INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

X.691

(04/95)

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

DATA NETWORKS AND OPEN SYSTEM COMMUNICATIONS

OSI NETWORKING AND SYSTEM ASPECTS – ABSTRACT SYNTAX NOTATION ONE (ASN.1)

INFORMATION TECHNOLOGY – ASN.1 ENCODING RULES – SPECIFICATION OF PACKED ENCODING RULES (PER)

ITU-T Recommendation X.691

Superseded by a more recent version

(Previously "CCITT Recommendation")

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.691 was approved on 10th of April 1995. The identical text is also published as ISO/IEC International Standard 8825-2.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized 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Superseded by a more recent version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

			Page
1	Scope		1
2	Normative references		1
	2.1	Identical Recommendations | International Standards	1
	2.2	Additional references	2
3	Definitions		2
	3.1	Basic Presentation Service Definition	2
	3.2	Specification of Basic Notation	2
	3.3	ASN.1 Extensibility	2
	3.4	Information Object Specification	2
	3.5	Constraint Specification	2
	3.6	Parameterization of ASN.1 Specification	3
	3.7	Basic Encoding Rules	3
	3.8	Additional definitions	3
4	Abbreviations		5
5	Notation		5
6	Convention		6
7	Encoding rules defined in this Recommendation | International Standard		6
8	Conformance		7
9	The approach to encoding used for PER		7
	9.1	Use of the type notation	7
	9.2	Use of tags to provide a canonical order	7
	9.3	PER-visible constraints	7
	9.4	Type and value model used for encoding	8
	9.5	Structure of an encoding	9
	9.6	Types to be encoded	9
10	Encoding procedures		10
	10.1	Production of the complete encoding	10
	10.2	Open type fields	10
	10.3	Encoding as a non-negative-binary-integer	10
	10.4	Encoding as a 2's-complement-binary-integer	11
	10.5	Encoding of a constrained whole number	11
	10.6	Encoding of a normally small non-negative whole number	12
	10.7	Encoding of a semi-constrained whole number	12
	10.8	Encoding of an unconstrained whole number	13
	10.9	General rules for encoding a length determinant	13
11	Encoding the boolean type		15
12	Encoding the integer type		16
13	Encoding the enumerated type		16
14	Encoding the real type		17
15	Encoding the bitstring type		17
16	Encoding the octetstring type		18
17	Encoding the null type		19
18	Encoding the sequence type		19
19	Encoding the sequence-of type		19

Page

20 Encoding the set type 20

21 Encoding the set-of type 21

22 Encoding the choice type 21

23 Encoding the object identifier type 21

24 Encoding the embedded-pdv type 22

25 Encoding of a value of the external type 23

26 Encoding the restricted character string types 24

27 Encoding the unrestricted character string type 25

28 Object identifiers for transfer syntaxes 27

Annex A – Example of encodings 28

A.1 Record that does not use subtype constraints 28

A.2 Record that uses subtype constraints 31

A.3 Record that uses extension markers 34

Annex B – Observations on combining PER-visible constraints 38

Annex C – Support for the PER algorithms 39

Annex D – Support for the ASN.1 rules of extensibility 40

Annex E – Tutorial annex on concatenation of PER encodings 41

Annex F – Assignment of object identifier values 42

Summary

This Recommendation | International Standard describes a set of encoding rules that can be applied to values of all ASN.1 types to achieve a more compact representation than that achieved by the Basic Encoding Rules and its derivatives (described in Recommendation X.690).

Introduction

The publications ITU-T Rec. X.680 | ISO/IEC 8824-1, ITU-T Rec. X.681 | ISO/IEC 8824-2, ITU-T Rec. X.682 | ISO/IEC 8824-3, ITU-T Rec. X.683 | ISO/IEC 8824-4, ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1, ITU-T Rec. X.681/Amd. 1 | ISO/IEC 8824-2/Amd. 1 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 ITU-T Rec. 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 more compact representation than that achieved by the Basic Encoding Rules (BER) and its derivatives described in ITU-T Rec. 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 ITU-T Rec. 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 ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1.

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.200 (1994) | ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*.
- ITU-T Recommendation X.216 (1994) | ISO/IEC 8822:1994, *Information technology – Open Systems Interconnection – Presentation service definition*.
- ITU-T Recommendation X.226 (1994) | ISO/IEC 8823-1:1994, *Information technology – Open Systems Interconnection – Connection-oriented presentation protocol: Protocol specification.*
- 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.680 (1994)/Amd. 1 (1995) | ISO/IEC 8824-1:1995/Amd. 1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation – Amendment 1: Rules of extensibility*.
- ITU-T Recommendation X.681 (1994) | ISO/IEC 8824-2:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- ITU-T Recommendation X.681 (1994)/Amd. 1 (1995) | ISO/IEC 8824-2:1995/Amd. 1:1995, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification – Amendment 1: Rules of extensibility*.
- 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)*.

2.2 Additional references

– CCITT Rec. X.208 (1988), *Specification of Abstract Syntax Notation One (ASN.1)*.

– ISO/IEC 2022:1994, *Information technology – Character code structure and extension techniques*.

– ISO 2375:1985, *Data processing – Procedure for registration of escape sequences.*

– ISO 6093:1985, *Information processing – Representation of numerical values in character strings for information interchange.*

– ISO/IEC 8824:1990, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1).*

– *ISO International Register of Coded Character Sets to be Used with Escape Sequences.*

– ISO/IEC 10646-1:1993, *Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane.*

3 Definitions

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

3.1 Basic Presentation Service Definition

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.216 | ISO/IEC 8822:

a) defined context set;

b) presentation context identifier.

3.2 Specification of Basic Notation

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

3.3 ASN.1 Extensibility

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1:

a) extension marker;

b) extension series;

c) extension root;

d) extension addition.

3.4 Information Object Specification

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

3.5 Constraint Specification

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

a) component relation constraint;

b) table constraint.

3.6 Parameterization of ASN.1 Specification

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

– variable constraint.

3.7 Basic Encoding Rules

This Recommendation | International Standard makes use of the following terms defined in ITU-T Rec. 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.

3.8 Additional definitions

In addition, the following definitions apply.

3.8.1 2's-complement-binary-integer encoding: The encoding of a whole number into an octet-aligned-bit-field 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 10.4.

NOTES

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.

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

3.8.2 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.

3.8.3 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, and do not necessarily begin on an octet boundary in the complete encoding of the abstract syntax value.

3.8.4 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.

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

3.8.6 composite value: The value of a composite type.

3.8.7 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.

3.8.8 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 – 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"))

3.8.9 effective PermittedAlphabet constraint (for a constrained restricted character string type): A single PermittedAlphabet 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 any character position of any of the values in the constrained restricted character string type.

NOTE – An effective PermittedAlphabet constraint is either the entire alphabet of the unconstrained character string type or a PermittedAlphabet specification that happens to be a superset of all PermittedAlphabet constraints imposed on the type. For example, in

Ax ::= IA5String (FROM("AB") | FROM("CD"))

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

"Ax" has an effective PermittedAlphabet constraints that consist of the entire IA5String alphabet since there is no PermittedAlphabet constraint that applies to all values of "Ax". The same is true for "Bx". On the other hand, the following has an effective PermittedAlphabet constraint of "ABCDE" since there is a PermittedAlphabet constraint specified that applies to all values:

A ::= IA5String (FROM("AB") | FROM("CD") | FROM("ABCDE"))

3.8.10 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".

3.8.11 extensible for PER encoding: A property of a type whose definition contains an extension marker that affects the PER encoding.

3.8.12 field-list: An ordered set of bit-field and/or octet-aligned-bit-field values that is produced as a result of applying these encoding rules to components of a value.

NOTE – (Tutorial) The model employed in this Recommendation | International Standard uses the term "field-list" to indicate a linked list of buffers each containing an encoding, a length in bits, and an octet alignment indicator of "bit-field" or "octet-aligned-bit-field". Each encoding is that of a value of an ASN.1 type. The octet alignment indicator says whether the encoding is to be aligned on an octet boundary when used to form the complete encoding of the abstract syntax value, or if it should be added immediately after the last bit of the previous encoding in the complete encoding. The notion of "field-list" is for descriptive purposes only and does not suggest an implementation method.

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

3.8.14 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.

3.8.15 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.

3.8.16 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.

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

3.8.18 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.

3.8.19 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.

3.8.20 octet-aligned-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, but which are required to begin on an octet boundary in the complete encoding of the abstract syntax value.

3.8.21 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 an octet-aligned-bit-field 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 10.3.

NOTE – 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.

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

3.8.23 relay-safe encoding: A complete encoding of an abstract syntax value which can be decoded (including any embedded encodings) without knowledge of the presentation layer defined context set that formed the environment in which the encoding was performed.

3.8.24 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.

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

3.8.26 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 9.3.4 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)

3.8.27 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

ASN.1 Abstract Syntax Notation One

BER Basic Encoding Rules of ASN.1

PER Packed Encoding Rules of ASN.1

CER Canonical Encoding Rules of ASN.1

16K 16384

32K 32768

48K 49152

64K 65536

5 Notation

This Recommendation | International Standard references the notation defined by ITU-T Rec. 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 purpose 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 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. CANONICAL-PER produces canonical encodings that have applications when authenticators need to be applied to abstract values, as described in ITU-T Rec. X.690 | ISO/IEC 8825-1, Annex D.

NOTE – 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 BASIC-PER or 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 ITU-T Rec. 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.

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 of this Recommendation | International Standard 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.

NOTES

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 ITU-T Rec. X.690 | ISO/IEC 8825-1) where a canonical and relay-safe encoding is required.

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 The approach to encoding used for PER

9.1 Use of the type notation

9.1.1 These encoding rules make specific use of the ASN.1 type notation as specified in ITU-T Rec. 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.

9.1.2 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 an extension marker is present.

9.2 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 tags of the components, as specified in 6.4 of ITU-T Rec. X.680 | ISO/IEC 8824-1.

9.3 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.

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

9.3.2 Variable constraints are not PER-visible (see 10.4 and 10.5 of ITU-T Rec. X.683 | ISO/IEC 8824-4).

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

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

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

9.3.6 Constraints on restricted character string types which are not (see clause 34 of ITU-T Rec. X.680 | ISO/IEC 8824-1) known-multiplier character string types are not PER-visible (see 3.8.16).

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

9.3.8 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. If the constrained type has values of a size that does not satisfy the constraint there is no effective size constraint.

9.3.9 PermittedAlphabet constraints on known-multiplier character string types are PER-visible.

9.3.10 The effective PermittedAlphabet constraint for a constrained type is a single PermittedAlphabet constraint such that a character is allowed 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 PermittedAlphabet constraint is the set of characters defined for the unconstrained type.

NOTES

1 In the definition of a constrained type, multiple PER-visible constraints may be applied either directly or through the use of "ContainedSubtype"s.

2 See Annex B for observations on the effect of combining constraints that individually are PER-visible.

9.3.11 An inner type constraint applied to a real type is PER-visible.

9.3.12 An inner type constraint applied to an unrestricted character string or embedded-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 24 and 27).

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

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

9.3.15 If a PER-visible constraint other than PermittedAlphabet has an extension marker, then the type is defined to be extensible for PER encodings.

NOTES

1 If an extension marker is present in a ConstraintSpec which is not PER-visible, and there is no other extension marker present in the constraint, then the type is encoded by PER as if it has no extension marker.

2 If there are multiple SizeConstraint specifications applied to a type and one of them is extensible, then the type is encoded in PER as if the extension marker was present on all the SizeConstraint specifications.

9.3.16 A type is also extensible for PER encodings 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.

9.4 Type and value model used for encoding

9.4.1 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 9.2. 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.

9.4.2 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.

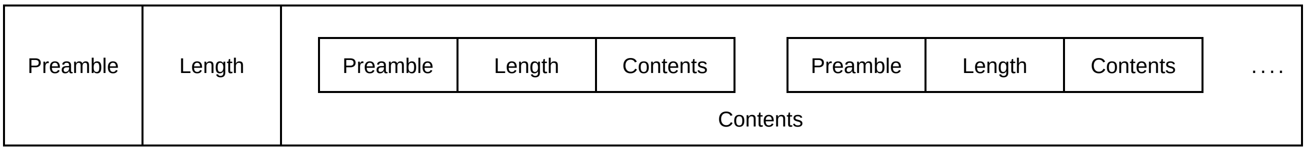
9.5 Structure of an encoding

9.5.1 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 10.1).



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

9.5.2 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 18, 20 and 22);

2) a length determinant (see 10.9);

3) contents; or

b) (where the contents are large) consists of an arbitrary number of parts, as shown in Figure 2, of which the first is a preamble (see clauses 18, 20 and 22) and the following parts are pairs of octet-aligned-bit-fields, the first being a length determinant for a fragment of the contents, and the second that fragment of the contents; the last pair of fields is identified by the length determinant part, as specified in 10.9.

Preamble	Length	Contents	Length	Contents	. . . .	Length	Contents (may be missing)

Figure 2 – Encoding of a long data value

9.5.3 Each of the parts mentioned in 9.5.2 generates either:

a) a null field (nothing); or

b) a bit-field; or

c) an octet-aligned-bit-field; or

d) a field-list which may contain either bit-fields, octet-aligned-bit-fields, or both.

9.6 Types to be encoded

9.6.1 The following clauses specify the encoding of the following types into a field-list: boolean, integer, enumerated, real, bitstring, octetstring, null, sequence, sequence-of, set, set-of, choice, open, object identifier, embedded-pdv, external, restricted character string and unrestricted character string types.

9.6.2 The ANY type, defined in CCITT Rec. X.208 (1988) | ISO/IEC 8824:1990, shall be encoded as an open type.

9.6.3 The selection type shall be encoded as an encoding of the selected type.

9.6.4 Encoding of tagged types is not included in this Recommendation | International Standard as, except as stated in 9.2, tagging is not visible in the type and value model used for these encoding rules. Tagged types are thus encoded according to the encoding of the type which has been tagged.

