit tagging indicates, for those encoding rules which provide the option, that explicit identification of the original tag of the "Type" in the "TaggedType" is not needed during transfer.

NOTE – It can be useful to retain the old tag where this was universal class, and hence unambiguously identifies the old type without knowledge of the ASN.1 definition of the new type. Minimum transfer octets is, however, normally achieved by the use of **`IMPLICIT`**. An example of an encoding using **`IMPLICIT`** is given in ITU-T Rec. X.690 | ISO/IEC 8825-1.

30.2.7 The tagging construction specifies explicit tagging if any of the following holds:

a) the "Tag **`EXPLICIT`** Type" alternative is used;

b) the "Tag Type" alternative is used and the value of "TagDefault" for the module is either **`EXPLICIT TAGS`** or is empty;

c) the "Tag Type" alternative is used and the value of "TagDefault" for the module is **`IMPLICIT TAGS`** or **`AUTOMATIC TAGS`**, but the type defined by "Type" is an untagged choice type, an untagged open type, or an untagged "DummyReference" (see ITU-T Rec. X.683 | ISO/IEC 8824-4, 8.3).

The tagging construction specifies implicit tagging otherwise.

30.2.8 If the "Class" is "empty", there are no restrictions on the use of "Tag", other than those implied by the requirement for distinct tags in 24.5 to 24.6, 26.3 and 28.3.

30.2.9 The **`IMPLICIT`** alternative shall not be used if the type defined by "Type" is an untagged choice type or an untagged open type or an untagged "DummyReference" (see ITU-T Rec. X.683 | ISO/IEC 8824-4, 8.3).

30.3 The encoding prefixed type

30.3.1 The notation for an encoding prefixed type shall be "EncodingPrefixedType":

```
EncodingPrefixedType ::=
        EncodingPrefix Type

EncodingPrefix ::=
        "[" EncodingReference EncodingInstruction "]"
```

"EncodingReference" is defined in 30.2.1.

30.3.2 The "EncodingInstruction" production is specified in the Recommendation | International Standard identified by the "EncodingReference" (see Annex Cbis) and can consist of any sequence of ASN.1 lexical items (including comment, cstring and white-space).

NOTE 1 – The "**`[`**" and "**`]`**" lexical items never appear in "EncodingInstruction".

NOTE 2 – Future versions of this Recommendation | International Standard may add further encoding references to Annex Cbis. It is recommended that ASN.1 tools provide (only) warnings if an "encodingreference" is not one of those specified in Annex Cbis and then ignore the whole "EncodingPrefix" using a "**`]`**" as the terminator (see NOTE 1 above).

30.3.3 If the "EncodingReference" is empty, then the encoding reference for the encoding prefix is the default encoding reference for the module.

NOTE – If the default encoding reference for the module is **`TAG`** (see 30.2.2) and the "EncodingReference" is "empty", then the "PrefixedType" is a "TaggedType", not an "EncodingPrefixedType".

30.3.4 There are in general restrictions on the encoding instructions (with the same encoding reference) that can be used in combination, and on the types to which particular instructions or combinations of instructions can be applied. These restrictions are specified in the Recommendation | International Standard associated with the encoding reference (see Annex Cbis), and are not specified in this Recommendation | International Standard.

Replace clause 38 with the following:

38 Naming characters, collections and property category sets

This clause specifies an ASN.1 built-in module which contains the definition of a value reference name for each character from ISO/IEC 10646-1, where each name references a **`UniversalString`** value of size 1. This module also contains the definition of a type reference name for each collection of characters from ISO/IEC 10646-1, where each name references a subset of the **`UniversalString`** type. Finally, it contains the definition of a "typereference" name for the set of characters in each general category of character properties that are listed in 4.5 of The Unicode Standard, where each name references a subset of the **`UniversalString`** type.

NOTE – These values are available for use in the value notation of the **`UniversalString`** type and types derived from it. All of the value and type references defined in the module specified in 38.1 are exported and must be imported by any module that uses them.

Insert two new subclauses 38.1.4bis and 38.1.4ter as follows:

38.1.4bis For each abbreviation and each description listed in The Unicode Standard, Table 4-5, two statements are included in the module of the form:

```
<categoryabbreviation> ::= UniversalString (FROM (<alternativelist>))
     -- represents the set of characters with the property
     -- category <categoryabbreviation>.

<categorydescription> UniversalString ::= <categoryabbreviation>
```

where:

a) <categoryabbreviation> is the abbreviation for the general category of character properties listed in The Unicode Standard, Table 4-5 (for example, **`Lu`** or **`Nd`** or **`Pi`**);

b) <categorydescription> is the description for the same general category of characters, with the initial letter of all words uppercased, the comma and all spaces removed, and all description in parentheses removed (for example, **`LetterUppercase`** or **`NumericDigit`** or **`PunctuationInitialQuote`**);

c) The <alternativelist> for each <categoryabbreviation> is a list of the <namedcharacter> names produced by 38.2 for each of the characters listed in The Unicode Character Database (version 3.2.0) of The Unicode Standard that have the corresponding <categoryabbreviation>.

