INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

Amendment 1
X.680
(04/95)

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: RULES OF EXTENSIBILITY

Amendment 1 to
ITU-T Recommendation X.680
Superseded by a more recent version

(Previously "CCITT Recommendations")

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. Some 179 member countries, 84 telecom operating entities, 145 scientific and industrial organizations and 38 international organizations participate in ITU-T which is the body which sets world telecommunications standards (Recommendations).

The approval of Recommendations by the Members of ITU-T is covered by the procedure laid down in WTSC Resolution No. 1 (Helsinki, 1993). In addition, the World Telecommunication Standardization Conference (WTSC), which meets every four years, approves Recommendations submitted to it and establishes the study programme for the following period.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC. The text of ITU-T Recommendation X.680, Amendment 1, was approved on 10th of April 1995. The identical text is also published as ISO/IEC International Standard 8824-1.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized private operating agency.

Superseded by a more recent version

ITU-T X-SERIES RECOMMENDATIONS

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

(February 1994)

ORGANIZATION OF X-SERIES RECOMMENDATIONS

Subject area	Recommendation Series
PUBLIC DATA NETWORKS	
Services and Facilities	X.1-X.19
Interfaces	X.20-X.49
Transmission, Signalling and Switching	X.50-X.89
Network Aspects	X.90-X.149
Maintenance	X.150-X.179
Administrative Arrangements	X.180-X.199
OPEN SYSTEMS INTERCONNECTION	
Model and Notation	X.200-X.209
Service Definitions	X.210-X.219
Connection-mode Protocol Specifications	X.220-X.229
Connectionless-mode Protocol Specifications	X.230-X.239
PICS Proformas	X.240-X.259
Protocol Identification	X.260-X.269
Security Protocols	X.270-X.279
Layer Managed Objects	X.280-X.289
Conformance Testing	X.290-X.299
INTERWORKING BETWEEN NETWORKS	
General	X.300-X.349
Mobile Data Transmission Systems	X.350-X.369
Management	X.370-X.399
MESSAGE HANDLING SYSTEMS	X.400-X.499
DIRECTORY	X.500-X.599
OSI NETWORKING AND SYSTEM ASPECTS	
Networking	X.600-X.649
Naming, Addressing and Registration	X.650-X.679
Abstract Syntax Notation One (ASN.1)	X.680-X.699
OSI MANAGEMENT	X.700-X.799
SECURITY	X.800-X.849
OSI APPLICATIONS	
Commitment, Concurrency and Recovery	X.850-X.859
Transaction Processing	X.860-X.879
Remote Operations	X.880-X.899
OPEN DISTRIBUTED PROCESSING	X.900-X.999

CONTENTS

Summary

The ASN.1 rules of extensibility describe how to write an ASN.1 module so as to permit a phased migration to a new version of an ASN.1 specification.

Superseded by a more recent version

Introduction

This Recommendation | International Standard documents the changes to ITU-T Rec. X.680 | ISO/IEC 8824-1 needed to support the ASN.1 Rules of Extensibility.

The ASN.1 Rules of Extensibility describes how to write an ASN.1 module in such a way as to allow a phased migration to a new version of an ASN.1 specification. The new version may differ from the previous version by new components being added to a SET, SEQUENCE or CHOICE, new enumerations being added to an enumerated type, and constraints on a subtype specification being relaxed. A phased migration to the new version of the ASN.1 specification allows communicating peers throughout a network to simultaneously have differing knowledge of the set of values allowed in the abstract syntax, yet be able to communicate without the knowledge by any peer that its peer has a different understanding of what the set of values in the abstract syntax is.

For example, initially peers A, B, C and D may have identical views of the types of values that can be exchanged. Assuming that the ASN.1 specification that describes these values was originally defined with extensibility in mind, it can be extended by adding new components to a SEQUENCE, for example, thereby creating a new version of the ASN.1 specification. Peers A and B may immediately adopt the new version, while peers C and D continue using the old version, yet all four peers can continue communicating with one another without difficulty because the SEQUENCE was originally defined as being extensible. When peers A and B exchange a value of the newly extended SEQUENCE they will both see values for the new components that were added to the SEQUENCE (provided they are sent). When peer A (or B) sends a message containing the newly added components to peer C (or D), the message will appear to C at the abstract level (i.e. after the message has been decoded) without any of the newly added components; only the original components in the SEQUENCE will be seen. When peer C (or D) sends a message to peer A (or B), the message will (of course) not contain any of the new components, however, it will appear to A as though a peer using the same version of the ASN.1 specification as A (e.g. B) had sent the message with all of the new components missing. Peers C and D continue to exchange messages with each other the way they always did.