9.6.5 The following "useful types" of clause 38 of ITU-T Rec. X.680 | ISO/IEC 8824-1 shall be encoded as if they had been replaced by their definitions given in ITU-T Rec. X.680 | ISO/IEC 8824-1:

– generalized time;

– universal time;

– object descriptor.

Constraints on the useful types are not PER-visible.

10 Encoding procedures

10.1 Production of the complete encoding

10.1.1 The field-list produced as a result of applying this Recommendation | International Standard to the outermost value shall be used to produce the complete encoding of the abstract syntax value as follows: each field in the field-list shall be taken in turn and concatenated to the end of the bit string which is to form the complete encoding of the abstract syntax value preceded by additional zero bits for padding as specified below.

10.1.2 In the UNALIGNED variant of these encoding rules, all fields shall be concatenated without padding. If the result of encoding the outermost value is an empty bit string, the bit string shall be replaced with a single octet with all bits set to 0. If it is a non-empty bit string and it is not a multiple of eight bits, (zero to seven) zero bits shall be appended to it to produce a multiple of eight bits.

10.1.3 In the ALIGNED variant of these encoding rules, any bit-fields in the field-list shall be concatenated without padding, and any octet-aligned-bit-fields shall be concatenated after (zero to seven) zero bits have been concatenated to make the length of the encoding produced so far a multiple of eight bits. If the result of encoding the outermost value is an empty bit string, the bit string shall be replaced with a single octet with all bits set to 0. If it is a non-empty bit string and it is not a multiple of eight bits, (zero to seven) zero bits shall be appended to it to produce a multiple of eight bits.

NOTE – The encoding of the outermost value is the empty bit string if, for example, the abstract syntax value is of the null type or of an integer type constrained to a single value.

10.1.4 The resulting bit string is the complete encoding of the abstract syntax value.

10.2 Open type fields

10.2.1 In order to encode an open type field, the value of the actual type occupying the field shall be encoded to a field-list which shall then be converted to a complete encoding of an abstract syntax value as specified in 10.1 to produce an octet string of length "n" (say).

10.2.2 The field-list for the value in which the open type is to be embedded shall then have added to it (as specified in 10.9) an unconstrained length of "n" (in units of octets) and an associated octet-aligned-bit-field containing the bits produced in 10.2.1.

NOTE – Where the number of octets in the open type encoding is large, the fragmentation procedures of 10.9 will be used, and the encoding of the open type will be broken without regard to the position of the fragment boundary in the encoding of the type occupying the open type field.

10.3 Encoding as a non-negative-binary-integer

NOTE – (Tutorial) This subclause gives precision to the term "non-negative-binary-integer encoding", putting the integer into a field which is a fixed number of bits, a field which is a fixed number of octets, or a field that is the minimum number of octets needed to hold it.

10.3.1 Subsequent clauses refer to the generation of a non-negative-binary-integer encoding of a non-negative whole number into either a bit-field of specified length, a single octet, a double octet, or the minimum number of octets for the value. This clause specifies the precise encoding to be applied when such references are made.

10.3.2 The leading bit of the field is defined as the most significant bit of the first octet, and the trailing bit of the field is defined as the least significant bit of the last octet.

10.3.3 For the following definition only, the bits shall be numbered zero for the trailing bit of the field, one for the next bit, and so on up to the leading bit of the field.

10.3.4 In a non-negative-binary-integer encoding, the value of the whole number represented by the encoding shall be the sum of the values specified by each bit. A bit which is set to "0" has zero value. A bit with number "n" which is set to "1" has the value 2^n.

10.3.5 The encoding which sums (as defined above) to the value being encoded is an encoding of that value.

NOTE – Where the size of the encoded field is fixed (a bit-field of specified length, a single octet, or a double octet), then there is a unique encoding which sums to the value being encoded.

10.3.6 A minimum octet non-negative-binary-integer encoding of the whole number (which does not predetermine the number of octets to be used for the encoding) has a field which is a multiple of eight bits and also satisfies the condition that the leading eight bits of the field shall not all be zero unless the field is precisely eight bits long.

NOTE – This is a necessary and sufficient condition to produce a unique encoding.

10.4 Encoding as a 2's-complement-binary-integer

NOTE – (Tutorial) This subclause gives precision to the term "2's-complement-binary-integer encoding", putting a signed integer into a field that is the minimum number of octets needed to hold it. These procedures are referenced in later encoding specifications.

10.4.1 Subsequent clauses refer to the generation of a 2's-complement-binary-integer encoding of a whole number (which may be negative, zero, or positive) into the minimum number of octets for the value. This subclause specifies the precise encoding to be applied when such references are made.

10.4.2 The leading bit of the field is defined as the most significant bit of the first octet, and the trailing bit of the field is defined as the least significant bit of the last octet.

10.4.3 For the following definition only, the bits shall be numbered zero for the trailing bit of the field, one for the next bit, and so on up to the leading bit of the field.

10.4.4 In a 2's-complement-binary-integer encoding, the value of the whole number represented by the encoding shall be the sum of the values specified by each bit. A bit which is set to "0" has zero value. A bit with number "n" which is set to "1" has the value 2^n unless it is the leading bit, in which case it has the (negative) value -2^n.

10.4.5 Any encoding which sums (as defined above) to the value being encoded is an encoding of that value.

10.4.6 A minimum octet 2's-complement-binary-integer encoding of the whole number has a field-width that is a multiple of eight bits and also satisfies the condition that the leading nine bits of the field shall not all be zero and shall not all be ones.

NOTE – This is a necessary and sufficient condition to produce a unique encoding.

10.5 Encoding of a constrained whole number

NOTE – (Tutorial) This subclause is referenced by other clauses, and itself references earlier clauses for the production of a non-negative-binary-integer or a 2's-complement-binary-integer encoding. For the UNALIGNED variant the value is always encoded in the minimum number of bits necessary to represent the range (defined in 10.5.3). The rest of this note addresses the ALIGNED variant. Where the range is less than or equal to 255, the value encodes into a bit-field of the minimum size for the range. Where the range is exactly 256, the value encodes into a single octet octet-aligned-bit-field. Where the range is 257 to 64K, the value encodes into a two octet octet-aligned-bit-field. Where the range is greater than 64K, the range is ignored and the value encodes into an octet-aligned-bit-field which is the minimum number of octets for the value. In this latter case, later procedures (see 10.9) also encode a length field (usually a single octet) to indicate the length of the encoding. For the other cases, the length of the encoding is independent of the value being encoded, and is not explicitly encoded.

10.5.1 This subclause specifies a mapping from a constrained whole number into either a bit-field or an octet-aligned-bit-field, and is invoked by later clauses in this Recommendation | International Standard.

10.5.2 The procedures of this subclause are invoked only if a constrained whole number to be encoded is available, and the values of the lower bound, "lb", and the upper bound, "ub", have been determined from the type notation (after the application of PER-visible constraints).

NOTE – A lower bound cannot be determined if MIN evaluates to an infinite number, nor can an upper bound be determined if MAX evaluates to an infinite number. For example, no upper or lower bound can be determined for INTEGER(MIN..MAX).

10.5.3 Let "range" be defined as the integer value ("ub" – "lb" + 1), and let the value to be encoded be "n".

10.5.4 If "range" has the value 1, then the result of the encoding shall be an empty bit-field (no bits).

10.5.5 There are five other cases (leading to different encodings) to consider, where one applies to the UNALIGNED variant and four to the ALIGNED variant.

10.5.6 In the case of the UNALIGNED variant the value ("n" – "lb") shall be encoded as a non-negative-binary-integer in a bit-field as specified in 10.3 with the minimum number of bits necessary to represent the range.

NOTE – If "range" satisfies the inequality 2^m < "range" <= 2^{m+1}, then the number of bits = m + 1.

10.5.7 In the case of the ALIGNED variant the encoding depends on whether:

a) "range" is less than or equal to 255 (the bit-field case);

b) "range" is exactly 256 (the one-octet case);

c) "range" is greater than 256 and less than or equal to 64K (the two-octet case);

d) "range" is greater than 64K (the indefinite length case).

10.5.7.1 (The bit-field case.) If "range" is less than or equal to 255, then invocation of this clause requires the generation of a bit-field with a number of bits as specified in the table below, and containing the value ("n" – "lb") as a non-negative-binary-integer encoding in a bit-field as specified in 10.3.

"Range"	Bit-field size (in bits)
2	1
3, 4	2
5, 6, 7, 8	3
9 to 16	4
17 to 32	5
33 to 64	6
65 to 128	7
129 to 255	8

10.5.7.2 (The one-octet case.) If the range has a value of 256, then the value ("n" – "lb") shall be encoded in a one-octet octet-aligned-bit-field as a non-negative-binary-integer as specified in 10.3.

10.5.7.3 (The two-octet case.) If the "range" has a value greater than or equal to 257 and less than or equal to 64K, then the value ("n" – "lb") shall be encoded in a two-octet octet-aligned-bit-field as a non-negative-binary-integer encoding as specified in 10.3

10.5.7.4 (The indefinite length case.) Otherwise, the value ("n" – "lb") shall be encoded as a non-negative-binary-integer in an octet-aligned-bit-field with the minimum number of octets as specified in 10.3, and the number of octets "len" used in the encoding is used by other clauses that reference this subclause to specify an encoding of the length.

10.6 Encoding of a normally small non-negative whole number

NOTE – (Tutorial) This procedure is used when encoding a non-negative whole number that is expected to be small, but whose size is potentially unlimited due to the presence of an extension marker. An example is a choice index.

10.6.1 If the non-negative whole number, "n", is less than or equal to 63 then a single-bit bit-field shall be appended to the field-list with the bit set to 0, and "n" shall be encoded as a non-negative-binary-integer into a 6-bit bit-field.

10.6.2 If "n" is greater than or equal to 64, a single-bit bit-field with the bit set to 1 shall be appended to the field-list. The value "n" shall then be encoded as a semi-constrained whole number with "lb" equal to 0 and the procedures of 10.9 shall be invoked to add it to the field-list preceded by a length determinant.

10.7 Encoding of a semi-constrained whole number

NOTE – (Tutorial) This procedure is used when a lower bound can be identified but not an upper bound. The encoding procedure places the offset from the lower bound into the minimum number of octets as a non-negative-binary-integer, and requires an explicit length encoding (typically a single octet) as specified in later procedures.

10.7.1 This subclause specifies a mapping from a semi-constrained whole number into an octet-aligned-bit-field, and is invoked by later clauses in this Recommendation | International Standard.

10.7.2 The procedures of this clause are invoked only if a semi-constrained whole number ("n" say) to be encoded is available, and the value of "lb" has been determined from the type notation (after the application of PER-visible constraints).

NOTE – A lower bound cannot be determined if MIN evaluates to an infinite number. For example, no lower bound can be determined for INTEGER(MIN..MAX).

10.7.3 The procedures of this subclause always produce the indefinite length case.

10.7.4 (The indefinite length case.) The value ("n" – "lb") shall be encoded as a non-negative-binary-integer in an octet-aligned-bit-field with the minimum number of octets as specified in 10.3, and the number of octets "len" used in the encoding is used by other clauses that reference this clause to specify an encoding of the length.

10.8 Encoding of an unconstrained whole number

NOTE – (Tutorial) This case only arises in the encoding of the value of an integer type with no lower bound. The procedure encodes the value as a 2's-complement-binary-integer into the minimum number of octets required to accommodate the encoding, and requires an explicit length encoding (typically a single octet) as specified in later procedures.

10.8.1 This subclause specifies a mapping from an unconstrained whole number ("n" say) into an octet-aligned-bit-field, and is invoked by later clauses in this Recommendation | International Standard.

10.8.2 The procedures of this subclause always produce the indefinite length case.

10.8.3 (The indefinite length case.) The value "n" shall be encoded as a 2's-complement-binary-integer in an octet-aligned-bit-field with the minimum number of octets as specified in 10.4, and the number of octets "len" used in the encoding is used by other clauses that reference this subclause to specify an encoding of the length.

10.9 General rules for encoding a length determinant

NOTES

1 (Tutorial) The procedures of this subclause are invoked when an explicit length field is needed for some part of the encoding regardless of whether the length count is bounded above (by PER-visible constraints) or not. The part of the encoding to which the length applies may be a bit string (with the length count in bits), an octet string (with the length count in octets), a known-multiplier character string (with the length count in characters), or a list of fields (with the length count in components of a sequence-of or set-of).

2 (Tutorial) In the case of the ALIGNED variant if the length count is bounded above by an upper bound that is less than 64K, then the constrained whole number encoding is used for the length. For sufficiently small ranges the result is a bit-field, otherwise the unconstrained length ("n" say) is encoded into an octet-aligned-bit-field in one of three ways (in order of increasing size):

a) ("n" less than 128) a single octet containing "n" with bit 8 set to zero;

b) ("n" less than 16K) two octets containing "n" with bit 8 of the first octet set to 1 and bit 7 set to zero;

c) (large "n") a single octet containing a count "m" with bit 8 set to 1 and bit 7 set to 1. The count "m" is one to four, and the length indicates that a fragment of the material follows (a multiple "m" of 16K items). For all values of "m" the fragment is then followed by another length encoding for the remainder of the material.

3 (Tutorial) In the UNALIGNED variant, if the length count is bounded above by an upper bound that is less than 64K, then the constrained whole number encoding is used to encode the length in the minimum number of bits necessary to represent the range. Otherwise, the unconstrained length ("n" say) is encoded into a bit-field in the manner described above in Note 2.

10.9.1 This subclause is not invoked if, in accordance with the specification of later clauses, the value of the length determinant, "n", is fixed by the type definition (constrained by PER-visible constraints) to a value less than 64K.

