DRAFT INTERNATIONAL STANDARD

ISO/IEC DIS 8825-2

ISO/IEC JTC **1**/SC **6** Secretariat: **KATS**

Voting begins on: **2020-09-28** Voting terminates on: **2020-12-21**

Information technology — ASN.1 encoding rules —

Part 2:
Specification of Packed Encoding Rules (PER)

Technologies de l'information — Règles de codage ASN.1 —

Partie 2: Spécification des règles de codage compact (PER)

ICS: 35.100.60

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENT AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

This document is circulated as received from the committee secretariat.

Reference number
ISO/IEC DIS 8825-2:2020(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

CONTENTS

Page

Introduction

Specifications Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4 together describe Abstract Syntax Notation One (ASN.1), a notation for the definition of messages to be exchanged between peer applications.

This Recommendation | International Standard defines encoding rules that may be applied to values of types defined using the notation specified in Rec. ITU-T X.680 | ISO/IEC 8824-1. Application of these encoding rules produces a transfer syntax for such values. It is implicit in the specification of these encoding rules that they are also to be used for decoding.

There are more than one set of encoding rules that can be applied to values of ASN.1 types. This Recommendation | International Standard defines a set of Packed Encoding Rules (PER), so called because they achieve a much more compact representation than that achieved by the Basic Encoding Rules (BER) and its derivatives described in Rec. ITU-T X.690 | ISO/IEC 8825-1 which is referenced for some parts of the specification of these Packed Encoding Rules.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)

1 Scope

This Recommendation | International Standard specifies a set of Packed Encoding Rules that may be used to derive a transfer syntax for values of types defined in Rec. ITU-T X.680 | ISO/IEC 8824-1. These Packed Encoding Rules are also to be applied for decoding such a transfer syntax in order to identify the data values being transferred.

The encoding rules specified in this Recommendation | International Standard:

- are used at the time of communication;
- are intended for use in circumstances where minimizing the size of the representation of values is the major concern in the choice of encoding rules;
- allow the extension of an abstract syntax by addition of extra values, preserving the encodings of the existing values, for all forms of extension described in Rec. ITU-T X.680 | ISO/IEC 8824-1;
- can be modified in accordance with the provisions of Rec. ITU-T X.695 | ISO/IEC 8825-6.

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 edition of the Recommendations and Standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations.

NOTE – This Recommendation | International Standard is based on ISO/IEC 10646:2003. It cannot be applied using later versions of this standard.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.680 (2020) | ISO/IEC 8824-1:2020, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.
- Recommendation ITU-T X.681 (2020) | ISO/IEC 8824-2:2020, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.682 (2020) | ISO/IEC 8824-3:2020, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- Recommendation ITU-T X.683 (2020) | ISO/IEC 8824-4:2020, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- Recommendation ITU-T X.690 (2020) | ISO/IEC 8825-1:2020, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*
- Recommendation ITU-T X.695 (2020) | ISO/IEC 8825-6:2020, *Information technology – ASN.1 encoding rules: Registration and application of PER encoding instructions.*

2.2 Additional references

- ISO/IEC 646:1991, *Information technology – ISO 7-bit coded character set for information interchange*.
- ISO/IEC 2022:1994, *Information technology – Character code structure and extension techniques*.

- ISO/IEC 2375:2003, *Information technology – Procedure for registration of escape sequences and coded character sets.*
- ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange.*
- *ISO International Register of Coded Character Sets to be Used with Escape Sequences.*
- ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS).*

3 Definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.1 Specification of Basic Notation

For the purposes of this Recommendation | International Standard, all the definitions in Rec. ITU-T X.680 | ISO/IEC 8824-1 apply.

3.2 Information Object Specification

For the purposes of this Recommendation | International Standard, all the definitions in Rec. ITU-T X.681 | ISO/IEC 8824-2 apply.

3.3 Constraint Specification

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.682 | ISO/IEC 8824-3:

a) component relation constraint;

b) table constraint.

3.4 Parameterization of ASN.1 Specification

This Recommendation | International Standard makes use of the following term defined in Rec. ITU-T X.683 | ISO/IEC 8824-4:

- variable constraint.

3.5 Basic Encoding Rules

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.690 | ISO/IEC 8825-1:

a) dynamic conformance;

b) static conformance;

c) data value;

d) encoding (of a data value);

e) sender;

f) receiver.