INTERNATIONAL STANDARD

ITU-T RECOMMENDATION

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

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

Rules of extensibility

1 Scope

This Recommendation | International Standard documents the changes to ITU-T Rec. X.680 | ISO/IEC 8824-1 needed to support the ASN.1 Rules of Extensibility.

2 Normative references

The following Recommendations and International Standards contain provisions which, through reference in this text, constitute provisions of this Recommendation | International Standard. At the time of publication, the editions indicated were valid. All Recommendations and Standards are subject to revision, and parties to agreements based on this Recommendation | International Standard are encouraged to investigate the possibility of applying the most recent editions of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunications Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.680 (1994) | ISO/IEC 8824-1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.682 (1994) | ISO/IEC 8824-3:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- ITU-T Recommendation X.683 (1994) | ISO/IEC 8824-4:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.*
- ITU-T Recommendation X.690 (1994) | ISO/IEC 8825-1:1995, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*
- ITU-T Recommendation X.691 (1995) | ISO/IEC 8825-2:1995, *Information technology – ASN.1 encoding rules – Specification of Packed Encoding Rules (PER).*

3 Changes to Introduction

{Add the following text to the Introduction of ITU-T Rec. X.680 | ISO/IEC 8824-1 immediately before the existing paragraph which begins "Clauses 8 to 31":}

An ASN.1 specification will initially be produced with a set of fully defined ASN.1 types. At a later stage, however, it may be necessary to change those types (usually by the addition of extra components in a sequence or set type). If this is to be possible in such a way that implementations using the old type definitions can interwork with implementations using the new type definitions in a defined way, encoding rules need to provide appropriate support. The ASN.1 notation supports the inclusion of an **extension marker** on a number of types. This signals to encoding rules the intention of the designer that this type is one of a series of related types (i.e. versions of the same initial type) called an **extension series**, and that the encoding rules are required to enable information transfer between implementations using different types that are related by being part of the same extension series.

4 Changes to Definitions

{Add the following new definitions to ITU-T Rec. X.680 | ISO/IEC 8824-1, maintaining the alphabetical order of definitions in ITU-T Rec. X.680 | ISO/IEC 8824-1. Note that the alphabetic character appearing in the clause numbers below will be changed to an appropriate numeric character when the definitions are added to the base document:}

3.8.a extension series: A series of ASN.1 types which can be ordered in such a way that each successive type in the series is formed by the addition of text to the end of the type notation of the immediately preceding type in the series.

NOTE – Both nested and unnested types can be extended.

3.8.b extension marker: A syntactic flag (an ellipsis) that is included in all types that form part of an extension series.

3.8.c extension root: An extensible type that is the first type in an extension series. It carries the extension marker with no additional notation other than comments and whitespace between the extension marker and the matching "}" or ")".

NOTE – Only an extension root can be the first type in an extension series.

3.8.d extension addition: One of the added notations in an extension series. For set types and sequence types, each extension addition is the addition of a single element. For enumerated types it is the addition of a single further enumeration. For choice types it is the addition of a single further choice. For a constraint it is the addition of a subtype element.

NOTE – Extension additions are both textually ordered (following the extension marker) and logically ordered (having increasing tags or enumeration values).

3.8.e extension-related: Two types that have the same extension root, and where one was created by adding zero or more extension additions to the other.

3.8.f extensible type: A type with an extension marker.

3.8.g extensible constraint: A subtype constraint with an extension marker.

5 The ASN.1 model of type extension

{Add this text as a new clause before clause 6 of ITU-T Rec. X.680 | ISO/IEC 8824-1, using the same clause heading as this clause.}

When decoding an extensible type, a decoder may detect:

a) the absence of expected extension additions in a sequence or set type; or

b) the presence of arbitrary unexpected extension additions above those defined (if any) in a sequence or set type, or of an unknown alternative in a choice type, or an unknown enumeration in an enumerated type, or of an unexpected length or value of a type whose constraint is extensible.