10.9.2 This subclause is invoked for addition to the field-list of a field, or list of fields, preceded by a length determinant "n" which determines either:

a) the length in octets of an associated field (units are octets); or

b) the length in bits of an associated field (units are bits); or

c) the number of component encodings in an associated list of fields (units are components of a set-of or sequence-of); or

d) the number of characters in the value of an associated known-multiplier character string type (units are characters).

10.9.3 (ALIGNED variant) The procedures for the ALIGNED variant are specified in 10.9.3.1-10.9.3.8.4. (The procedures for the UNALIGNED variant are specified in 10.9.4.)

10.9.3.1 As a result of the analysis of the type definition (specified in later clauses) the length determinant (a whole number "n") will have been determined to be either:

a) a normally small length with a lower bound "lb" equal to one; or

b) a constrained whole number with a lower bound "lb" (greater than or equal to zero), and an upper bound "ub" less than 64K; or

c) a semi-constrained whole number with a lower bound "lb" (greater than or equal to zero), or a constrained whole number with a lower bound "lb" (greater than or equal to zero) and an upper bound "ub" greater than or equal to 64K.

10.9.3.2 The subclauses invoking the procedures of this subclause will have determined a value for "lb", the lower bound of the length (this is zero if the length is unconstrained), and for "ub", the upper bound of the length. "ub" is unset if there is no upper bound determinable from PER-visible constraints.

10.9.3.3 Where the length determinant is a constrained whole number with "ub" less than 64K, then the field-list shall have appended to it the encoding of the constrained whole number for the length determinant as specified in 10.5. If "n" is non-zero, this shall be followed by the associated field or list of fields, completing these procedures. If "n" is zero there shall be no further addition to the field-list, completing these procedures.

NOTES

1 For example:

A ::= Foo (SIZE (3..6)) -- *Length is encoded in 2-bit bit-field*

B ::= Foo (SIZE (40000..40254)) -- *Length is encoded in 8-bit bit-field*

C ::= Foo (SIZE (0..32000)) -- *Length is encoded in 2-octet octet-aligned bit-field*

D ::= Foo (SIZE (64000)) -- *Length is not encoded*

2 The effect of making no addition in the case of "n" equals zero is that padding to an octet boundary does not occur when these procedures are invoked to add an octet-aligned-bit-field of zero length, unless required by 10.5.

10.9.3.4 Where the length determinant is a normally small length and "n" is less than or equal to 64, a single-bit bit-field shall be appended to the field-list with the bit set to 0, and the value "n-1" shall be encoded as a non-negative-binary-integer into a 6-bit bit-field. This shall be followed by the associated field, completing these procedures. If "n" is greater than 64, a single-bit bit-field shall be appended to the field-list with the bit set to 1, followed by the encoding of "n" as an unconstrained length determinant followed by the associated field, according to the procedures of 10.9.3.5-10.9.3.8.4.

NOTE – Normally small lengths are only used to indicate the length of the bitmap that prefixes the extension addition values of a set or sequence type.

10.9.3.5 Otherwise (unconstrained, or large "ub"), "n" is encoded and appended to the field-list followed by the associated fields as specified below.

NOTE – The lower bound, "lb", does not affect the length encodings specified in 10.9.3.6-10.9.3.8.4.

10.9.3.6 If "n" is less than or equal to 127, then "n" shall be encoded as a non-negative-binary-integer (using the procedures of 10.3) into bits 7 (most significant) to 1 (least significant) of a single octet and bit 8 shall be set to zero. This shall be appended to the field-list as an octet-aligned-bit-field followed by the associated field or list of fields, completing these procedures.

NOTE – For example, if in the following a value of "A" is 4 characters long, and that of "B" is 4 items long,

A ::= IA5String

B ::= Foo (SIZE (4..123456))

both values are encoded with the length octet occupying one octet, and with the first bit set to 0 to indicate that the length is less than or equal to 127:

0 0000100	4 characters/items
length	value

10.9.3.7 If "n" is greater than 127 and less than 16K, then "n" shall be encoded as a non-negative-binary-integer (using the procedures of 10.3) into bit 6 of octet one (most significant) to bit 1 of octet two (least significant) of a two-octet octet-aligned-bit-field with bit 8 of the first octet set to 1 and bit 7 of the first octet set to zero. This shall be appended to the field-list followed by the associated field or list of fields, completing these procedures.

NOTE – If in the example of 10.9.3.6 a value of "A" is 130 characters long, and a value of "B" is 130 items long, both values are encoded with the length component occupying 2-octets, and with the first two bits set to 10 to indicate that the length is greater than 127 but less than 16K.

10 000000 10000010	130 characters/items
length	value

10.9.3.8 If "n" is greater than or equal to 16K, then there shall be appended to the field-list a single octet in an octet-aligned-bit-field with bit 8 set to 1 and bit 7 set to 1, and bits 6 to 1 encoding the value 1, 2, 3 or 4 as a non-negative-binary-integer (using the procedures of 10.8). This single octet shall be followed by part of the associated field or list of fields, as specified below.

NOTE – The value of bits 6 to 1 is restricted to 1-4 (instead of the theoretical limits of 0-63) so as to limit the number of items that an implementation has to have knowledge of to a more manageable number (64K instead of 1024K).

10.9.3.8.1 The value of bits 6 to 1 (1 to 4) shall be multiplied by 16K giving a count ("m" say). The choice of the integer in bits 6 to 1 shall be the maximum allowed value such that the associated field or list of fields contains more than or exactly "m" octets, bits, components or characters, as appropriate.

NOTES

1 The unfragmented form handles lengths up to 16K. The fragmentation therefore provides for lengths up to 64K with a granularity of 16K.

2 If in the example of 10.9.3.6 a value of "B" is 144K + 1 (i.e. 64K + 64K + 16K + 1) items long, the value is fragmented, with the first two bits of the first three fragments set to 11 to indicate that one to four blocks each of 16K items follow, and that another length component will follow the last block of each fragment:

11	000100	64K items	11	000100	64K items	11	000001	16K items	0	0000001	1 item
length		value	length		value	length		value	length		value

10.9.3.8.2 That part of the contents specified by "m" shall then be appended to the field-list as either:

a) a single octet-aligned-bit-field of "m" octets containing the first "m" octets of the associated field, for units which are octets; or

b) a single octet-aligned-bit-field of "m" bits containing the first "m" bits of the associated field, for units which are bits; or

c) the list of fields encoding the first "m" components in the associated list of fields, for units which are components of a set-of or sequence-of types; or

d) a single octet-aligned-bit-field of "m" characters containing the first "m" characters of the associated field, for units which are characters.

10.9.3.8.3 The procedures of 10.9 shall then be reapplied to add the remaining part of the associated field or list of fields to the field-list with a length which is a semi-constrained whole number equal to ("n" – "m") with a lower bound of zero.

NOTE – If the last fragment that contains part of the encoded value has a length that is an exact multiple of 16K it is followed by a final fragment that consists only of a single octet length component set to 0.

10.9.3.8.4 The addition of only a part of the associated field(s) to the field-list with reapplication of these procedures is called **the fragmentation procedure**.

10.9.4 (UNALIGNED variant) The procedures for the UNALIGNED variant are specified in 10.9.4.1 to 10.9.4.2 (The procedures for the ALIGNED variant are specified in 10.9.3):

10.9.4.1 If the length determinant "n" to be encoded is a constrained whole number with "range" ("ub"-"lb"+1) less than 64K, then "n" shall be encoded as a non-negative-binary-integer (as specified in 10.3) using the minimum number of bits necessary to encode the "range". If "n" is non-zero this shall be followed by an associated field or list of fields, completing these procedures. If "n" is zero there shall be no further addition to the field-list, completing these procedures.

NOTE – If "range" satisfies the inequality 2^m < "range" <= 2^{m+1}, then the number of bits in the length determinant is m + 1.

10.9.4.2 If the length determinant "n" to be encoded is a constrained whole number greater than or equal to 64K, or is a semi-constrained whole number, then "n" shall be encoded as specified in 10.9.3.4-10.9.3.8.4.

11 Encoding the boolean type

11.1 A value of the boolean type shall be encoded as a bit-field consisting of a single bit.

11.2 The bit shall be set to 1 for TRUE and 0 for FALSE.

11.3 The bit-field shall be appended to the field-list with no length determinant.

12 Encoding the integer type

NOTES

1 (Tutorial ALIGNED variant) Ranges which allow the encoding of all values into one octet or less go into a minimum-sized bit-field with no length count. Ranges which allow encoding of all values into two octets go into two octets in an octet-aligned-bit-field with no length count. Otherwise, the value is encoded into the minimum number of octets (using non-negative-binary-integer or 2's-complement-binary-integer encoding as appropriate) and a length determinant is added. In this case, if the integer value can be encoded in less than 127 octets (as an offset from any lower bound that might be determined), and there is no finite upper and lower bound, there is a one-octet length determinant, else the length is encoded in the fewest number of bits needed. Other cases are not of any practical interest, but are specified for completeness.

2 (Tutorial UNALIGNED variant) Constrained integers are encoded in the fewest number of bits necessary to represent the range regardless of its size. Unconstrained integers are encoded as in Note 1.

12.1 If an extension marker is present in the constraint specification of the integer type, then a single bit shall be added to the field-list in a bit-field of length one. The bit shall be set to 1 if the value to be encoded is not within the range of the extension root, and zero otherwise. In the former case, the value shall be added to the field-list as an unconstrained integer value, as specified in 12.2.4-12.2.6, completing this procedure. In the latter case, the value shall be encoded as if the extension marker is not present.

12.2 If an extension marker is not present in the constraint specification of the integer type, then the following applies:

12.2.1 If PER-visible constraints restrict the integer value to a single value then there shall be no addition to the field-list, completing these procedures.

12.2.2 If PER-visible constraints restrict the integer value to be a constrained whole number, then it shall be converted to a field according to the procedures of 10.5 (encoding of a constrained whole number), and the procedures of 12.2.5-12.2.6 shall then be applied.

12.2.3 If PER-visible constraints restrict the integer value to be a semi-constrained whole number, then it shall be converted to a field according to the procedures of 10.7 (encoding of a semi-constrained whole number), and the procedures of 12.2.6 shall then be applied.

12.2.4 If PER-visible constraints do not restrict the integer to be either a constrained or a semi-constrained whole number, then it shall be converted to a field according to the procedures of 10.8 (encoding of an unconstrained whole number), and the procedures of 12.2.6 shall then be applied.

12.2.5 If the procedures invoked to encode the integer value into a field did not produce the indefinite length case (see 10.5.7.4, and 10.8.2) then that field shall be appended to the field-list completing these procedures.

12.2.6 Otherwise, (the indefinite length case) the procedures of 10.9 shall be invoked to append the field to the field-list preceded by one of the following:

a) A constrained length determinant "len" (as determined by 10.5.7.4) if PER-visible constraints restrict the type with finite upper and lower bounds and, if the type is extensible, the value lies within the range of the extension root. The lower bound "lb" used in the length determinant shall be 1, and the upper bound "ub" shall be the count of the number of octets required to hold the range of the integer value.

NOTE – The encoding of the value "foo INTEGER (256..1234567) ::= 256" would thus be encoded as 00xxxxxx00000000, where each 'x' represents a zero pad bit that may or may not be present depending on where within the octet the length occurs (e.g. the encoding is 00 xxxxxx 00000000 if the length starts on an octet boundary, and 00 00000000 if it starts in the 2nd from the last bit of an octet).

b) An unconstrained length determinant equal to "len" (as determined by 10.7 and 10.8) if PER-visible constraints do not restrict the type with finite upper and lower bounds, or if the type is extensible and the value does not lie within the range of the extension root.

13 Encoding the enumerated type

NOTE – (Tutorial) An enumerated type without an extension marker is encoded as if it were a constrained integer whose subtype constraint does not contain an extension marker. This means that an enumerated type will almost always in practice be encoded as a bit-field in the smallest number of bits needed to express every enumeration. In the presence of an extension marker, it is encoded as a normally small non-negative whole number if the value is not in the extension root.

13.1 The enumerations in the enumeration root shall be sorted into ascending order by their enumeration value, and shall then be assigned an enumeration index starting with zero for the first enumeration, one for the second, and so on up to the last enumeration in the sorted list. The extension additions (which are always defined in ascending order) shall be assigned an enumeration index starting with zero for the first enumeration, one for the second, and so on up to the last enumeration in the extension additions.

NOTE – ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1 requires that each successive extension addition shall have a greater enumeration value than the last.

13.2 If the extension marker is absent in the definition of the enumerated type, then the enumeration index shall be encoded. Its encoding shall be as though it were a value of a constrained integer type for which there is no extension marker present, where the lower bound is 0 and the upper bound is the largest enumeration index associated with the type, completing this procedure.

13.3 If the extension marker is present, then a single bit shall be added to the field-list in a bit-field of length one. The bit shall be set to 1 if the value to be encoded is not within the extension root, and zero otherwise. In the former case, the enumeration additions shall be sorted according to and the value shall be added to the field-list as a normally small non-negative whole number whose value is the enumeration index of the additional enumeration and with "lb" set to 0, completing this procedure. In the latter case, the value shall be encoded as if the extension marker is not present, as specified in 13.2.

NOTE – There are no PER-visible constraints that can be applied to an enumerated type that are visible to these encoding rules.

14 Encoding the real type

NOTE – (Tutorial) A real uses the contents octets of CER/DER preceded by a length determinant that will in practice be a single octet.

14.1 If the base of the abstract value is 10, then the base of the encoded value shall be 10, and if the base of the abstract value is 2 the base of the encoded value shall be 2.

14.2 The encoding of REAL specified for the Canonical Encoding Rules and Distinguished Encoding Rules in ITU-T Rec. X.690 | ISO/IEC 8825-1 shall be applied to give an octet-aligned-bit-field which is the contents octets of the CER/DER encoding. The contents octets of this encoding consists of "n" (say) octets and is placed in an octet-aligned field of "n" octets. The procedures of 10.9 shall be invoked to append this octet-aligned-bit-field of "n" octets to the field-list, preceded by an unconstrained length determinant equal to "n".