3.6 PER Encoding Instructions

This Recommendation | International Standard makes use of the following term defined in Rec. ITU-T X.695 | ISO/IEC 8825-6:

- identifying keyword.

3.7 Additional definitions

For the purposes of this Recommendation | International Standard, the following definitions apply.

3.7.1 2's-complement-binary-integer encoding: The encoding of a whole number into a bit-field (octet-aligned in the ALIGNED variant) of a specified length, or into the minimum number of octets that will accommodate that whole number encoded as a 2's-complement-integer, which provides representations for whole numbers that are equal to, greater than, or less than zero, as specified in 11.4.

NOTE 1 – The value of a two's complement binary number is derived by numbering the bits in the contents octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the two's complement binary number is obtained by summing the numerical values assigned to each bit for those bits which are set to one, excluding bit 8 of the first octet, and then reducing this value by the numerical value assigned to bit 8 of the first octet if that bit is set to one.

NOTE 2 – *Whole number* is a synonym for the mathematical term *integer*. It is used here to avoid confusion with the ASN.1 type *integer*.

3.7.2 abstract syntax value: A value of an abstract syntax (defined as the set of values of a single ASN.1 type), which is to be encoded by PER, or which is to be generated by PER decoding.

NOTE – The single ASN.1 type associated with an abstract syntax is formally identified by an object of class **ABSTRACT-SYNTAX**.

3.7.3 bit-field: The product of some part of the encoding mechanism that consists of an ordered set of bits that are not necessarily a multiple of eight.

NOTE – If the use of this term is followed by "octet-aligned in the ALIGNED variant", this means that the bit-field is required to begin on an octet boundary in the complete encoding for the aligned variant of PER.

3.7.4 canonical encoding: A complete encoding of an abstract syntax value obtained by the application of encoding rules that have no implementation-dependent options; such rules result in the definition of a 1-1 mapping between unambiguous and unique bitstrings in the transfer syntax and values in the abstract syntax.

3.7.5 composite type: A set, sequence, set-of, sequence-of, choice, embedded-pdv, external or unrestricted character string type.

3.7.6 composite value: The value of a composite type.

3.7.7 constrained whole number: A whole number which is constrained by PER-visible constraints to lie within a range from "lb" to "ub" with the value "lb" less than or equal to "ub", and the values of "lb" and "ub" as permitted values.

NOTE – Constrained whole numbers occur in the encoding which identifies the chosen alternative of a choice type, the length of character, octet and bit string types whose length has been restricted by PER-visible constraints to a maximum length, the count of the number of components in a sequence-of or set-of type that has been restricted by PER-visible constraints to a maximum number of components, the value of an integer type that has been constrained by PER-visible constraints to lie within finite minimum and maximum values, and the value that denotes an enumeration in an enumerated type.

3.7.8 effective size constraint (for a constrained string type): A single finite size constraint that could be applied to a built-in string type and whose effect would be to permit all and only those lengths that can be present in the constrained string type.

NOTE 1 – For example, the following has an effective size constraint:

```
A ::= IA5String (SIZE(1..4) | SIZE(10..15))
```

since it can be rewritten with a single size constraint that applies to all values:

```
A ::= IA5String (SIZE(1..4 | 10..15))
```

whereas the following has no effective size constraint since the string can be arbitrarily long if it does not contain any characters other than 'a', 'b' and 'c':

```
B ::= IA5String (SIZE(1..4) | FROM("abc"))
```

NOTE 2 – The effective size constraint is used only to determine the encoding of lengths.

3.7.9 effective permitted-alphabet constraint (for a constrained restricted character string type): A single permitted-alphabet constraint that could be applied to a built-in known-multiplier character string type and whose effect would be to permit all and only those characters that can be present in at least one character position of any one of the values in the constrained restricted character string type.

NOTE 1 – For example, in:

```
Ax ::= IA5String (FROM("AB") | FROM("CD"))
Bx ::= IA5String (SIZE(1..4) | FROM("abc"))
```

Ax has an effective permitted-alphabet constraint of **"ABCD"**. **Bx** has an effective permitted-alphabet constraint that consists of the entire **IA5String** alphabet since there is no smaller permitted-alphabet constraint that applies to all values of **Bx**.

NOTE 2 – The effective permitted-alphabet constraint is used only to determine the encoding of characters.