In formal terms, an abstract syntax defined by the extensible type "X" contains not only the values of type "X", but also the values of all types that are extension-related to "X". Thus, the decoding process never signals an error when either of the above situations (a or b) is detected. The action that is taken in each situation is a matter for the application layer designer to specify.

NOTE – Frequently the action will be to ignore the presence of unexpected additional extensions, and to use a default value or a "missing" indicator for expected extension additions that are absent.

Unexpected extension additions detected by a decoder in an extensible type can later be included in a subsequent encoding of that type (for transmission back to the sender, or to some third party), provided that the same transfer syntax is used on the subsequent transmission.

6 Extensibility requirements on encoding rules

{Add this text as a new clause to ITU-T Rec. X.680 | ISO/IEC 8824-1 before clause 6 and after the newly added clause above, using the same clause heading as this clause:}

6.1 All ASN.1 encoding rules shall allow the encoding of values of an extensible type "X" in such a way that they can be decoded using an extensible type "Y" that is extension-related to "X". Further, the encoding rules shall allow the values that were decoded using "Y" to be re-encoded (using "Y") and decoded using a third extensible type "Z" that is extension related to "Y" (and hence "X" also).

NOTE – Types "X", "Y" and "Z" may appear in any order in the extension series.

If a value of an extensible type "X" is encoded and then relayed (directly or through a relaying application using extension-related type "Z") to another application that decodes the value using extensible type "Y" that is extension-related to "X", then the decoder using type "Y" obtains an abstract value composed of:

a) n abstract value of the extension root type;

b) n abstract value of each extension addition that is present in both "X" and "Y";

c) delimited encoding for each extension addition (if any) that is in "X" but not in "Y".

The encodings in c) shall be capable of being included in a later encoding of a value of "Y", if so required by the application. That encoding shall be a valid encoding of a value of "X".

Tutorial example: If system A is using an extensible root type (type "X") that is a sequence type or a set type with an extension addition of an optional integer type, while system B is using an extension-related type (type "Y") that has two extension additions where each is an optional integer type, then transmission by B of a value of "Y" which omits the integer value of the first extension addition and includes the second must not be confused by A with the presence of the first (only) extension addition of "X" that it knows about. Moreover, A must be able to re-encode the value of "X" with a value present for the first integer type, followed by the second integer value received from B, if so required by the application protocol.

6.2 All ASN.1 encoding rules shall specify the encoding and decoding of the value of an enumerated type and a choice type in such a way that if a transmitted value is in the set of extension additions held in common by the encoder and the decoder, then it is successfully decoded, otherwise it shall be possible for the decoder to delimit the encoding of it and to identify it as a value of an (unknown) extension addition.

6.3 All ASN.1 encoding rules shall specify the encoding and decoding of types with extensible constraints in such a way that if a transmitted value is in the set of extension additions held in common by the encoder and the decoder, then it is successfully decoded, otherwise it shall be possible for the decoder to delimit the encoding of and to identify it as a value of an (unknown) extension addition.

In all cases, the presence of extension additions shall not affect the ability to recognize later material when a type with an extension marker is nested inside some other type.

NOTE – All variants of the Basic Encoding Rules of ASN.1 and the Packed Encoding Rules of ASN.1 satisfy all these requirements.

7 Changes to Module Definition

{Change the productions in 10.1 of ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

ModuleDefinition ::=
ModuleIdentifier
DEFINITIONS
TagDefault
ExtensionDefault
"::="
BEGIN
ModuleBody
END

ExtensionDefault ::=
EXTENSIBILITY IMPLIED | empty

{All other productions in 10.1 remain the same}

{Insert a new clause after 10.3 in ITU-T Rec. X.680 | ISO/IEC 8824-1:}

10.3 *bis* The "EXTENSIBILITY IMPLIED" option is equivalent to the textual insertion of an extension marker (...) in the definition of each type in the module for which it is permitted. The absence of "EXTENSIBILITY IMPLIED" means that extensibility is only provided for those types within the module where an extension marker is explicitly present.