15 Encoding the bitstring type

NOTE – (Tutorial) Bitstrings with a constrained length less than or equal to two octets do not cause octet alignment. Larger bitstrings are octet-aligned. If the length is fixed by constraints and the upper bound is less than 64K, there is no explicit length encoding, otherwise a length encoding is included which can take any of the forms specified earlier for length encodings, including fragmentation for large bit strings.

15.1 PER-visible constraints can only constrain the length of the bitstring.

15.2 Where there are no PER-visible constraints and 19.7 of ITU-T Rec. X.680 | ISO/IEC 8824-1 applies, the value shall be encoded with no trailing 0 bits (note that this means that a value with no 1 bits is always encoded as an empty bit string).

15.3 Where there is a PER-visible constraint and 19.7 of ITU-T Rec. X.680 | ISO/IEC 8824-1 applies (i.e. the bitstring type is defined with a "NamedBitList", the value shall be encoded with trailing 0 bits added or removed as necessary to ensure that the size of the transmitted value is the smallest size capable of carrying this value and satisfies the effective size constraint.

15.4 Let the maximum number of bits in the bitstring (as determined by PER-visible constraints on the length) be "ub" and the minimum number of bits be "lb". If there is no finite maximum we say that "ub" is unset. If there is no constraint on the minimum then "lb" has the value zero. Let the length of the actual bit string value to be encoded be "n" bits.

15.5 If an extension marker is present in the size constraint specification of the bitstring type, a single bit shall be added to the field-list in a bit-field of length one. The bit shall be set to 1 if the length of this encoding is not within the range of the extension root, and zero otherwise. In the former case, 15.10 shall be invoked to add the length as a semi-constrained whole number to the field-list, followed by the bitstring value. In the latter case the length and value shall be encoded as if the extension marker is not present.

15.6 If an extension marker is not present in the constraint specification of the bitstring type, then 15.7-15.10 apply.

15.7 If the bitstring is constrained to be of zero length ("ub" equals zero), then it shall not be encoded (no additions to the field-list), completing the procedures of this clause.

15.8 If all values of the bitstring are constrained to be of the same length ("ub" equals "lb") and that length is less than or equal to sixteen bits, then the bitstring shall be placed in a bit-field of the constrained length "ub" which shall be appended to the field-list with no length determinant, completing the procedures of this clause.

15.9 If all values of the bitstring are constrained to be of the same length ("ub" equals "lb") and that length is greater than sixteen bits but less than 64K bits, then the bitstring shall be placed in an octet-aligned-bit-field of length "ub" (which is not necessarily a multiple of eight bits) and shall be appended to the field-list with no length determinant, completing the procedures of this clause.

15.10 If 15.7-15.9 do not apply, the bitstring shall be placed in an octet-aligned-bit-field of length "n" bits and the procedures of 10.9 shall be invoked to add this octet-aligned-bit-field of "n" bits to the field-list, preceded by a length determinant equal to "n" bits as a constrained whole number if "ub" is set and is less than 64K or as a semi-constrained whole number if "ub" is unset. "lb" is as determined above.

NOTE – Fragmentation applies for unconstrained or large "ub" after 16K, 32K, 48K or 64K bits.

16 Encoding the octetstring type

NOTE – Octet strings of fixed length less than or equal to two octets are not octet-aligned. All other octet strings are octet-aligned. Fixed length octet strings encode with no length octets if they are shorter than 64K. For unconstrained octet strings the length is explicitly encoded (with fragmentation if necessary).

16.1 PER-visible constraints can only constrain the length of the octetstring.

16.2 Let the maximum number of octets in the octetstring (as determined by PER-visible constraints on the length) be "ub" and the minimum number of octets be "lb". If there is no finite maximum we say that "ub" is unset. If there is no constraint on the minimum then "lb" has the value zero. Let the length of the actual octetstring value to be encoded be "n" octets.

16.3 If there is a PER-visible size constraint and an extension marker is present in it, a single bit shall be added to the field-list in a bit-field of length one. The bit shall be set to 1 if the length of this encoding is not within the range of the extension root, and zero otherwise. In the former case 16.8 shall be invoked to add the length as a semi-constrained whole number to the field-list, followed by the octetstring value. In the latter case the length and value shall be encoded as if the extension marker is not present.

16.4 If an extension marker is not present in the constraint specification of the octetstring type, then 16.5-16.8 apply.

16.5 If the octetstring is constrained to be of zero length ("ub" equals zero), then it shall not be encoded (no additions to the field-list), completing the procedures of this clause.

16.6 If all values of the octetstring are constrained to be of the same length ("ub" equals "lb") and that length is less than or equal to two octets, the octetstring shall be placed in a bit-field with a number of bits equal to the constrained length "ub" multiplied by eight which shall be appended to the field-list with no length determinant, completing the procedures of this clause.

16.7 If all values of the octetstring are constrained to be of the same length ("ub" equals "lb") and that length is greater than two octets but less than 64K, then the octetstring shall be placed in an octet-aligned-bit-field with the constrained length "ub" octets which shall be appended to the field-list with no length determinant, completing the procedures of this clause.

16.8 If 16.5-16.7 do not apply, the octetstring shall be placed in an octet-aligned-bit-field of length "n" octets and the procedures of 10.9 shall be invoked to add this octet-aligned-bit-field of "n" octets to the field-list, preceded by a length determinant equal to "n" octets as a constrained whole number if "ub" is set, and as a semi-constrained whole number if "ub" is unset. "lb" is as determined above.

NOTE – The fragmentation procedures may apply after 16K, 32K, 48K, or 64K octets.

17 Encoding the null type

NOTE – (Tutorial) The null type is essentially a place holder, with practical meaning only in the case of a choice or an optional set or sequence component. Identification of the null in a choice, or its presence as an optional element, is performed in these encoding rules without the need to have octets representing the null. Null values therefore never contribute to the octets of an encoding.

There shall be no addition to the field-list for a null value.

18 Encoding the sequence type

NOTE – (Tutorial) A sequence type begins with a preamble which is a bit-map. If the sequence type has no extension marker, then the bit-map merely records the presence or absence of default and optional components in the type, encoded as a fixed length bit-field. If the sequence type does have an extension marker, then the bit-map is preceded by a single bit that says whether values of extension additions are actually present in the encoding. The preamble is encoded without any length determinant provided it is less than 64K bits long, otherwise a length determinant is encoded to obtain fragmentation. The preamble is followed by the fields that encode each of the components, taken in turn. If there are extension additions, then immediately before the first one is encoded there is the encoding (as a semi-constrained whole number) of a count of the number of extension additions in the type being encoded, followed by a bit-map equal in length to this count which records the presence or absence of values of each extension addition. This is followed by the encodings of the extension additions as if each one was the value of an open type field.

18.1 If the sequence type has an extension marker, then a single bit shall first be added to the field-list in a bit-field of length one. The bit shall be one if values of extension additions are present in this encoding, and zero otherwise. (This bit is called the "extension bit" in the following text.) If there is no extension marker, there shall be no extension bit added.

18.2 If the sequence type has "n" components in the extension root that are marked OPTIONAL or DEFAULT, then a single bit-field with "n" bits shall be produced for addition to the field-list. The bits of the bit-field shall, taken in order, encode the presence or absence of an encoding of each optional or default component in the sequence type. A bit value of 1 shall encode the presence of the encoding of the component, and a bit value of 0 shall encode the absence of the encoding of the component. The leading bit in the preamble shall encode the presence or absence of the first optional or default component, and the trailing bit shall encode the presence or absence of the last optional or default component.

18.3 If "n" is less than 64K, the bit-field shall be appended to the field-list. If "n" is greater than or equal to 64K, then the procedures of 10.9 shall be invoked to add this bit-field of "n" bits to the field-list, preceded by a length determinant equal to "n" bits as a constrained whole number with "ub" and "lb" both set to "n".

NOTE – In this case, "ub" and "lb" will be ignored by the length procedures. These procedures are invoked here in order to provide fragmentation of a large preamble. The situation is expected to arise only rarely.

18.4 The preamble shall be followed by the field-lists of each of the components of the sequence value which are present, taken in turn.

18.5 For CANONICAL-PER encodings of components marked DEFAULT shall always be absent if the value to be encoded is the default value. For BASIC-PER encodings of components marked DEFAULT shall always be absent if the value to be encoded is the default value of a simple type (see 3.8.25), otherwise it shall be encoded if and only if it is explicitly present in the abstract sequence value.

18.6 This completes the encoding if the extension bit is absent or is zero. If the extension bit is present and set to one, then the following procedures apply.

18.7 Let the number of extension additions in the type being encoded be "n", then a bit-field with "n" bits shall be produced for addition to the field-list. The bits of the bit-field shall, taken in order, encode the presence or absence of an encoding of each extension addition in the type being encoded. A bit value of 1 shall encode the presence of the encoding of the extension addition, and a bit value of 0 shall encode the absence of the encoding of the extension addition. The leading bit in the bit-field shall encode the presence or absence of the first extension addition, and the trailing bit shall encode the presence or absence of the last extension addition.

18.8 The procedures of 10.9 shall be invoked to add this bit-field of "n" bits to the field-list, preceded by a length determinant equal to "n" as a normally small length.

NOTE – "n" cannot be zero, as this procedure is only invoked if there is at least one extension addition being encoded.

18.9 This shall be followed by field-lists containing the encodings of each extension addition that is present, taken in turn, with each extension addition encoded as if it were the value of an open type field as specified in 10.2.1.

19 Encoding the sequence-of type

19.1 PER-visible constraints can constrain the number of components of the sequence-of type.

19.2 Let the maximum number of components in the sequence-of (as determined by PER-visible constraints) be "ub" components and the minimum number of components be "lb". If there is no finite maximum or "ub" is greater than or equal to 64K we say that "ub" is unset. If there is no constraint on the minimum then "lb" has the value zero. Let the number of components in the actual sequence-of value to be encoded be "n" components.

19.3 The encoding of each component of the sequence-of will generate a number of fields to be appended to the field-list for the sequence-of type.

19.4 If there is a PER-visible constraint and an extension marker is present in it, a single bit shall be added to the field-list in a bit-field of length one. The bit shall be set to 1 if the number of components in this encoding is not within the range of the extension root, and zero otherwise. In the former case clause 19.7 shall be invoked to add the length as a semi-constrained whole number to the field-list, followed by the component values. In the latter case the length and value shall be encoded as if the extension marker is not present.

19.5 If the number of components is fixed ("ub" equals "lb") and "ub" is less than 64K, then there shall be no length determinant for the sequence-of, and the fields of each component shall be appended in turn to the field-list of the sequence-of.

19.6 Otherwise, the procedures of 10.9 shall be invoked to add the list of fields generated by the "n" components to the field-list, preceded by a length determinant equal to "n" components as a constrained whole number if "ub" is set, and as a semi-constrained whole number if "ub" is unset. "lb" is as determined above.

NOTES

1 The fragmentation procedures may apply after 16K, 32K, 48K, or 64K components.

2 The break-points for fragmentation are between fields. The number of bits prior to a break-point are not necessarily a multiple of eight.

20 Encoding the set type

The set type shall have the elements in its "RootComponentTypeList" sorted into the canonical order specified in 6.4 of ITU-T Rec. X.680 | ISO/IEC 8824-1 and additionally for the purposes of determining the order in which components are encoded when one or more component is an untagged choice type, each untagged choice type is ordered as though it has a tag equal to that of the smallest tag in the "RootAlternativeTypeList" of that choice type or any untagged choice types nested within. The set elements that occur in the "AdditionalComponentTypeList" shall be sorted into the canonical order specified in 6.4 of ITU-T Rec. X.680 | ISO/IEC 8824-1 independent of how the NamedTypes in the RootComponentTypeList were sorted. The set value shall then be encoded as if it had been declared a sequence type.

EXAMPLE – In the following which assumes a tagging environment of IMPLICIT TAGS

```
A ::= SET
{
    a   [3] INTEGER,
    b   [1] CHOICE
    {
        c   [2] INTEGER,
        d   [4] INTEGER
    },
    e   CHOICE
    {
        f   CHOICE
        {
            g   [5] INTEGER,
            h   [6] INTEGER
        },
        i   CHOICE
        {
            j   [0] INTEGER
        }
    }
}
```

the order in which the components of the set are encoded will always be e, b, a, since the tag [0] sorts lowest, then [1], then [3].

NOTE – ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1 requires that each successive extension addition shall have a greater tag value than the last added to the "AdditionalComponentTypeList".

21 Encoding the set-of type

21.1 For CANONICAL-PER the encoding of the component values of the set-of type shall appear in ascending order, the component encodings being compared as bit strings padded with as many as seven 0 bits to an octet boundary.

NOTE – Any pad bits added to achieve octet alignment for the sort do not appear in the actual encoding.

21.2 For BASIC-PER the set-of shall be encoded as if it had been declared a sequence-of type.

22 Encoding the choice type

NOTE – (Tutorial) A choice type is encoded by encoding an index specifying the chosen alternative. This is encoded as for a constrained integer (unless the extension marker is present in the choice type, in which case it is a normally small non-negative whole number) and would therefore typically occupy a fixed length bit-field of the minimum number of bits needed to encode the index. (Although it could in principle be arbitrarily large.) This is followed by the encoding of the chosen alternative, with alternatives that are extension additions encoded as if they were the value of an open type field. Where the choice has only one alternative, there is no encoding for the index.

22.1 Encoding of choice types are not affected by PER-visible constraints.

22.2 Each component of a choice has an index associated with it which has the value zero for the first alternative in the root of the choice (taking the alternatives in the canonical order specified in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 6.4), one for the second, and so on up to the last component in the extension root of the choice. An index value is similarly assigned to each extension addition, starting with 0 just as with the components of the extension root. Let "n" be the value of the largest index in the root.