3.7.10 enumeration index: The non-negative whole number associated with an "EnumerationItem" in an enumerated type. The enumeration indices are determined by sorting the "EnumerationItem"s into ascending order by their enumeration value, then by assigning an enumeration index starting with zero for the first "EnumerationItem", one for the second, and so on up to the last "EnumerationItem" in the sorted list.

NOTE – "EnumerationItem"s in the "RootEnumeration" are sorted separately from those in the "AdditionalEnumeration".

3.7.11 extensible for PER encoding: A property of a type which requires that PER identifies an encoding of a value as that of a root value or as that of an extension addition.

NOTE – Root values are normally encoded more efficiently than extension additions.

3.7.12 field-list: An ordered set of bit-fields that is produced as a result of applying these encoding rules to components of an abstract value.

3.7.13 indefinite-length: An encoding whose length is greater than 64K-1 or whose maximum length cannot be determined from the ASN.1 notation.

3.7.14 fixed-length type: A type such that the value of the outermost length determinant in an encoding of this type can be determined (using the mechanisms specified in this Recommendation | International Standard) from the type notation (after the application of PER-visible constraints only) and is the same for all possible values of the type.

3.7.15 fixed value: A value such that it can be determined (using the mechanisms specified in this Recommendation | International Standard) that this is the only permitted value (after the application of PER-visible constraints only) of the type governing it.

3.7.16 known-multiplier character string type: A restricted character string type where the number of octets in the encoding is a known fixed multiple of the number of characters in the character string for all permitted character string values. The known-multiplier character string types are **`IA5String`**, **`PrintableString`**, **`VisibleString`**, **`NumericString`**, **`UniversalString`** and **`BMPString`**.

3.7.17 length determinant: A count (of bits, octets, characters, or components) determining the length of part or all of a PER encoding.

3.7.18 normally small non-negative whole number: A part of an encoding which represents values of an unbounded non-negative integer, but where small values are more likely to occur than large ones.

3.7.19 normally small length: A length encoding which represents values of an unbounded length, but where small lengths are more likely to occur than large ones.

3.7.20 non-negative-binary-integer encoding: The encoding of a constrained or semi-constrained whole number into either a bit-field of a specified length, or into a bit-field (octet-aligned in the ALIGNED variant) of a specified length, or into the minimum number of octets that will accommodate that whole number encoded as a non-negative-binary-integer which provides representations for whole numbers greater than or equal to zero, as specified in 11.3.

NOTE – The value of a non-negative-binary-number is derived by numbering the bits in the contents octets, starting with bit 1 of the last octet as bit zero and ending the numbering with bit 8 of the first octet. Each bit is assigned a numerical value of 2^N, where N is its position in the above numbering sequence. The value of the non-negative-binary-number is obtained by summing the numerical values assigned to each bit for those bits which are set to one.

3.7.21 outermost type: An ASN.1 type whose encoding is included in a non-ASN.1 carrier or as the value of other ASN.1 constructs (see 11.1.1).

NOTE – PER encodings of an outermost type are always an integral multiple of eight bits.

3.7.22 PER-visible constraint: An instance of use of the ASN.1 constraint notation which affects the PER encoding of a value.

3.7.23 relay-safe encoding: A complete encoding of an abstract syntax value which can be decoded (including any embedded encodings) without knowledge of the environment in which the encoding was performed.

3.7.24 semi-constrained whole number: A whole number which is constrained by PER-visible constraints to exceed or equal some value "lb" with the value "lb" as a permitted value, and which is not a constrained whole number.

NOTE – Semi-constrained whole numbers occur in the encoding of the length of unconstrained (and in some cases constrained) character, octet and bit string types, the count of the number of components in unconstrained (and in some cases constrained) sequence-of and set-of types, and the value of an integer type that has been constrained to exceed some minimum value.

3.7.25 simple type: A type that is not a composite type.

3.7.26 textually dependent: A term used to identify the case where if some reference name is used in evaluating an element set, the value of the element set is considered to be dependent on that reference name, regardless of whether the actual set arithmetic being performed is such that the final value of the element set is independent of the actual element set value assigned to the reference name.

NOTE – For example, the following definition of **`Foo`** is textually dependent on **`Bar`** even though **`Bar`** has no effect on **`Foo`**s set of values (thus, according to 10.3.6 the constraint on **`Foo`** is not PER-visible since **`Bar`** is constrained by a table constraint and **`Foo`** is textually dependent on **`Bar`**).

```
MY-CLASS ::= CLASS { &name PrintableString, &age INTEGER } WITH SYNTAX{&name , &age}
MyObjectSet MY-CLASS ::= { {"Jack", 7} | {"Jill", 5} }
```

```
    Bar ::= MY-CLASS.&age ({MyObjectSet})
    Foo ::= INTEGER (Bar | 1..100)
```

3.7.27 **unconstrained whole number**: A whole number which is not constrained by PER-visible constraints.