NOTE – "EXTENSIBILITY IMPLIED" affects only types. It has no effect on object sets.

8 Change to production for ValueSet

{Replace the production for ValueSet in 13.5 of ITU-T Rec. X.680 | ISO/IEC 8824-1 with:}

ValueSet ::= "{" ElementSetSpecs "}"

9 Change to definition of types and values

{Insert a (new) 14.12 in ITU-T Rec. X.680 | ISO/IEC 8824-1:}

14.12 The implied or explicit presence of an extension marker in the definition of a type has no effect on the value notation. That is, the value notation for a type with an extension marker is exactly the same as if the extension marker was absent.

10 Changes to ENUMERATED

{Change the production in clause 17 in ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

```
EnumeratedType ::= ENUMERATED "{" Enumerations "}"
Enumerations ::= RootEnumeration |
                 RootEnumeration "," "..." |
                 RootEnumeration "," "..." "," AdditionalEnumeration

RootEnumeration ::= Enumeration

AdditionalEnumeration ::= Enumeration
```

{"Enumeration" is defined as it is now.}

{Change Note 2 of 17.1 to read:}

2 The numeric values inside the "NamedNumber"s in the "RootEnumeration" are not necessarily ordered or contiguous, and the numeric values inside the "NamedNumber"s in the "AdditionalEnumeration" are not necessarily contiguous.

{Add the following text after 17.3:}

17.3 *bis* The value of each new "AdditionalEnumeration" shall be greater than all previously defined "AdditionalEnumeration"s in the type.

17.3 *ter* When a "NamedNumber" is used in defining an "AdditionalEnumeration" the value associated with it shall be different from the value of all previously defined "EnumerationItem"s (in this type) regardless of whether the previously defined "EnumerationItem"s occur in the enumeration root or not. For example:

```
A ::= ENUMERATED {a, b, ..., c(0)}          -- invalid, since both 'a' and 'c' equal 0
B ::= ENUMERATED {a, b, ..., c, d(2)}       -- invalid, since both 'c' and 'd' equal 2
C ::= ENUMERATED {a, b(3), ..., c(1)}       -- valid, 'c' = 1
D ::= ENUMERATED {a, b, ..., c(2)}          -- valid, 'c' = 2
```

17.3 *quater* The value associated with the first "AdditionalEnumeration" alternative that is an "identifier" (not a "NamedNumber") shall be the smallest value for which (a) an "EnumerationItem" is not defined in the "RootEnumeration" and (b) all preceding "EnumerationItem"s in the "AdditionalEnumeration" (if any) are smaller. For example, the following are all valid:

```
A ::= ENUMERATED {a, b, ..., c}             -- c = 2
B ::= ENUMERATED {a, b, c(0), ..., d}       -- d = 3
C ::= ENUMERATED {a, b, ..., c(3), d}       -- d = 4
D ::= ENUMERATED {a, z(25), ..., d}         -- d = 1
```

11 Changes to SEQUENCE

{Change the production in clause 22 in ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

```
SequenceType ::= SEQUENCE "{" "}" |
                 SEQUENCE "{" ExtensionAndException "}" |
                 SEQUENCE "{" ComponentTypeLists "}"

ExtensionAndException ::= "..." | "..." ExceptionSpec
```

```
ComponentTypeLists ::= RootComponentTypeList |
                       RootComponentTypeList "," ExtensionAndException |
                       RootComponentTypeList "," ExtensionAndException "," AdditionalComponentTypeList |
                       ExtensionAndException "," AdditionalComponentTypeList

RootComponentTypeList ::= ComponentTypeList

AdditionalComponentTypeList ::= ComponentTypeList
```

{"ComponentTypeList" is defined as now.}

NOTE – "ComponentType"s that are not marked OPTIONAL or DEFAULT in the "AdditionalComponentTypeList" should always be encoded, except when the presentation data value is being relayed from a sender that is using an earlier version of the abstract syntax in which the "ComponentType" is not defined.

*{Replace the **first sentence** of 22.4 with the following:}*

"Type" in the "COMPONENTS OF Type" notation shall be a sequence type that itself contains no extension marker, but it may include components that do.