NOTE – ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1 requires that each successive extension addition shall have a greater tag value than the last added to the "AdditionalAlternativeTypeList".

22.3 For the purposes of canonical ordering of choice alternatives that contain an untagged choice each untagged choice type shall be ordered as though it has a tag equal to that of the smallest tag in the extension root of either that choice type or any untagged choice types nested within.

22.4 If the choice has only one alternative in the extension root there shall be no encoding for the index if that alternative is chosen.

22.5 If the choice type has an extension marker, then a single bit shall first be added to the field-list in a bit-field of length one. The bit shall be 1 if a value of an extension addition is present in the encoding, and zero otherwise. (This bit is called the "extension bit" in the following text.) If there is no extension marker, there shall be no extension bit added.

22.6 If the extension bit is absent, then the choice index of the chosen alternative shall be encoded into a field according to the procedures of clause 12 as if it were a value of an integer type (with no extension marker in its subtype constraint) constrained to the range 0 to "n", and that field shall be appended to the field-list. This shall then be followed by the fields of the chosen alternative, completing the procedures of this clause.

22.7 If the extension bit is present and the chosen alternative lies within the extension root, the choice index of the chosen alternative shall be encoded as if the extension marker is absent, according to the procedure of , completing the procedures of this clause.

22.8 If the extension bit is present and the chosen alternative does not lie within the extension root, the choice index of the chosen alternative shall be encoded as a normally small non-negative whole number with "lb" set to 0 and that field shall be appended to the field-list. This shall then be followed by a field-list containing the encoding of the chosen alternative encoded as if it were the value of an open type field as specified in 10.2, completing the procedures of this clause.

23 Encoding the object identifier type

NOTE – (Tutorial) An object identifier type encoding uses the contents octets of BER preceded by a length determinant that will in practice be a single octet.

The encoding specified for BER shall be applied to give an octet-aligned-bit-field which is the contents octets of the BER encoding. The contents octets of this BER encoding consists of "n" (say) octets and is placed in an octet-aligned-bit-field of "n" octets. The procedures of 10.9 shall be invoked to append this octet-aligned-bit-field to the field-list, preceded by a length determinant equal to "n" as a semi-constrained whole number octet count.

24 Encoding the embedded-pdv type

NOTE – (Tutorial) The embedded-pdv type is typically encoded with a single flag bit at its head, followed by a normally small non-negative whole number that encodes an "index", optionally followed by the "identification" and finally followed by the "data-value". The single flag bit determines which of two sub-rules is in use in the remainder of the embedded-pdv value. This bit is set to 1 to indicate that the "index-setting" rule EP-A is in use, and it is set to 0 to indicate that the "index-using" sub-rule is in use. The "index-setting" sub-rule is used upon the first occurrence of a given "identification" in the abstract syntax value. For all subsequent occurrences of that "identification" value within the abstract syntax value in which the embedded-pdv type is defined the "index-using" sub-rule is used. When the "index-setting" sub-rule is used the full "identification" value is encoded. An "index" value which appears with the flag set to 0 in an embedded-pdv encoding is required to appear precisely once as an "index" value with the flag set to 1. For CANONICAL-PER, only the "syntaxes" alternative of "identification" is allowed (explicitly or via ASN.1 constraints on the embedded-pdv type). There is one additional optimization: if both the abstract and transfer syntaxes are constrained by PER-visible constraints to be a single value, or if "identification" is constrained to the "fixed" alternative, then there is no single flag bit, "index" nor "identification" present in the encoding.

24.1 There are three ways in which an embedded-pdv type can be encoded:

a) the "syntaxes" alternative of the embedded-pdv type is constrained to a single value or "identification" is constrained to the "fixed" alternative, in which case only the data-value shall be encoded; this alternative is called the "predefined" option;

b) the first bit of the embedded-pdv value is set to 1, in which case an "index" value shall be prefixed to the "identification" which shall be present, followed by the "data-value"; this alternative is called the "index-setting" sub-rule, or EP-A;

c) the first bit of the embedded-pdv value is set to 0, in which case an "index" value and "data-value" shall be present but the "identification" shall be absent; this alternative is called the "index-using" sub-rule, or EP-B.

24.2 The "predefined" option shall be used when either "identification" is constrained to the "fixed" alternative or when the "syntaxes" alternative of "identification" is constrained to a single value. When the "predefined" option is used the complete encoding of the "data-value" shall be added to an octet-aligned-bit-field and 10.9.3.5 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this clause.

24.3 When the "predefined" option is not selected a single bit shall be added to the field-list in a bit-field of length 1 and set to 1. This bit shall be set to 1 if the "index-setting" sub-rule is being employed, otherwise it shall be set to 0.

24.4 The "index-setting" sub-rule shall be used the first time that a given abstract value for "identification" is specified. When the "index-setting" sub-rule is used, 10.6 is invoked to add an index value to the field-list as a normally small non-negative whole number. The first index value shall be 0, and it shall be incremented by 1 for each distinct "identification" value that is encountered. Clause 22 shall then be invoked to add the "identification" to the field-list. The complete encoding of the "data-value" shall then be added to an octet-aligned-bit-field and 10.9 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this clause.

24.5 The "index-using" sub-rule shall be used on all subsequent occurrences of a given abstract value for "identification". When the "index-using" sub-rule is used, 10.6 is invoked to add an index value to the field-list as a normally small non-negative whole number. This index value shall correspond to the one which was previously added to the field-list for this given "identification" abstract value. The complete encoding of the "data-value" shall then be added to an octet-aligned-bit-field and 10.9 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this clause.

24.6 TUTORIAL: Thus for any given "identification" value there is a (relatively inefficient) EP-A encoding with both a unique index value and the full "identification" value, followed by arbitrarily many (efficient) EP-B encodings linked to the EP-A encoding by the index value.

24.7 The EP-A encoding shall be the PER encoding of the single index-setting/index-using bit, followed by the index, followed by the following sequence type for which AUTOMATIC TAGS is specified (see 22.6 and 26.3 of ITU-T Rec. X.680 | ISO/IEC 8824-1):

```
SEQUENCE {
    identification          CHOICE {
        syntaxes                SEQUENCE {
            abstract                OBJECT IDENTIFIER,
            transfer                OBJECT IDENTIFIER },
        syntax                  OBJECT IDENTIFIER,
```

```
        presentation-context-id      INTEGER,
        context-negotiation          SEQUENCE {
            presentation-context-id      INTEGER,
            transfer-syntax              OBJECT IDENTIFIER },
        transfer-syntax              OBJECT IDENTIFIER,
        fixed                        NULL },
    data-value                   BIT STRING }
```

24.8 The value of "data-value" shall be the encoding of the abstract data value using the identified transfer syntax, and the value of all other fields shall be encoded using the same transfer syntax as the other values appearing in the abstract syntax value.

NOTE – Both alternatives of "data-value" in the abstract syntax are encoded identically – as a bit string – in the transfer syntax.

25 Encoding of a value of the external type

25.1 The encoding of a value of the external type shall be the PER encoding of the following sequence type, assumed to be defined in an environment of EXPLICIT TAGS, with a value as specified in the subclauses below:

```
[UNIVERSAL 8] IMPLICIT SEQUENCE {
    direct-reference         OBJECT IDENTIFIER OPTIONAL,
    indirect-reference       INTEGER OPTIONAL,
    data-value-descriptor    ObjectDescriptor OPTIONAL,
    encoding                 CHOICE {
        single-ASN1-type             [0] ABSTRACT-SYNTAX.&Type,
        octet-aligned                [1] IMPLICIT OCTET STRING,
        arbitrary                    [2] IMPLICIT BIT STRING } }
```

NOTE – This sequence type is the same as that specified in CCITT Rec. X.208 (1988) | ISO/IEC 8824 (1990).

25.2 The value of the components depends on the abstract value being transmitted, which is a value of the type specified in 30.5 of ITU-T Rec. X.680 | ISO/IEC 8824-1.

25.3 The "data-value-descriptor" above shall be present if and only if the "data-value-descriptor" is present in the abstract value, and shall have the same value.

25.4 Values of "direct-reference" and "indirect-reference" above shall be present or absent in accordance with Table 1. Table 1 maps the external type alternatives of "identification" defined in ITU-T Rec. X.680 | ISO/IEC 8824-1, clause 30.5, to the external type components "direct-reference" and "indirect-reference" defined in 8.18.1.

Table 1 – Alternative encodings for "identification"

identification	direct-reference	indirect-reference
syntaxes	*** CANNOT OCCUR ***	*** CANNOT OCCUR ***
syntax	syntax	ABSENT
presentation-context-id	ABSENT	presentation-context-id
context-negotiation	transfer-syntax	presentation-context-id
transfer-syntax	*** CANNOT OCCUR ***	*** CANNOT OCCUR ***
fixed	*** CANNOT OCCUR ***	*** CANNOT OCCUR ***

25.5 The data value shall be encoded according to the transfer syntax identified by the encoding, and shall be placed in an alternative of the "encoding" choice as specified below.

25.6 If the data value is the value of a single ASN.1 data type, and if the encoding rules for this data value are the same as those for the complete "EXTERNAL" data type, then the sending implementation shall be "single-ASN1-type".

25.7 If the encoding of the data value, using the agreed or negotiated encoding, is an integral number of octets, then the sending implementation shall be "octet-aligned".

NOTE – A data value which is a series of ASN.1 types, and for which the transfer syntax specifies simple concatenation of the octet strings produced by applying the ASN.1 Basic Encoding Rules to each ASN.1 type, falls into this category, not that of 25.6.

25.8 If the encoding of the data value, using the agreed or negotiated encoding, is not an integral number of octets, the "Encoding" choice shall be "arbitrary".

25.9 If the "Encoding" choice is chosen as "single-ASN1-type", then the ASN.1 type shall replace the open type, with a value equal to the data value to be encoded.

NOTE – The range of values which might occur in the open type is determined by the registration of the object identifier value associated with the "direct-reference", and/or the integer value associated with the "indirect-reference".

25.10 If the "Encoding" choice is "octet-aligned", then the data value shall be encoded according to the agreed or negotiated transfer syntax, and the resulting octets shall form the value of the octetstring.

25.11 If the "Encoding" choice is "arbitrary", then the data value shall be encoded according to the agreed or negotiated transfer syntax, and the result shall form the value of the bitstring.

26 Encoding the restricted character string types

NOTES

1 (Tutorial ALIGNED variant) Character strings of fixed length less than or equal to two octets are not octet-aligned. Character strings of variable length that are constrained to have a maximum length of less than two octets are not octet-aligned. All other character strings are octet-aligned. Fixed length character strings encode with no length octets if they are shorter than 64K characters. For unconstrained character strings or constrained character strings longer than 64K-1 the length is explicitly encoded (with fragmentation if necessary). Each NumericString, PrintableString, VisibleString (ISO646String), IA5String, BMPString and UniversalString character is encoded into the number of bits that is the smallest power of two that can accommodate all characters allowed by the effective PermittedAlphabet constraint.

2 (Tutorial UNALIGNED variant) Character strings are not octet-aligned. If there is only one possible length value there is no length encoding if they are shorter than 64K characters. For unconstrained character strings or constrained character strings longer than 64K-1 the length is explicitly encoded (with fragmentation if necessary). Each NumericString, PrintableString, VisibleString (ISO646String), IA5String, BMPString and UniversalString character is encoded into the number of bits that is the smallest that can accommodate all characters allowed by the effective PermittedAlphabet constraint.

3 (Tutorial on size of each encoded character) Encoding of each character depends on the effective PermittedAlphabet constraint (see 9.3.10), which defines the alphabet in use for the type. Suppose this alphabet consists of a set of characters ALPHA (say). For each of the known-multiplier character string types (see 3.8.16), there is an integer value associated with each character, obtained by reference to some code table associated with the restricted character string type. The set of values BETA (say) corresponding to the set of characters ALPHA is used to determine the encoding to be used, as follows: The number of bits for the encoding of each character is determined solely by the number of elements, N, in the set BETA (or ALPHA). For the UNALIGNED variant is the smallest number of bits that can encode the value N-1 as a non-negative binary integer. For the ALIGNED variant this is the smallest number of bits that is a power of two and that can encode the value N-1. Suppose the selected number of bits is B. Then if every value in the set BETA can be encoded (with no transformation) in B bits, then the value in set BETA is used to represent the corresponding characters in the set ALPHA. Otherwise, the values in set BETA are taken in ascending order and replaced by values 0, 1, 2, and so on up to N-1, and it is these values that are used to represent the corresponding character. In summary: minimum bits (taken to the next power of two for the ALIGNED variant) are always used. Preference is then given to using the value normally associated with the character, but if any of these values cannot be encoded in the minimum number of bits a compaction is applied.

26.1 The following restricted character string types are known-multiplier character string types: NumericString, PrintableString, VisibleString (ISO646String), IA5String, BMPString, UniversalString. Effective PermittedAlphabet constraints are PER-visible only for these types.

26.2 The effective size constraint notation may determine an upper bound "aub" for the length of the abstract character string. Otherwise, "aub" is unset.

26.3 The effective size constraint notation may determine a non-zero lower bound "alb" for the length of the abstract character string. Otherwise, "alb" is zero.

NOTE – PER-visible constraints only apply to known-multiplier character string types. For other restricted character string types "aub" will be unset and "alb" will be zero.

26.4 If the type is extensible for PER encodings (see 9.3.15), then a bit-field consisting of a single bit shall be added to the field-list. The single bit shall be set to zero if the value is within the range of the extension root, and to one otherwise. If the value is outside the range of the extension root, then the following encoding shall be as if there was neither an effective size constraint nor an effective PermittedAlphabet constraint.

NOTE – Only the known-multiplier character string types can be extensible for PER encodings. Extensibility markers on other character string types do not affect the PER encoding.