NOTE – The Unicode name for a character is the same as the <iso10646name> for that character.

38.1.4ter For the initial letter of each abbreviation listed in The Unicode Standard, Table 4-5, two statements are included in the module of the form:

```
<categoryabbreviationletter> ::= UniversalString (FROM (<alternativelist>))
     -- represents the set of characters with any category property
     -- with the initial letter <categoryabbreviationletter>.

<maincategorydescription> UniversalString ::= <categoryabbreviationletter>
```

where:

a) <categoryabbreviationletter> is the first letter of the abbreviation for the general category of character properties listed in The Unicode Standard, Table 4-5 (for example, **`L`** or **`N`** or **`P`**);

b) <categorydescription> is the first word of the description for the same general category of characters (for example, **`Letter`** or **`Numeric`** or **`Punctuation`**);

d) The <alternativelist> for each <categoryabbreviationletter> is a list of the <namedcharacter> names produced by 38.2 for each of the characters listed in The Unicode Character Database (version 3.2.0) of The Unicode Standard that have the corresponding <categoryabbreviationletter>.

NOTE – The Unicode name for a character is the same as the <iso10646name> for that character.

*In subclause 45.4 replace the two occurrences of "*TaggedType*" with "*PrefixedType*".*

Replace the NOTE in subclause 47.4.2 with the following:

NOTE – For the purpose of subtyping, **`NOT-A-NUMBER`** exceeds all real values, **`PLUS-INFINITY`** exceeds all real values except **`NOT-A-NUMBER`**, minus zero exceeds all negative real values and is less than plus zero, and **`MINUS-INFINITY`** is less than all real values. Otherwise, normal mathematical ordering is applied.

*In subclause 48.7.1 a), replace "*heading*" with "*header*".*

Insert a new clause 50 as follows:

50 Encoding control sections

50.1 The "EncodingControlSections" is specified by the following productions:

EncodingControlSections ::=
EncodingControlSection EncodingControlSections |
empty

EncodingControlSection ::=
`ENCODING-CONTROL`
encodingreference
EncodingInstructionAssignmentList

50.2 Each "EncodingControlSection" within an ASN.1 module shall have a different "encodingreference", and assigns encoding instructions for that encoding reference to one or more types in the module.

50.3 The "encodingreference" shall not be **`TAG`**.

50.4 The "EncodingInstructionAssignmentList" production and the associated semantics is specified in the Recommendation | International Standard identified by the "encodingreference" (see Annex Cbis) and can consist of any sequence of ASN.1 lexical items (including comment, cstring and white-space) except the lexical items **`END`** and **`ENCODING-CONTROL`**, which will not appear in an "EncodingInstructionAssignmentList".

NOTE 1 – Future versions of this Recommendation | International Standard may add further encoding references to Annex Cbis. It is recommended that ASN.1 tools provide (only) warnings if the "encodingreference" in an "EncodingControlSection" is not one of those specified in Annex Cbis and then ignore the "EncodingControlSection" until the next occurrence of **`END`** or **`ENCODING-CONTROL`**, whichever comes first.

NOTE 2 – The "encodingreference" in an "EncodingControlSection" cannot be omitted. The default encoding reference for the module has no effect on an "EncodingControlSection".

50.5 There are interactions and restrictions on the assignment of encoding instructions (with the same encoding reference) to a type using a type prefix and using an "EncodingControlSection". It is always possible (as a matter of style) to use only "EncodingControlSection"s, but there are in general some encoding instructions (particularly those that apply to all types in a module) that can only be assigned in an "EncodingControlSection". There are also restrictions on the types to which particular instructions or combinations of instructions can be applied. These interactions and restrictions are specified in the Recommendation | International Standard associated with the encoding reference (see Annex Cbis), and are not specified in this Recommendation | International Standard.

In subclause A.2.4, add the following to the end of the clause:

The regular expressions **`"\N{LetterUppercase}"`** and **`"\N{Lu}"`** match any (single) character of the general category "Letter, uppercase" (abbreviated as "Lu") as defined by The Unicode Standard.

*In subclause B.3.2.2, replace each of the two occurrences of "*clause 30*" with "*30.2*".*

*In subclause B.4.2, replace "*clause 30*" with "*30.2*".*

Insert a new annex Cbis as follows:

Annex Cbis

Encoding references

(This annex forms an integral part of this Recommendation | International Standard)

Cbis.1 This annex specifies the currently defined encoding references and the Recommendation | International Standard that specifies the syntactic form (and semantics) of encoding instructions with that encoding reference (except for the **`TAG`** encoding reference, which has no associated encoding instructions).