{Create a new clause after 22.4 with the following:}

22.4 *bis* The "COMPONENTS OF Type" notation shall not be used in "AdditionalComponentTypeList".

12 Changes to SET

{Change the production in clause 24 in ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

```
SetType ::= SET "{" "}" |
            SET "{" ExtensionAndException "}" |
            SET "{" ComponentTypeLists "}"
```

"ComponentTypeLists" is specified in 22.1.

*{Replace the **first sentence** of 24.2 with the following:}*

If a "Type" contains an extension marker and that "Type" is used as the "Type" in a "COMPONENTS OF Type", the extension marker is not visible to the "COMPONENTS OF Type". Thus, for the purposes of the "COMPONENTS OF Type" the "Type" is treated as though it was not defined with an extension marker.

{Add the following text after 24.3:}

24.3 *bis* The tag of each new "ComponentType" added to the "AdditionalComponentTypeList" shall be canonically greater (see 6.4) than those of the other components in the "AdditionalComponentTypeList", and shall occur last in the set type.

{Modify clause 24.4 to say:}

24.4 Subclauses 22.2, 22.2 *bis* and 22.6-22.10 also apply to set types.

13 Changes to CHOICE

{Change the production in clause 26 in ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

```
ChoiceType ::= CHOICE "{" AlternativeTypeLists "}"
AlternativeTypeLists ::=
    RootAlternativeTypeList |
    RootAlternativeTypeList "," ExtensionAndException |
    RootAlternativeTypeList "," ExtensionAndException "," AdditionalAlternativeTypeList

RootAlternativeTypeList ::= AlternativeTypeList

AdditionalAlternativeTypeList ::= AlternativeTypeList
```

{"AlternativeTypeList" is defined as now.}

{Add the following text after 26.3:}

26.3 *bis* The tag of each new "NamedType" added to the "AdditionalAlternativeTypeList" shall be canonically greater (see 6.4) than those of the other components of the "AdditionalAlternativeTypeList", and shall be the last "NamedType" in the choice type.

*{Modify 26.5, changing "*Where this type is used *..." to:}*

When this type does not have an extension marker and is used ...

{Modify 26.5, adding new text at the end of the first sentence to say:}

When this type has an extension marker, it shall not be used where this International Standard requires distinct tags (see 22.5, 24.3, 26.2).

14 Changes to Constrained Types

{Replace the last two sentences of 42.5 of ITU-T Rec. X.680 | ISO/IEC 8824-1 with:}

"ExceptionSpec" is defined in clause 43. Unless it is used in conjunction with an "extension marker" (see clause 44 *bis*), it shall only be present if the "ConstraintSpec" includes an occurrence of "DummyReference" (see 8.4 of ITU-T Rec. X.683 | ISO/IEC 8824-4) or is a "UserDefinedConstraint" (see ITU-T Rec. X.682 | ISO/IEC 8824-3, clause 9).

{Change the production in 42.6 in ITU-T Rec. X.680 | ISO/IEC 8824-1 to read:}

SubtypeConstraint ::= ElementSetSpecs

15 Changes to exception identifier

{Add two new subclauses 43.5 and 43.6 to ITU-T Rec. X.680 | ISO/IEC 8824-1:}

43.5 Where a type is constrained by multiple constraints, more than one of which has an exception identifier, then the exception identifier in the outermost constraint shall be regarded as the exception identifier for that type.

43.6 Where an exception marker is present on types that are used in set arithmetic, the exception identifier is ignored and is not inherited by the type being constrained as a result of the set arithmetic.

16 Changes to element set specification

{Add immediately in front of the production "ElementSetSpec" in 44.1 in ITU-T Rec. X.680 | ISO/IEC 8824-1:}

ElementSetSpecs ::=
RootElementSetSpec |
RootElementSetSpec "," "..." |
"..." "," AdditionalElementSetSpec |
RootElementSetSpec "," "..." "," AdditionalElementSetSpec

RootElementSetSpec ::= ElementSetSpec

AdditionalElementSetSpec ::= ElementSetSpec

NOTE – The elements that are referenced by "ElementSetSpecs" is the union of the elements referenced by the "RootElementSetSpec" and "AdditionalElementSetSpec".