NOTE – Unconstrained whole numbers occur only in the encoding of a value of the integer type.

4 Abbreviations

For the purposes of this Recommendation | International Standard, the following abbreviations apply:

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules of ASN.1
CER	Canonical Encoding Rules of ASN.1
DER	Distinguished Encoding Rules of ASN.1
PER	Packed Encoding Rules of ASN.1
16K	16384
32K	32768
48K	49152
64K	65536

5 Notation

This Recommendation | International Standard references the notation defined by Rec. ITU-T X.680 | ISO/IEC 8824-1.

6 Convention

6.1 This Recommendation | International Standard defines the value of each octet in an encoding by use of the terms "most significant bit" and "least significant bit".

NOTE – Lower layer specifications use the same notation to define the order of bit transmission on a serial line, or the assignment of bits to parallel channels.

6.2 For the purposes of this Recommendation | International Standard, the bits of an octet are numbered from 8 to 1, where bit 8 is the "most significant bit" and bit 1 the "least significant bit".

6.3 The term "octet" is frequently used in this Recommendation | International Standard to stand for "eight bits". The use of this term in place of "eight bits" does not carry any implications of alignment. Where alignment is intended, it is explicitly stated in this Recommendation | International Standard.

7 Encoding rules defined in this Recommendation | International Standard

7.1 This Recommendation | International Standard specifies four encoding rules (together with their associated object identifiers) which can be used to encode and decode the values of an abstract syntax defined as the values of a single (known) ASN.1 type. This clause describes their applicability and properties.

7.2 Without knowledge of the type of the value encoded, it is not possible to determine the structure of the encoding (under any of the PER encoding rule algorithms). In particular, the end of the encoding cannot be determined from the encoding itself without knowledge of the type being encoded.

7.3 PER encodings are always relay-safe provided the abstract values of the types **EXTERNAL**, **EMBEDDED PDV** and **CHARACTER STRING** are constrained to prevent the carriage of OSI presentation context identifiers.

7.4 The most general encoding rule algorithm specified in this Recommendation | International Standard is BASIC-PER, which does not in general produce a canonical encoding.

7.5 A second encoding rule algorithm specified in this Recommendation | International Standard is CANONICAL-PER, which produces encodings that are canonical. This is defined as a restriction of implementation-dependent choices in the BASIC-PER encoding.

NOTE 1 – CANONICAL-PER produces canonical encodings that have applications when authenticators need to be applied to abstract values.

NOTE 2 – Any implementation conforming to CANONICAL-PER for encoding is conformant to BASIC-PER for encoding. Any implementation conforming to BASIC-PER for decoding is conformant to CANONICAL-PER for decoding. Thus, encodings made according to CANONICAL-PER are encodings that are permitted by BASIC-PER.

7.6 If a type encoded with BASIC-PER or CANONICAL-PER contains **EMBEDDED PDV**, **CHARACTER STRING** or **EXTERNAL** types, then the outer encoding ceases to be relay-safe unless the transfer syntax used for all the **EMBEDDED PDV**, **CHARACTER STRING** and **EXTERNAL** types is relay safe. If a type encoded with CANONICAL-PER contains **EMBEDDED PDV**, **EXTERNAL** or **CHARACTER STRING** types, then the outer encoding ceases to be canonical unless the transfer syntax used for all the **EMBEDDED PDV**, **EXTERNAL** and **CHARACTER STRING** types is canonical.

NOTE – The character transfer syntaxes supporting all character abstract syntaxes of the form **{iso standard 10646 level-1(1) ....}** are canonical. Those supporting **{iso standard 10646 level-2(2) ....}** and **{iso standard 10646 level-3(3) ....}** are not always canonical. All the above character transfer syntaxes are relay-safe.

7.7 Both BASIC-PER and CANONICAL-PER come in two variants, the ALIGNED variant, and the UNALIGNED variant. In the ALIGNED variant, padding bits are inserted from time to time to restore octet alignment. In the UNALIGNED variant, no padding bits are ever inserted.