26.5 This subclause applies to known-multiplier character strings. Encoding of the other restricted character string types is specified in 26.6.

26.5.1 The effective permitted alphabet is defined to be that alphabet permitted by the PermittedAlphabet constraint, or the entire alphabet of the built-in type if there is no PermittedAlphabet constraint.

26.5.2 Let N be the number of characters in the effective permitted alphabet. Let B be the smallest integer such that 2 to the power B is greater than or equal to N. Let B2 be the smallest power of 2 that is greater than or equal to B. Then in the ALIGNED variant, each character shall encode into B2 bits, and in the UNALIGNED variant into B bits. Let the number of bits identified by this rule be "b".

26.5.3 A numerical value "v" is associated with each character by reference to clause 36 of ITU-T Rec. X.680 | ISO/IEC 8824-1 as follows. For UniversalString, the value is that used to determine the canonical order in 36.4 (The value is in the range 0 to $2^{32} – 1$). For BMPString, the value is that used to determine the canonical order in 36.5 (The value is in the range 0 to $2^{16} – 1$). For NumericString and PrintableString and VisibleString and IA5String the value is that defined for the ISO 646 encoding of the corresponding character. (For IA5String the range is 0 to 127, for VisibleString it is 32 to 126, for NumericString it is 32 to 57, and for PrintableString it is 32 to 122. For IA5String and VisibleString all values in the range are present, but for NumericString and PrintableString not all values in the range are in use.)

26.5.4 Let the smallest value in the range for the set of characters in the permitted alphabet be "lb" and the largest value be "ub". Then the encoding of a character into "b" bits is the non-negative-binary-integer encoding of the value "v" identified as follows:

a) if "ub" is less than or equal to $2^b – 1$, then "v" is the value specified in above; otherwise

b) the characters are placed in the canonical order defined in clause 36 of ITU-T Rec. X.680 | ISO/IEC 8824-1. The first is assigned the value zero and the next in canonical order is assigned a value that is one greater than the value assigned to the previous character in the canonical order. These are the values "v".

NOTE – Item a) above can never apply to a constrained or unconstrained NumericString character, which always encodes into four bits or less using b).

26.5.5 The encoding of the entire character string shall be obtained by encoding each character (using an appropriate value "v") as a non-negative-binary-integer into "b" bits which shall be concatenated to form a bit-field that is a multiple of "b" bits.

26.5.6 If "aub" equals "alb" and is less than 64K, then the bit-field shall be added to the field-list as an octet-aligned field if "aub" times "b" is greater than 16, but shall otherwise be added as a bit-field that is not octet-aligned. This completes the procedures of this subclause.

26.5.7 If "aub" does not equal "alb" or is greater than or equal to 64K, then 10.9 shall be invoked to add a length determinant with "n" as a count of the characters in the character string with a lower bound for the length determinant of "alb" and an upper bound of "aub". The bit-field of shall then be added as an octet-aligned field if "aub" times "b" is greater than or equal to 16, but shall otherwise be added as a bit-field that is not octet-aligned. This completes the procedures of this subclause.

26.6 This subclause applies to character strings that are not known-multiplier character strings. In this case, constraints are never PER-visible, and the type can never be extensible for PER encoding.

26.6.1 For BASIC-PER, reference below to "base encoding" means BER. For CANONICAL-PER it means CANONICAL-BER.

26.6.2 The "base encoding" shall be applied to the character string to give a field of "n" octets.

26.6.3 Subclause 10.9 shall be invoked to add an unconstrained length determinant with "n" as a count in octets and the field of "n" octets shall be added as an octet-aligned bit-field, completing the procedures of this subclause.

27 Encoding the unrestricted character string type

NOTE – (Tutorial) The unrestricted character string type is typically encoded with a single flag bit at its head, followed by a normally small non-negative whole number that encodes an "index", optionally followed by the "identification" and finally followed by the "string-value". The single flag bit determines which of two sub-rules is in use in the remainder of the unrestricted character string value. This bit is set to 1 to indicate that the "index-setting" rule EP-A is in use, and it is set to 0 to indicate that the "index-using" sub-rule is in use. The "index-setting" sub-rule is used upon the first occurrence of a given "identification" in the abstract syntax value. For all subsequent occurrences of that "identification" value within the abstract syntax value in which the unrestricted character string type is defined the "index-using" sub-rule is used. When the "index-setting" sub-rule is used the full "identification" value is encoded. An "index" value which appears with the flag set to 0 in an unrestricted character string encoding is required to appear precisely once as an "index" value with the flag set to 1. For CANONICAL-PER, only the "syntaxes" alternative of "identification" is allowed (explicitly or via ASN.1 constraints on the unrestricted character string type). There is one additional optimization: if both the abstract and transfer syntaxes are constrained by PER-visible constraints to be a single value, or if "identification" is constrained to the "fixed" alternative, then there is no single flag bit, "index" nor "identification" present in the encoding.

27.1 There are three ways in which an unrestricted character string type can be encoded:

a) the "syntaxes" alternative of the unrestricted character string type is constrained to a single value or "identification" is constrained to the "fixed" alternative, in which case only the "string-value" shall be encoded; this alternative is called the "predefined" option;

b) the first bit of the unrestricted character string value is set to 1, in which case an "index" value shall be prefixed to the "identification" which shall be present, followed by the "string-value"; this alternative is called the "index-setting" sub-rule, or EP-A;

c) the first bit of the unrestricted character string value is set to 0, in which case an "index" value and "string-value" shall be present but the "identification" shall be absent; this alternative is called the "index-using" sub-rule, or EP-B.

27.2 The "predefined" option shall be used when either "identification" is constrained to the "fixed" alternative or when the "syntaxes" alternative of "identification" is constrained to a single value. When the "predefined" option is used the complete encoding of the "string-value" shall be added to an octet-aligned-bit-field and 10.9.3.5 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this subclause.

27.3 When the "predefined" option is not selected a single bit shall be added to the field-list in a bit-field of length 1. This bit shall be set to 1 if the "index-setting" sub-rule is being employed, otherwise it shall be set to 0.

27.4 The "index-setting" sub-rule shall be used the first time that a given abstract value for "identification" is specified. When the "index-setting" sub-rule is used 10.6 is invoked to add an index value to the field-list as a normally small non-negative whole number. The first index value shall be 0, and it shall be incremented by 1 for each distinct "identification" value that is encountered. Clause 22 shall then be invoked to add the "identification" to the field-list. The complete encoding of the "string-value" shall then be added to an octet-aligned-bit-field and 10.9 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this subclause.

27.5 The "index-using" sub-rule shall be used on all subsequent occurrences of a given abstract value for "identification". When the "index-using" sub-rule is used 10.6 is invoked to add an index value to the field-list as a normally small non-negative whole number. This index value shall correspond to the one which was previously added to the field-list for this given "identification" abstract value. The complete encoding of the "string-value" shall then be added to an octet-aligned-bit-field and 10.9 shall be invoked to append it to the field-list, prefixed with the length as a semi-constrained whole number. This completes the procedures for this subclause.

27.6 TUTORIAL: Thus for any given "identification" value there is a (relatively inefficient) EP-A encoding with both a unique index value and the full "identification" value, followed by arbitrarily many (efficient) EP-B encodings linked to the EP-A encoding by the index value.

27.7 The EP-A encoding shall be the PER encoding of the single index-setting/index-using bit, followed by the index, followed by the following sequence type for which AUTOMATIC TAGS is specified (see 22.6 and 26.3 of ITU-T Rec. X.680 | ISO/IEC 8824-1):

```
SEQUENCE {
    identification              CHOICE {
        syntaxes                    SEQUENCE {
            abstract                    OBJECT IDENTIFIER,
            transfer                    OBJECT IDENTIFIER },
        syntax                      OBJECT IDENTIFIER,
        presentation-context-id     INTEGER,
        context-negotiation         SEQUENCE {
            presentation-context-id     INTEGER,
            transfer-syntax             OBJECT IDENTIFIER },
        transfer-syntax             OBJECT IDENTIFIER,
        fixed                       NULL },
    string-value                OCTET STRING }
```

27.8 The value of "string-value" shall be the encoding of the abstract data value using the identified transfer syntax, and the value of all other fields shall be encoded using the same transfer syntax as the other values appearing in the abstract syntax value.

NOTE – Both alternatives of "string-value" in the abstract syntax are encoded identically – as an octet string – in the transfer syntax.

28 Object identifiers for transfer syntaxes

28.1 The encoding rules specified in this Recommendation | International Standard can be referenced and applied whenever there is a need to specify an unambiguous bit string representation for all of the values of a single ASN.1 type.

28.2 The following object identifiers and object descriptor values are assigned to identify and describe the encoding rules specified in this Recommendation | International Standard:

For BASIC-PER, ALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) aligned (0)}

"Packed encoding of a single ASN.1 type (basic aligned)"

For BASIC-PER, UNALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) unaligned (1)}

"Packed encoding of a single ASN.1 type (basic unaligned)"

For CANONICAL-PER, ALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) aligned (0)}

"Packed encoding of a single ASN.1 type (canonical aligned)"

For CANONICAL-PER, UNALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) unaligned (1)}

"Packed encoding of a single ASN.1 type (canonical unaligned)"

28.3 Where an application standard defines an abstract syntax as a set of abstract values, each of which is a value of some specifically named ASN.1 type defined using the ASN.1 notation, then the object identifier values specified in 28.2 may be used with the abstract syntax name to identify those transfer syntaxes which result from the application of the encoding rules specified in this Recommendation | International Standard to the specifically named ASN.1 type used in defining the abstract syntax.

NOTE – In particular, these identifications of the encoding rules can appear in the "transfer syntax name" field of the presentation protocol (ITU-T Rec. X.226 | ISO/IEC 8823-1).

28.4 The names specified in 28.2 shall not be used with an abstract syntax name to identify a transfer syntax if the conditions of 28.3 for the definition of the abstract syntax are not met.

Annex A

Example of encodings

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

This annex illustrates use of the Packed Encoding Rules specified in this Recommendation | International Standard by showing representations in octets of a (hypothetical) personnel record which is defined using ASN.1.

A.1 Record that does not use subtype constraints

A.1.1 ASN.1 description of the record structure

The structure of the hypothetical personnel record is formally described below using ASN.1 specified in ITU-T Rec. X.680 | ISO/IEC 8824-1 for defining types. This is identical to the example defined in Annex A of ITU-T Rec. X.690 | ISO/IEC 8825-1.

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        name            Name,
        title           [0] VisibleString,
        number          EmployeeNumber,
        dateOfHire      [1] Date,
        nameOfSpouse    [2] Name,
        children        [3] IMPLICIT
            SEQUENCE OF ChildInformation DEFAULT {} }

ChildInformation ::= SET
        { name          Name,
          dateOfBirth   [0] Date}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE
    {givenName          VisibleString,
     initial            VisibleString,
     familyName         VisibleString}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString  -- YYYYMMDD
```

A.1.2 ASN.1 description of a record value

The value of John Smith's personnel record is formally described below using ASN.1.

```
{ name {givenName "John",initial "P",familyName "Smith"},
  title         "Director",
  number        51,
  dateOfHire        "19710917",
  nameOfSpouse      {givenName "Mary",initial "T",familyName "Smith"},
  children
     {{name {givenName "Ralph",initial "T",familyName "Smith"},
              dateOfBirth "19571111"},
     {name {givenName "Susan",initial "B",familyName "Jones"},
              dateOfBirth "19590717"}}}
```

A.1.3 ALIGNED PER representation of this record value

The representation of the record value given above (after applying the ALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary.

The length of this encoding is 94 octets. For comparison, the same PersonnelRecord value encoded using the UNALIGNED variant of PER is 84 octets, BER with the definite length form is at least 136 octets, and BER with the indefinite length form is at least 161 octets.

A.1.3.1 Hexadecimal view

```
80044A6F 686E0150 05536D69 74680133 08446972 6563746F 72083139 37313039
3137044D 61727901 5405536D 69746802 0552616C 70680154 05536D69 74680831
39353731 31313105 53757361 6E014205 4A6F6E65 73083139 35393037 3137
```

A.1.3.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; and an ’x’ represents a zero pad bit that is used from time to time to align fields on an octet boundary.

1xxxxxxx	Bitmap bit = 1 indicates "children" is present
00000100	Length of name.givenName = 4
01001010 01101111 01101000 01101110	name.givenName = "John"
00000001	Length of name.initial = 1
01010000	name.initial = "P"
00000101	Length of name.familyName = 5
01010011 01101101 01101001 01110100 01101000	name.familyName = "Smith"
00000001	Length of (employee) number = 1
00110011	(employee) number = 51
00001000	Length of title = 8
01000100 01101001 01110010 01100101 01100011 01110100 01101111 01110010	title = "Director"
00001000	Length of dateOfHire = 8
00110001 00111001 00110111 00110001 00110000 00111001 00110001 00111111	dateOfHire = "19590717"
00000100	Length of nameOfSpouse.givenName = 4
01001101 01100001 01110010 01111001	nameOfSpouse.givenName = "Mary"
00000001	Length of nameOfSpouse.initial = 1
01010100	nameOfSpouse.initial = "T"
00000101	Length of nameOfSpouse.familyName = 5
01010011 01101101 01101001 01110100 01101000	nameOfSpouse.familyName = "Smith"
00000010	Number of children
00000101	Length of children[0].givenName = 5
01010010 01100001 01101100 01110000 01101000	children[0].givenName = "Ralph"
00000001	Length of children[0].initial = 1
01010100	children[0].initial = "T"
00000101	Length of children[0].familyName = 5
01010011 01101101 01101001 01110100 01101000	children[0].familyName = "Smith"
00001000	Length of children[0].dateOfBirth = 8
00110001 00111001 00110101 00110111 00110001 00110001 00110001 00110001	children[0].dateOfBirth = "19571111"
00000101	Length of children[1].givenName = 5
01010011 01110101 01110011 01100001 01101110	children[1].givenName = "Susan"
00000001	Length of children[1].initial = 1
01000010	children[1].initial = "B"
00000101	Length of children[1].familyName = 5
01001010 01101111 01101110 01100101 01110011	children[1].familyName = "Jones"
00001000	Length of children[1].dateOfBirth = 8
00110001 00111001 00110101 00111001 00110000 00110111 00110001 00110111	children[1].dateOfBirth = "19590717"

A.1.4 UNALIGNED PER representation of this record value

The representation of the record value given above (after applying the UNALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary. Note that pad bits do not occur in the UNALIGNED variant, and characters are encoded in the fewest number of bits possible.

The length of this encoding is 84 octets. For comparison, the same PersonnelRecord value encoded using the ALIGNED variant of PER is 94 octets, BER with the definite length form is at least 136 octets, and BER with the indefinite length form is at least 161 octets.

A.1.4.1 Hexadecimal view

```
824ADFA3 700D005A 7B74F4D0 02661113 4F2CB8FA 6FE410C5 CB762C1C B16E0937
0F2F2035 0169EDD3 D340102D 2C3B3868 01A80B4F 6E9E9A02 18B96ADD 8B162C41
69F5E787 700C2059 5BF765E6 10C5CB57 2C1BB16E
```

A.1.4.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; a period (.) is used to mark octet boundaries; and an 'x' represents a zero-bit used to pad the final octet to an octet boundary.