{Add a new clause 44 bis and the following subclauses in ITU-T Rec. X.680 | ISO/IEC 8824-1:}

44 *bis* The extension marker

NOTE – Like the constraint notation in general, the extension marker has no effect on some encoding rules of ASN.1, such as the Basic Encoding Rules, but does on others, such as the Packed Encoding Rules.

44.1 The extension marker, ellipsis, is an indication that extension additions are expected. It makes no statement as to how such additions should be handled other than that they shall not be treated as an error during the decoding process.

44.2 The joint use of the extension marker and an exception identifier is an indication that extension additions are expected and thus should not be treated as an error in the decoding process, and that the application standards prescribes specific action to be taken by the application if there is a constraint violation. It is recommended that this notation be used in those situations where store and forward or any other form of relaying is in use, so as to indicate that any unrecognized extension additions are to be returned to the application for possible re-encoding and relaying.

44.3 Set arithmetic, if any, in the "ElementSetSpecs" notation shall be performed without consideration given to the presence of the extension marker.

NOTE – In other words, the presence of an extension marker has no effect on set arithmetic.

44.4 If a type defined with an extensible constraint is referenced in a "ContainedSubtype", the newly defined type does not inherit the extension marker. If the newly defined type is meant to be extensible then an extension marker shall be explicitly added to its "ElementSetSpecs". For example:

```
A ::= INTEGER (0..10, ...)          -- A is extensible
B ::= INTEGER (A)                   -- B is inextensible
C ::= INTEGER (A, ...)              -- C is extensible
```

44.5 If a type defined with an extensible constraint is further constrained with an "ElementSetSpecs" that does not contain an extension marker, the resulting type is one whose constraint is not extensible. For example:

```
A ::= INTEGER (0..10, ...)          -- A is extensible
B ::= A (2..5)                      -- B is inextensible
C ::= A                             -- C is extensible
```

44.6 Components of a set, sequence or choice type that are constrained to be absent shall not be present, regardless of whether the set, sequence or choice type is an extensible type.

NOTE – Inner type constraints have no effect on extensibility.

For example:

```
A ::= SEQUENCE   {
   a    INTEGER
   b    BOOLEAN OPTIONAL,
   ...
}
B ::= A (WITH COMPONENTS {b ABSENT})    -- B is extensible, but 'b' shall not be
                                        -- present in any of its values
```

{Add the following tutorial annex to ITU-T Rec. X.680 | ISO/IEC 8824-1:}

Annex A

Tutorial annex on the ASN.1 model of type extension

(This annex does not form an integral part of this Recommendation | International Standard)

A.1 Overview

A.1.1 It can happen that an ASN.1 type evolves over time from an **extension root** type by means of a series of extensions called **extension additions**.

A.1.2 An ASN.1 type available to a particular implementation may be the extension root type, or may be the extension root type plus one or more extension additions. Each such ASN.1 type that contains an extension addition also contains all previously defined extension additions.

A.1.3 The ASN.1 type definitions in this series are said to be **extension-related** (see clause 3.8.e for a more precise definition of "extension-related"), and encoding rules are required to encode extension-related types in a such a way that if two systems are using two different types which are extension-related, transmissions between the two systems will successfully transfer the information content of those parts of the extension-related types that are common to the two systems. It is also required that those parts that are not common to both systems can be delimited and retransmitted (perhaps to a third party) on a subsequent transmission, provided the same transfer syntax is used.

NOTE – The sender may be using a type that is either earlier or later in the series of extension additions.

A.1.4 The series of types obtained by progressively adding to a root type is called an **extension series**. In order for encoding rules to make appropriate provision for transmissions of extension-related types (which may require more bits on the line), such types (including the extension root type) need to be syntactically flagged. The flag is an ellipsis (...), and is called an **extension marker**.

EXAMPLE:

```
Root type                1st extension            2nd extension

SEQUENCE {               SEQUENCE {               SEQUENCE {
    a  INTEGER,              a  INTEGER,              a  INTEGER,
    ...                      ...,                     ...,
}                            b  BOOLEAN,              b  BOOLEAN,
                             c  INTEGER               c  INTEGER,
                         }                            d  REAL
                                                  }
```

A.1.5 The specification of the extension root type appears before/above the ASN.1 extension marker, and the extension additions appear after/below it.