7.8 There are no interworking possibilities between the ALIGNED variant and the UNALIGNED variant.

7.9 PER encodings are self-delimiting only with knowledge of the type of the encoded value. Encodings are always a multiple of eight bits. When carried in an **EXTERNAL** type they shall be carried in the **OCTET STRING** choice alternative, unless the **EXTERNAL** type itself is encoded in PER, in which case the value may be encoded as a single ASN.1 type (i.e., an open type). When carried in OSI presentation protocol, the "full encoding" (as defined in Rec. ITU-T X.226 | ISO/IEC 8823-1) with the **OCTET STRING** choice alternative shall be used.

7.10 The rules of this Recommendation | International Standard apply to both algorithms and to both variants unless otherwise stated (but see 9.2 and 9.3).

7.11 Annex C is informative, and gives recommendations on which combinations of PER to implement in order to maximize the chances of interworking.

8 Conformance

8.1 Dynamic conformance is specified by clause 9 onwards.

8.2 Static conformance is specified by those standards which specify the application of these Packed Encoding Rules.

NOTE – Annex C provides guidance on static conformance in relation to support for the two variants of the two encoding rule algorithms. This guidance is designed to ensure interworking, while recognizing the benefits to some applications of encodings that are neither relay-safe nor canonical.

8.3 The rules in this Recommendation | International Standard are specified in terms of an encoding procedure. Implementations are not required to mirror the procedure specified, provided the bit string produced as the complete encoding of an abstract syntax value is identical to one of those specified in this Recommendation | International Standard for the applicable transfer syntax.

8.4 Implementations performing decoding are required to produce the abstract syntax value corresponding to any received bit string which could be produced by a sender conforming to the encoding rules identified in the transfer syntax associated with the material being decoded.

NOTE 1 – In general there are no alternative encodings defined for the BASIC-PER explicitly stated in this Recommendation | International Standard. The BASIC-PER becomes canonical by specifying relay-safe operation and by restricting some of the encoding options of other ISO/IEC Standards that are referenced. CANONICAL-PER provides an alternative to both the Distinguished Encoding Rules and Canonical Encoding Rules (see Rec. ITU-T X.690 | ISO/IEC 8825-1) where a canonical and relay-safe encoding is required.

NOTE 2 – When CANONICAL-PER is used to provide a canonical encoding, it is recommended that any resulting encrypted hash value that is derived from it should have associated with it an algorithm identifier that identifies CANONICAL-PER as the transformation from the abstract syntax value to an initial bitstring (which is then hashed).

9 PER encoding instructions

9.1 PER encoding instructions can be associated with a type in accordance with the provisions of Rec. ITU-T X.680 | ISO/IEC 8824-1 and Rec. ITU-T X.695 | ISO/IEC 8825-6.

NOTE 1 – The application of some PER encoding instructions can make it impossible to encode all the abstract values of the type. Where this can arise, the specific PER encoding instruction identifies the problem. It is a designers decision, based on the possible need to use multiple encoding rules, whether to add an explicit constraint on the type in order to restrict the range of abstract values to those that can be handled by the encoding using the PER encoding instruction. This can make the specification less readable, but ensures that all encoding rules can encode all allowed abstract values, making relaying possible without errors.

NOTE 2 – Each PER encoding instruction starts with an identifying keyword that unambiguously identifies that encoding instruction.

9.2 If the ALIGNED version of either BASIC-PER or CANONICAL-PER is in use, then all PER encoding instructions shall be silently ignored and have no affect on the encoding.

9.3 If the UNALIGNED version of either BASIC-PER or CANONICAL-PER is in use, then if a type has an associated encoding instruction, the following subclauses shall apply.

9.3.1 If the identifying keyword is not known, then a "not supported" error message shall be issued.

9.3.2 If the identifying keyword is known, the procedures of this Recommendation | International Standard shall be modified by the amendments to those procedures that are specified by the PER encoding instruction (see Rec. ITU-T X.695 | ISO/IEC 8825-6).

NOTE 1 – If multiple PER encoding instructions are associated with a type, then the amendments specified for all of them shall be applied.

NOTE 2 – It is an error in the register of PER encoding instructions if amendments produced by two or more separate encoding instructions conflict and it is not stated that they are mutually exclusive.

10 The approach to encoding used for PER

10.1 Use of the type notation

10.1.1 These encoding rules make specific use of the ASN.1 type notation as specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, and can only be applied to encode the values of a single ASN.1 type specified using that notation.