```
1                                                                   Bitmap bit = 1 indicates "children" is present

0000010.0                                                           Length of name.givenName = 4
1001010 .1101111  1.101000  11.01110                                name.givenName = "John"

000.00001                                                           Length of name.initial = 1
101.0000                                                            name.initial = "P"

0000.0101                                                           Length of name.familyName = 5
1010.011  11011.01  110100.1  1110100  .1101000                     name.familyName = "Smith"

0.0000001                                                           Length of (employee) number = 1
0.0110011                                                           (employee) number = 51

0.0001000                                                           Length of title = 8
1.000100  11.01001  111.0010  1100.101  11000.11  111010.0  1101111 .1110010    title = "Director"

0.0001000                                                           Length of dateOfHire = 8
0.110001  01.11001  011.0111  0110.001  01100.00  011100.1  0110001  .0111111   dateOfHire = "19590717"

0.0000100                                                           Length of nameOfSpouse.givenName = 4
1.001101  11.00001  111.0010  1111.001                              nameOfSpouse.givenName = "Mary"

00000.001                                                           Length of nameOfSpouse.initial = 1
10101.00                                                            nameOfSpouse.initial = "T"

000001.01                                                           Length of nameOfSpouse.familyName = 5
101001.1  1101101  .1101001  1.110100  11.01000                     nameOfSpouse.familyName = "Smith"

000.00010                                                           Number of children

000.00101                                                           Length of children[0].givenName = 5
101.0010  1100.001  11011.00  111000.0  1101000                     children[0].givenName = "Ralph"

.00000001                                                           Length of children[0].initial = 1
.1010100                                                            children[0].initial = "T"

0.0000101                                                           Length of children[0].familyName = 5
1.010011  11.01101  110.1001  1110.100  11010.00                    children[0].familyName = "Smith"

000010.00                                                           Length of children[0].dateOfBirth = 8
011000.1  0111001  .0110101  0.110111  01.10001  011.0001  0110.001  01100.01   children[0].dateOfBirth = "19571111"

000001.01                                                           Length of children[1].givenName = 5
101001.1  1110101  .1110011  1.100001  11.01110                     children[1].givenName = "Susan"

000.00001                                                           Length of children[1].initial = 1
100.0010                                                            children[1].initial = "B"

0000.0101                                                           Length of children[1].familyName = 5
1001.100  11011.11  110111.0  1100101  .1110011                     children[1].familyName = "Jones"

0.0001000                                                           Length of children[1].dateOfBirth = 8
0.110001  01.11001  011.0101  0111.001  01100.00  011011.1  0110001  .0110111x  children[1].dateOfBirth = "19590717"
```

A.2 Record that uses subtype constraints

This example is the same as that shown in clause A.1, except that it makes use of the subtype notation to impose constraints on some items.

A.2.1 ASN.1 description of the record structure

The structure of the hypothetical personnel record is formally described below using ASN.1 specified in ITU-T Rec. X.680 | ISO/IEC 8824-1 for defining types.

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        name            Name,
        title           [0] VisibleString,
        number          EmployeeNumber,
        dateOfHire      [1] Date,
        nameOfSpouse    [2] Name,
        children        [3] IMPLICIT
            SEQUENCE OF ChildInformation  DEFAULT  {} }

ChildInformation ::= SET
        { name          Name,
          dateOfBirth   [0] Date}

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE
    {givenName          NameString,
     initial            NameString (SIZE(1)),
     familyName         NameString}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER

Date ::= [APPLICATION 3] IMPLICIT VisibleString (FROM("0".."9") ^ SIZE(8)) -- YYYYMMDD

NameString ::= VisibleString (FROM("a".."z" | "A".."Z" | "-.") ^ SIZE(1..64))
```

A.2.2 ASN.1 description of a record value

The value of John Smith's personnel record is formally described below using ASN.1.

```
{ name {givenName "John",initial "P",familyName "Smith"},
  title         "Director",
  number        51,
  dateOfHire            "19710917",
  nameOfSpouse          {givenName "Mary",initial "T",familyName "Smith"},
  children
      {{name {givenName "Ralph",initial "T",familyName "Smith"},
             dateOfBirth "19571111"},
       {name {givenName "Susan",initial "B",familyName "Jones"},
             dateOfBirth "19590717"}}}
```

A.2.3 ALIGNED PER representation of this record value

The representation of the record value given above (after applying the ALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary. In the binary view an 'x' is used to represent pad bits that are encoded as zero-bits; they are used to align the fields from time to time.

The length of this encoding is 74 octets. For comparison, the same PersonnelRecord value encoded using the UNALIGNED variant of PER is 61 octets, BER with the definite length form is at least 136 octets, and BER with the indefinite length form is at least 161 octets.

A.2.3.1 Hexadecimal view

```
864A6F68 6E501053 6D697468 01330844 69726563 746F7219 7109170C 4D617279
5410536D 69746802 1052616C 70685410 536D6974 68195711 11105375 73616E42
104A6F6E 65731959 0717
```

A.2.3.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; and an ’x’ represents a zero pad bit that is used from time to time to align fields on an octet boundary.

1	Bitmap bit = 1 indicates "children" is present
000011x	Length of name.givenName = 4
01001010 01101111 01101000 01101110	name.givenName = "John"
01010000	name.initial = "P"
000100xx	Length of name.familyName = 5
01010011 01101101 01101001 01110100 01101000	name.familyName = "Smith"
00000001	Length of (employee) number = 1
00110011	(employee) number = 51
00001000	Length of title = 8
01000100 01101001 01110010 01100101 01100011 01110100 01101111 01110010	title = "Director"
0001 1001 0111 0001 0000 1001 0001 0111	dateOfHire = "19590717"
000011xx	Length of nameOfSpouse.givenName = 4
01001101 01100001 01110010 01111001	nameOfSpouse.givenName = "Mary"
01010100	nameOfSpouse.initial = "T"
000100xx	Length of nameOfSpouse.familyName = 5
01010011 01101101 01101001 01110100 01101000	nameOfSpouse.familyName = "Smith"
00000010	Number of children
000100xx	Length of children[0].givenName = 5
01010010 01100001 01101100 01110000 01101000	children[0].givenName = "Ralph"
01010100	children[0].initial = "T"
000100xx	Length of children[0].familyName = 5
01010011 01101101 01101001 01110100 01101000	children[0].familyName = "Smith"
0001 1001 0101 0111 0001 0001 0001 0001	children[0].dateOfBirth = "19571111"
000100xx	Length of children[1].givenName = 5
01010011 01110101 01110011 01100001 01101110	children[1].givenName = "Susan"
01000010	children[1].initial = "B"
000100xx	Length of children[1].familyName = 5
01001010 01101111 01101110 01100101 01110011	children[1].familyName = "Jones"
0001 1001 0101 1001 0000 0111 0001 0111	children[1].dateOfBirth = "19590717"

A.2.4 UNALIGNED PER representation of this record value

The representation of the record value given above (after applying the UNALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary. Note that pad bits do not occur in the UNALIGNED variant, and characters are encoded in the fewest number of bits possible.

The length of this encoding is 61 octets. For comparison, the same PersonnelRecord value encoded using the ALIGNED variant of PER is 74 octets, BER with the definite length form is at least 136 octets, and BER with the indefinite length form is at least 161 octets.

A.2.4.1 Hexadecimal view

```
865D51D2 888A5125 F1809984 44D3CB2E 3E9BF90C B8848B86 7396E8A8 8A5125F1
81089B93 D71AA229 4497C632 AE222222 985CE521 885D54C1 70CAC838 B8
```

A.2.4.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; a period (.) is used to mark octet boundaries; and an 'x' represents a zero-bit used to pad the final octet to an octet boundary.

1	Bitmap bit = 1 indicates "children" is present
000011	Length of name.givenName = 4
0.01011 101.010 10001.1 101001	name.givenName = "John"
0.10001	name.initial = "P"
000.100	Length of name.familyName = 5
01010.0 101000 1.00100 101.111 10001.1	name.familyName = "Smith"
0000000.1	Length of (employee) number = 1
0011001.1	(employee) number = 51
0000100.0	Length of title = 8
1000100 .1101001 1.110010 11.00101 110.0011 1110.100 11011.11 111001.0	title = "Director"
0001 100.1 0111 000.1 0000 100.1 0001 011.1	dateOfHire = "19590717"
000011	Length of nameOfSpouse.givenName = 4
0.01110 011.100 10110.1 110100	nameOfSpouse.givenName = "Mary"
0.10101	nameOfSpouse.initial = "T"
000.100	Length of nameOfSpouse.familyName = 5
01010.0 101000 1.00100 101.111 10001.1	nameOfSpouse.familyName = "Smith"
0000001.0	Number of children
000100	Length of children[0].givenName = 5
0.10011 011.100 10011.1 101011 1.00011	children[0].givenName = "Ralph"
010.101	children[0].initial = "T"
00010.0	Length of children[0].familyName = 5
010100 1.01000 100.100 10111.1 100011	children[0].familyName = "Smith"
0.001 1001 0.101 0111 0.001 0001 0001 0001	
0.001 1001 0.101 0111 0.001 0001 0.001 0001	children[0].dateOfBirth = "19571111"
0.00100	Length of children[1].givenName = 5
010.100 11000.0 101110 0.11100 101.001	children[1].givenName = "Susan"
00001.1	children[1].initial = "B"
000100	Length of children[1].familyName = 5
0.01011 101.010 10100.1 100000 1.01110	children[1].familyName = "Jones"
000.1 1001 010.1 1001 000.0 0111 000.1 0111xxx	children[1].dateOfBirth = "19590717"

A.3 Record that uses extension markers

A.3.1 ASN.1 description of the record structure

The structure of the hypothetical personnel record is formally described below using ASN.1 specified in ITU-T Rec. X.680 | ISO/IEC 8824-1 for defining types that are not extensible, and ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1 for defining extensible types.

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        name            Name,
        title           [0] VisibleString,
        number          EmployeeNumber,
        dateOfHire      [1] Date,
        nameOfSpouse    [2] Name,
        children        [3] IMPLICIT
            SEQUENCE (SIZE(2, ...)) OF ChildInformation  OPTIONAL  {},
    .. }

ChildInformation ::= SET
        { name          Name,
         dateOfBirth    [0] Date,
         ...,
         sex            [1] IMPLICIT ENUMERATED {male(1), female(2), unknown(3)} OPTIONAL
        }

Name ::= [APPLICATION 1] IMPLICIT SEQUENCE
        {givenName      NameString,
         initial        NameString (SIZE(1)),
         familyName     NameString,
         ...}

EmployeeNumber ::= [APPLICATION 2] IMPLICIT INTEGER (0..9999, ...)

Date ::= [APPLICATION 3] IMPLICIT VisibleString (FROM("0".."9") ^ SIZE(8, ..., 9..20)) -- YYYYMMDD

NameString ::= VisibleString (FROM("a".."z" | "A".."Z" | "-.") ^ SIZE(1..64, ...))
```

A.3.2 ASN.1 description of a record value

The value of John Smith’s personnel record is formally described below using ASN.1.