A.1.6 A type that has an extension marker can be nested inside a type that has none, or it can be nested within a type in an extension root, or it can be nested in a type that is an extension addition. In such cases the extension series are treated independently, and the nested type with the extension marker has no impact on the type within which it is nested. Only one ellipsis (i.e. extension marker) can appear in any specific construct.

A.1.7 A new type in the extension series is defined in terms of a single extension addition to the previous type in the extension series. This does not prevent multiple extension additions from being made in successive publications of some particular Recommendation | International Standard.

A.1.8 While the normal practice will be for extension additions to be added over time, the underlying ASN.1 model and specification does not involve time. Two types are extension-related if one can be "grown" from the other by extension additions. That is, one contains all the components of the other. There may be types that have to be "grown" in the opposite direction (although this is unlikely). It could even be that, over time, a type *starts* with a lot of extension additions which were progressively removed! All that ASN.1 and its encoding rules care about is whether a pair of type specifications are extension-related or not. If they are, then **all** ASN.1 encoding rules will ensure interworking between their users.

A.1.9 We start with a type and then decide whether we are going to want interworking with earlier versions if we later have to extend it. If so, we include the extension marker **now**. We can then add later extension additions to the type without changing the bits on the line for earlier values, and with defined handling of extended values by earlier systems. It is, however, important to note that adding an extension marker to a type that was previously without one (or removing an extension marker) **will** in general change the bits on the line and will prevent interworking. Such a change generally requires a version number change in all affected protocols.

A.1.10 The following table shows the ASN.1 types that can form the extension root type of an ASN.1 extension series, and the nature of the single extension addition that is permitted for that type (multiple extension additions can of course be made in succession or together):

Extension additions

Extension root type	Nature of extension addition
ENUMERATED	Addition of a single further enumeration at the end of the "AdditionalEnumeration"s, with an enumeration value greater than that of any enumeration already added.
SEQUENCE and SET	Addition of a single "Component" to the end of the "AdditionalComponentTypeList". Note that there is no requirement for the added "ComponentType" to be marked OPTIONAL or DEFAULT, although this will often be the case.
CHOICE	Addition of a single "NamedType" to the end of the "AdditionalAlternativeTypeList".
Constraint notation	Addition of a single "AdditionalElementSetSpec" to the "ElementSetSpecs" notation.

A.2 Effects on version numbering, etc.

A.2.1 Where an ASN.1 specification is re-issued with type definitions changed in terms of extension-related type definitions, then for all purposes these changes do not in themselves require a change in the object identifier of the module or the version number of the protocol.

A.2.2 It may be that for other reasons such changes might be accompanied by version number changes, but this is not a requirement.

A.2.3 By contrast, the addition of an extension marker to a type that previously had none, or the addition of components to a sequence or set type (for example) with no extension marker, creates a new type which is **not** extension-related to the old type, and the module containing it should be given a new object identifier, and a new version number would be appropriate in the associated protocol.

A.3 Requirements on encoding rules

A.3.1 An abstract syntax can be defined as the values of a single ASN.1 type that is an extensible type. It then contains all the values that can be obtained by the addition or removal of extension-additions. Such an abstract syntax is called an extension-related abstract syntax.

A.3.2 A set of well-formed encoding rules for an extension-related abstract syntax satisfies the following additional requirements.

NOTE – All ASN.1 encoding rules satisfy these requirements.

A.3.3 The definition of the procedures for transforming an abstract value into an encoding for transfer, and for transforming a received encoding into an abstract value shall recognize the possibility that the sender and receiver are using abstract syntaxes that are not identical, but are extension-related.

A.3.4 In this case, the encoding rules shall ensure that where the sender has a type specification that is earlier in the extension series than that of the receiver, values of the sender shall be transferred to the in such a way that the receiver can determine that extension additions are not present.

A.3.5 The encoding rules shall ensure that where the sender has a type specification that is later in the extension series than that of the receiver, transfer of values of that type to the receiver shall be possible.