10.1.2 In particular, but not exclusively, they are dependent on the following information being retained in the ASN.1 type and value model underlying the use of the notation:

a) the nesting of choice types within choice types;

b) the tags placed on the components in a set type, and on the alternatives in a choice type, and the values given to an enumeration;

c) whether a set or sequence type component is optional or not;

d) whether a set or sequence type component has a **`DEFAULT`** value or not;

e) the restricted range of values of a type which arise through the application of PER-visible constraints (only);

f) whether a component is an open type;

g) whether a type is extensible for PER encoding.

10.2 Use of tags to provide a canonical order

This Recommendation | International Standard requires components of a set type and a choice type to be canonically ordered independent of the textual ordering of the components. The canonical order is determined by sorting the outermost tag of each component, as specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 8.6.

10.3 PER-visible constraints

NOTE – The fact that some ASN.1 constraints may not be PER-visible for the purposes of encoding and decoding does not in any way affect the use of such constraints in the handling of errors detected during decoding, nor does it imply that values violating such constraints are allowed to be transmitted by a conforming sender. However, this Recommendation | International Standard makes no use of such constraints in the specification of encodings.

10.3.1 Constraints that are expressed in human-readable text or in ASN.1 comment are not PER-visible.

10.3.2 Variable constraints are not PER-visible (see Rec. ITU-T X.683 | ISO/IEC 8824-4, 10.3 and 10.4).

10.3.3 User-defined constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 9.1) are not PER visible.

10.3.4 Table constraints are not PER-visible (see Rec. ITU-T X.682 | ISO/IEC 8824-3).

10.3.5 Component relation constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 10.7) are not PER-visible.

10.3.6 Constraints whose evaluation is textually dependent on a table constraint or a component relation constraint are not PER-visible (see Rec. ITU-T X.682 | ISO/IEC 8824-3).

10.3.7 Constraints on restricted character string types which are not (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 41) known-multiplier character string types are not PER-visible (see 3.7.16).

10.3.8 Pattern constraints are not PER-visible.

10.3.9 Subject to the above, all size constraints are PER-visible.

10.3.10 The effective size constraint for a constrained type is a single size constraint such that a size is permitted if and only if there is some value of the constrained type that has that (permitted) size.

10.3.11 Permitted-alphabet constraints on known-multiplier character string types which are not extensible after application of Rec. ITU-T X.680 | ISO/IEC 8824-1, 52.3 to 52.5, are PER-visible. Permitted-alphabet constraints which are extensible are not PER-visible.

10.3.12 The effective permitted-alphabet constraint for a constrained type is a single permitted-alphabet constraint which allows a character if and only if there is some value of the constrained type that contains that character. If all characters of the type being constrained can be present in some value of the constrained type, then the effective permitted-alphabet constraint is the set of characters defined for the unconstrained type.

10.3.13 Property setting constraints on the time type (or on the useful and defined time types) which are not extensible after the application of Rec. ITU-T X.680 | ISO/IEC 8824-1, 52.3 to 52.5, are PER-visible. Property setting constraints which are extensible are not PER-visible.

10.3.14 Constraints applied to real types are not PER-visible.

10.3.15 An inner type constraint applied to an unrestricted character string or embbeded-pdv type is PER-visible only when it is used to restrict the value of the **`syntaxes`** component to a single value, or when it is used to restrict **`identification`** to the **`fixed`** alternative (see clauses 28 and 31).

10.3.16 Constraints on the useful types are not PER-visible.

10.3.17 Single value subtype constraints applied to a character string type are not PER-visible.

10.3.18 Subject to the above, all other constraints are PER-visible if and only if they are applied to an integer type or to a known-multiplier character string type.

10.3.19 In general the constraint on a type will consist of individual constraints combined using some or all of set arithmetic, contained subtype constraints, and serial application of constraints. The following clauses specify the effect if some of the component parts of the total constraint are PER-visible and some are not.

NOTE – See Annex B for further discussion on the effect of combining constraints that individually are PER-visible or not PER-visible.

10.3.20 If a constraint consists of a serial application of constraints, the constraints which are not PER-visible, if any, do not affect PER encodings, but cause the extensibility (and extension additions) present in any earlier constraints to be removed as specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 50.11.

NOTE 1 – If the final constraint in a serial application is not PER-visible, then the type is not extensible for PER-encodings, and is encoded without an extension bit.