```
{ name {givenName "John",initial "P",familyName "Smith"},
  title         "Director",
  number        51,
  dateOfHire            "19710917",
  nameOfSpouse          {givenName "Mary",initial "T",familyName "Smith"},
  children
      {{name {givenName "Ralph",initial "T",familyName "Smith"},
            dateOfBirth "19571111"},
       {name {givenName "Susan",initial "B",familyName "Jones"},
            dateOfBirth "19590717", sex female}}}
```

A.3.3 ALIGNED PER representation of this record value

The representation of the record value given above (after applying the ALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary. In the binary view an 'x' is used to represent pad bits that are encoded as zero-bits; they are used to align the fields from time to time.

The length of this encoding is 83 octets. For comparison, the same PersonnelRecord value encoded using the UNALIGNED variant of PER is 65 octets, BER with the definite length form is at least 139 octets, and BER with the indefinite length form is at least 164 octets.

A.3.3.1 Hexadecimal view

```
40C04A6F 686E5008 536D6974 68000033 08446972 6563746F 72001971 0917034D
61727954 08536D69 74680100 52616C70 68540853 6D697468 00195711 11820053
7573616E 42084A6F 6E657300 19590717 010140
```

A.3.3.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; and an 'x' represents a zero pad bit that is used from time to time to align fields on an octet boundary.

0	No extension values present in PersonnelRecord
1	Bitmap bit = 1 indicates "children" is present
0	No extension values present in "name"
0	Length is within range of extension root
0000 11xxxxxx	Length of name.givenName = 4
01001010 01101111 01101000 01101110	name.givenName = "John"
01010000	name.initial = "P"
0	Length is within range of extension root
000100x	Length of name.familyName = 5
01010011 01101101 01101001 01110100 01101000	name.familyName = "Smith"
0xxxxxxx	Value is within range of extension root
00000000 00110011	(employee) number = 51
00001000	Length of title = 8
01000100 01101001 01110010 01100101 01100011 01110100 01101111 01110010	title = "Director"
0xxxxxxx	Length is within range of extension root
0001 1001 0111 0001 0000 1001 0001 0111	dateOfHire = "19590717"
0	No extension values present in nameOfSpouse
0	Length is within range of extension root
000011	Length of nameOfSpouse.givenName = 4
01001101 01100001 01110010 01111001	nameOfSpouse.givenName = "Mary"
01010100	nameOfSpouse.initial = "T"
0	Length is within range of extension root
000100x	Length of nameOfSpouse.familyName = 5
01010011 01101101 01101001 01110100 01101000	nameOfSpouse.familyName = "Smith"
0	Number of "children" is within the range of the extension root
0	No extension values present in children[0]
0	No extension values present in children[0].name
0	Length is within range of extension root
000100xx xxxx	Length of children[0].givenName = 5
01010010 01100001 01101100 01110000 01101000	children[0].givenName = "Ralph"
01010100	children[0].initial = "T"
0	Length is within range of extension root
000100x	Length of children[0].familyName = 5
01010011 01101101 01101001 01110100 01101000	children[0].familyName = "Smith"
0xxxxxxx	Length is within range of extension root
0001 1001 0101 0111 0001 0001 0001 0001	children[0].dateOfBirth = "19571111"
1	Extension value(s) present in children[1]
0	No extension values present in children[0].name
0	Length is within range of extension root
00010 0xxxxxxx	Length of children[1].givenName = 5
01010011 01110101 01110011 01100001 01101110	children[1].givenName = "Susan"

01000010	children[1].initial = "B"
0	Length is within range of extension root
000100x	Length of children[1].familyName = 5
01001010 01101111 01101110 01100101 01110011	children[1].familyName = "Jones"
0xxxxxxx	Length is within range of extension root
0001 1001 0101 1001 0000 0111 0001 0111	children[1].dateOfBirth = "19590717"
0000000	Length of extension addition bitmap for children[1] = 1
1	Indicate extension value for "sex" is present
00000001	Length of the complete encoding of "sex"
01xxxxxx	Complete encoding of "sex" = female

A.3.4 UNALIGNED PER representation of this record value

The representation of the record value given above (after applying the UNALIGNED variant of the Packed Encoding Rules defined in this Recommendation | International Standard) is shown below. The encoding is shown in hexadecimal, followed by a commented description of the encoding shown in binary. Note that pad bits do not occur in the UNALIGNED variant, and characters are encoded in the fewest number of bits possible.

The length of this encoding is 65 octets. For comparison, the same PersonnelRecord value encoded using the ALIGNED variant of PER is 83 octets, BER with the definite length form is at least 139 octets, and BER with the indefinite length form is at least 164 octets.

A.3.4.1 Hexadecimal view

```
40CBAA3A  5108A512  5F180330  889A7965  C7D37F20  CB8848B8  19CE5BA2  A114A24B
E3011372  7AE35422  94497C61  95711118  22985CE5  21842EAA  60B832B2  0E2E0202
80
```

A.3.4.2 Binary view

So as to make it easier to read the binary view of the data, blank lines are used to group fields that logically belong together (typically length/value pairs); a newline is used to delineate fields; space is used to delineate characters within a character string; a period (.) is used to mark octet boundaries; and an 'x' represents a zero-bit used to pad the final octet to an octet boundary.

0	No extension values present in PersonnelRecord
1	Bitmap bit = 1 indicates "children" is present
0	No extension values present in "name"
0	Length is within range of extension root
0000.11	Length of name.givenName = 4
001011 .101010 10.0011 1010.01	name.givenName = "John"
010001	name.initial = "P"
.0	Length is within range of extension root
000100	Length of name.familyName = 5
0.10100 101.000 10010.0 101111 1.00011	name.familyName = "Smith"
0	Value is within range of extension root
00.00000011.0011	(employee) number = 51
0000.1000	Length of title = 8
1000.100 11010.01 111001.0 1100101 1100011 1.110100 11.01111 111.0010	title = "Director"
0	Length is within range of extension root
000.1 1001 011.1 0001 000.0 1001 000.1 0111	dateOfHire = "19590717"
0	No extension values present in nameOfSpouse
0	Length is within range of extension root
0.00011	Length of nameOfSpouse.givenName = 4
001.110 01110.0 101101 1.10100	nameOfSpouse.givenName = "Mary"
010.101	nameOfSpouse.initial = "T"

0	Length is within range of extension root
0001.00	Length of nameOfSpouse.familyName = 5
010100 .101000 10.0100 1011.11 100011	nameOfSpouse.familyName = "Smith"
.0	Number of "children" is within the range of the extension root
0	No extension values present in children[0]
0	No extension values present in children[0].name
0	Length is within range of extension root
0001.00	Length of children[0].givenName = 5
010011 .011100 10.0111 1010.11 100011	children[0].givenName = "Ralph"
.010101	children[0].initial = "T"
0	Length is within range of extension root
0.00100	Length of children[0].familyName = 5
010.100 10100.0 100100 1.01111 100.011	children[0].familyName = "Smith"
0	Length is within range of extension root
0001 .1001 0101 .0111 0001 .0001 0001 .0001	children[0].dateOfBirth = "19571111"
1	Extension value(s) present in children[1]
0	No extension values present in children[0].name
0	Length is within range of extension root
0.00100	Length of children[1].givenName = 5
010.100 11000.0 101110 0.11100 101.001	children[1].givenName = "Susan"
00001.1	children[1].initial = "B"
0	Length is within range of extension root
000100	Length of children[1].familyName = 5
.001011 10.1010 1010.01 100000 .101110	children[1].familyName = "Jones"
0	Length is within range of extension root
0.001 1001 0.101 1001 0.000 0111 0.001 0111	children[1].dateOfBirth = "19590717"
0.000000	Length of extension addition bitmap for children[1] = 1
1	Indicate extension value for "sex" is present
0.0000001	Length of the complete encoding of "sex"
0.1xxxxxx	Complete encoding of "sex" = female
x	Pad bit to create complete encoding of PersonnelRecord

Annex B

Observations on combining PER-visible constraints

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

Certain properties can be observed when subtype elements, each of which separately may be PER visible, are combined. The following describes by means of examples such properties:

B.1 The effective size constraint for

```
A ::= IA5String (SIZE(1..4) | SIZE(9..10))
```

is

```
A ::= IA5String (SIZE(1..4 | 9..10))
```

B.2 When a PermittedAlphabet constraint is combined in the same subtype specification with other PermittedAlphabet constraints, there is no effective PermittedAlphabet constraint containing fewer than all characters in the unconstrained type unless there is a single PermittedAlphabet specification that is a superset of all other PermittedAlphabet specifications in that subtype specification. Also, if size constraints are included in that constraint specification the superset PermittedAlphabet specification must have an effective size constraint combined with it by means of the INTERSECTION operation such that the effective size constraint is a superset of all other size constraints imposed on the type. For example,

```
B ::= IA5String ( FROM ("AB") ^ SIZE(1..2) |
                  FROM ("DE") ^ SIZE(3) |
                  FROM ("ABCDE") ^ (SIZE(1..5) )
```

has an effective size constraint and effective PermittedAlphabet constraint of

```
B ::= IA5String (FROM ("ABCDE") ^ SIZE(1..5))
```

because this is a superset of the more complex expression above – thus it is PER-visible. On the other hand, in the following the effective PermittedAlphabet constraint is the entire set of characters allowed for IA5String, since there is no single equivalent PermittedAlphabet constraint that can be written. Thus, the constraint on the following is not PER-visible:

```
C ::= IA5String (FROM("AB") | FROM("CD"))    -- This is not equivalent to (FROM("ABCD"))
```

B.3 Size constraints can be freely combined as long as there is no PermittedAlphabet involved. For example, with

```
E ::= IA5String (SIZE(1..4) | SIZE(5..10) ^ FROM("ABCD") | SIZE(6..10))
```

there is no PER-visible size constraint (since size 5 cannot occur for all possible characters), while if we had

```
E ::= IA5String (SIZE(1..4) | SIZE(6..10) ^ FROM("ABCD") | SIZE(6..10))
```

there would be a PER-visible size constraint of SIZE(1..4 | 6..10) though the effective PermittedAlphabet constraint would be the set of all IA5String characters. FROM("ABCD") in this case does not form a PER-visible constraint because this constraint does not apply to all possible values of E (e.g. if the string length is 1 then the character is not restricted to one of "ABCD".)

Annex C

Support for the PER algorithms

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

An application standard, or an International Standardized Profile, may specify which of the Packed Encoding Rules are to be supported, and the corresponding transfer syntaxes to be offered or accepted in negotiation.

Where it has requirements for the use of relay safe and/or canonical encodings within EMBEDDED PDVs (or EXTERNALs) or CHARACTER STRINGs, this should be clearly stated.

The following text provides guidelines that can be used in the production of normative text.

C.1 A canonical encoding is intended for use when security features are being applied to the encoding (see Annex D of ITU-T Rec. X.690 | ISO/IEC 8825-1). Use of CANONICAL-PER can involve significant additional CPU utilization cost when the value to be encoded includes a set-of type, and is generally not recommended for protocols unless security features are required.

C.2 Where an abstract syntax value contains embedded material that is encoded using a transfer or abstract syntax different from that associated with the abstract syntax value, it is strongly recommended that the embedded material be encoded in a relay-safe manner. A canonical encoding rule will be required if security features are important. In this context, particular attention should be placed on the level of ISO/IEC 10646-1 which is to be used for the type BMPString or UniversalString, as only ISO/IEC 10646-1 implementation level 1 is guaranteed to be canonical.

C.3 Where a presentation context is established for the one-way flow of data, it is strongly recommended that the data be encoded in a relay-safe manner.

C.4 Where a presentation context is established for the two-way flow of data, there may be significant gains of flexibility and economy if BASIC-PER is employed.

C.5 It is strongly recommended that all implementations supporting decoding of any PER ALIGNED variant transfer syntax should support decoding of the BASIC-PER ALIGNED variant (and hence of the CANONICAL-PER ALIGNED variant), and should accept presentation contexts identified with any of those two encoding rules provided the contexts are being established for the receipt of data by that implementation. Similarly for the UNALIGNED variant.

C.6 It is recommended, in the interests of interworking, that all implementations of PER support both the ALIGNED and the UNALIGNED variant (the added implementation complexity is small). Which is offered in an instance of communication (either or both) is a local management matter, and which is accepted if both are offered is also a local management matter. If only one is offered, it should be accepted.

C.7 Acceptance of these recommendations is particularly important for the vendors of general-purpose tools. Where an implementation is specific to some particular application, support of a single PER transfer syntax (perhaps specified by that application designer) may be fully acceptable.

Annex D

Support for the ASN.1 rules of extensibility

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

D.1 These Packed Encoding Rules are dependent on the total definition of the type to which they are applied. In general, if any changes other than those of a purely syntactic nature are made to the type definition then the encoding for all values using that part of the specification will be affected. In particular, addition of further optional components to a sequence, converting a component to a CHOICE of that component and some other type, or relaxing or tightening constraints on some component are all likely to change the encoding of values of the type.

D.2 Nonetheless, these encoding rules have been designed to ensure that the requirements on encoding rules specified in the ASN.1 model of type extension (see ITU-T Rec. X.680/Amd. 1 | ISO/IEC 8824-1/Amd. 1) are satisfied.

D.3 Where a type is not part of an extension sequence (no extension marker present), then the text earlier in this annex applies: PER provides no support for extensibility of that type. Where a sequence or set type has an extension marker, but no extension additions, then there is a one-bit overhead (which may become one octet due to padding in the ALIGNED variants), compared with the same type without an extension marker. Where additions are present in the type and are actually transmitted in an instance of communication, there is a further overhead of about one octet, plus an additional length field for each extension addition that is transmitted, compared with the same type with the extension marker removed.

D.4 It is important to note that both the addition and removal of an extension marker changes the bits on the line, and will in general require a version number change for the protocol.

D.5 There are no changes to the encoding from the inclusion of an extension marker in an information object set, or from the addition or removal of exception specifications, but these may of course represent changes in the required behavior of an implementation and could still require a version number change for the protocol.

Annex E

Tutorial annex on concatenation of PER encodings

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

E.1 PER encodings are self-delimiting given knowledge of the encoding rules and the type of the encoding. The complete encodings for the ALIGNED and UNALIGNED variant are always a multiple of 8 bits.

E.2 For the purposes of carrying PER encodings in the OSI presentation layer protocol, encodings of the ALIGNED and UNALIGNED variants can be concatenated in the octet string option.

Annex F

Assignment of object identifier values

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

The following object identifier and object descriptor values are assigned in this Recommendation | International Standard:

For BASIC-PER, ALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) aligned (0)}

"Packed encoding of a single ASN.1 type (basic aligned)"

For BASIC-PER, UNALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) basic (0) unaligned (1)}

"Packed encoding of a single ASN.1 type (basic unaligned)"

For CANONICAL-PER, ALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) aligned (0)}

"Packed encoding of a single ASN.1 type (canonical aligned)"

For CANONICAL-PER, UNALIGNED variant:

{joint-iso-itu-t asn1 (1) packed-encoding (3) canonical (1) unaligned (1)}

"Packed encoding of a single ASN.1 type (canonical unaligned)"