NOTE – It is recommended that, if an encoding reference that is not specified here appears in an ASN.1 specification, the associated encoding instructions be ignored with (only) a warning diagnostic.

Cbis.2 The encoding references in column 1 of Table Cbis.1 are currently defined. The syntax and semantics of the associated encoding instructions (where applicable) are defined in the Recommendation | International Standard referenced in column 2 of Table Cbis.1.

Encoding reference	Refer to standard
TAG	This Recommendation | International Standard
XER	ITU-T Rec. X.693/Amd. 1 | ISO/IEC 8825-4/Amd. 1: EXTENDED-XER

Table Cbis.1 – Standards defining the semantics associated with a given encoding reference

Replace subclause E.2.12.2 with the following:

E.2.12.2 A frequently encountered style for the use of tags is to assign an application class tag precisely once in the entire specification, using it to identify a type that finds wide, scattered, use within the specification. An application class tag is also frequently used (once only) to tag the types in the outermost **`CHOICE`** of an application, providing identification of individual messages by the application class tag. The following is an example use in the former case:

EXAMPLE

```
FileName ::= [APPLICATION 8] SEQUENCE {
           directoryName               VisibleString,
           directoryRelativeFileName   VisibleString}
```

The above example assumes that the default encoding reference is either "empty" or **`TAG`**. Otherwise, the above example would be written:

```
FileName ::= [TAG: APPLICATION 8] SEQUENCE {
           directoryName               VisibleString,
           directoryRelativeFileName   VisibleString}
```

A similar change would be needed in subsequent examples.

In Annex H do the following:

Insert the following items into the list of lexical items:

encodingreference

extended-true

extended-false

"NaN"

"INF"

`ENCODING-CONTROL`

`INSTRUCTIONS`

```
NOT-A-NUMBER
```

Replace the production "XMLBooleanValue" with the following three productions:

```
XMLBooleanValue ::=
                    EmptyElementBoolean
        |           TextBoolean

    EmptyElementBoolean ::=
                    "<" & "true" "/>"
        |           "<" & "false" "/>"

    TextBoolean ::=
                    extended-true
        |           extended-false
```

Replace the production "XMLIntegerValue" with the following four productions:

```
    XMLIntegerValue ::=
            XMLSignedNumber
        |   EmptyElementInteger
        |   TextInteger

    XMLSignedNumber ::=
            number
        |   "-" & number

    EmptyElementInteger ::=
            "<" & identifier "/>"

    TextInteger ::=
            identifier
```

Replace the production "XMLEnumeratedValue" with the following three productions:

```
XMLEnumeratedValue ::=
            EmptyElementEnumerated
        |   TextEnumerated

    EmptyElementEnumerated ::= "<" & identifier "/>"

    TextEnumerated ::= identifier
```

Replace the productions "XMLNumericRealValue" and "XMLSpecialRealValue" with the following 4 productions:

```
    XMLNumericRealValue ::=
            realnumber
        |   "-" & realnumber

    XMLSpecialRealValue ::=
            EmptyElementReal
        |   TextReal

    EmptyElementReal ::=
            "<" & PLUS-INFINITY "/>"
        |   "<" & MINUS-INFINITY "/>"
        |   "<" & NOT-A-NUMBER "/>"

    TextReal ::=
            "INF"
        |   "-" & "INF"
        |   "NaN"
```

Replace the production "XMLIdentifierList" with the following three productions:

```
XMLIdentifierList ::=
            EmptyElementList
```

```
        |    TextList

EmptyElementList ::=
             "<" & identifier "/>"
        |    EmptyElementList "<" & identifier "/>"

TextList ::
             identifier
        |    TextList identifier
```

In the production "XMLSequenceOfValue", delete the alternative "XMLSpaceSeparatedList".

Delete the production "XMLSpaceSeparatedList".

In the production "XMLSetOfValue", delete the alternative "XMLSpaceSeparatedList".

Insert the following productions after the production "SelectionType":

```
PrefixedType ::=
             TaggedType
        |    EncodingPrefixedType

EncodingPrefixedType ::=
        EncodingPrefix Type

EncodingPrefix ::=
        "[" EncodingReference EncodingInstruction "]"

PrefixedValue ::= Value

XMLPrefixedValue ::= XMLValue
```

Replace the "Tag" production with the following two productions:

```
Tag ::= "[" EncodingReference Class ClassNumber "]"

EncodingReference ::=
                    encodingreference  ":"
        |           empty
```

Delete the productions "TaggedValue" and "XMLTaggedValue".