NOTE 2 – For example:

```
A ::= IA5String(SIZE(1..4))(FROM("ABCD",...))
```

has an effective permitted-alphabet constraint that consists of the entire **`IA5String`** alphabet since the extensible permitted-alphabet constraint is not PER-visible. It has nevertheless an effective size constraint which is "**`SIZE(1..4)`**".

Similarly,

```
B ::= IA5String(A)
```

has the same effective size constraint and the same effective permitted-alphabet constraint.

10.3.21 If a constraint that is PER-visible is part of an **`INTERSECTION`** construction, then the resulting constraint is PER-visible, and consists of the **`INTERSECTION`** of all PER-visible parts (with the non-PER-visible parts ignored). If a constraint which is not PER-visible is part of a **`UNION`** construction, then the resulting constraint is not PER-visible. If a constraint has an **`EXCEPT`** clause, the **`EXCEPT`** and the following value set is completely ignored, whether the value set following the **`EXCEPT`** is PER-visible or not.

NOTE – For example:

```
A ::= IA5String (SIZE(1..4) INTERSECTION FROM("ABCD",...))
```

has an effective size constraint of 1..4 but the alphabet constraint is not visible because it is extensible.

10.3.22 A type is also extensible for PER encodings (whether subsequently constrained or not) if any of the following occurs:

a) it is derived from an **ENUMERATED** type (by subtyping, type referencing, or tagging) and there is an extension marker in the "Enumerations" production; or

b) it is derived from a **SEQUENCE** type (by subtyping, type referencing, or tagging) and there is an extension marker in the "ComponentTypeLists" or in the "SequenceType" productions; or

c) it is derived from a **SET** type (by subtyping, type referencing, or tagging) and there is an extension marker in the "ComponentTypeLists" or in the "SetType" productions; or

d) it is derived from a **CHOICE** type (by subtyping, type referencing, or tagging) and there is an extension marker in the "AlternativeTypeLists" production.

10.4 Type and value model used for encoding

10.4.1 An ASN.1 type is either a simple type or is a type built using other types. The notation permits the use of type references and tagging of types. For the purpose of these encoding rules, the use of type references and tagging have no effect on the encoding and are invisible in the model, except as stated in 10.2. The notation also permits the application of constraints and of error specifications. PER-visible constraints are present in the model as a restriction of the values of a type. Other constraints and error specifications do not affect encoding and are invisible in the PER type and value model.

10.4.2 A value to be encoded can be considered as either a simple value or as a composite value built using the structuring mechanisms from components which are either simple or composite values, paralleling the structure of the ASN.1 type definition.

10.4.3 When a constraint includes a value as an extension addition that is present in the root, that value is always encoded as a value in the root, not as a value which is an extension addition.

EXAMPLE

INTEGER (0..10, ..., 5)

-- *The value 5 encodes as a root value, not as an extension addition.*

10.5 Structure of an encoding

10.5.1 These encoding rules specify:

a) the encoding of a simple value into a field-list; and

b) the encoding of a composite value into a field-list, using the field-lists generated by application of these encoding rules to the components of the composite value; and

c) the transformation of the field-list of the outermost value into the complete encoding of the abstract syntax value (see 11.1).

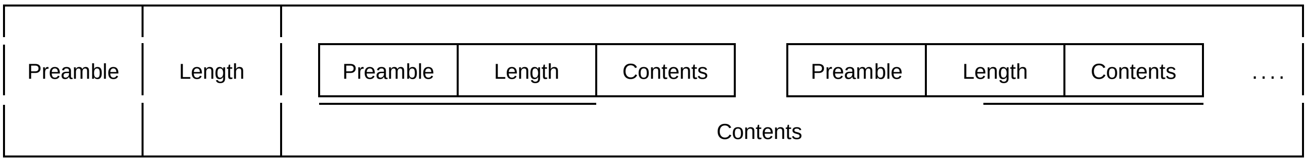
10.5.2 The encoding of a component of a data value either:

a) consists of three parts, as shown in Figure 1, which appear in the following order:

1) a preamble (see clauses 19, 21 and 23);

2) a length determinant (see 11.9);

3) contents; or



NOTE – The preamble, length, and contents are all "fields" which, concatenated together, form a "field-list". The field-list of a composite type other than the choice type may consist of the fields of several values concatenated together. Either the preamble, length and/or contents of any value may be missing.

Figure 1 – Encoding of a composite value into a field-list