INTERNATIONAL STANDARD

ISO/IEC 8825-7

Third edition
2021-06

Information technology — ASN.1 encoding rules —

Part 7: Specification of Octet Encoding Rules (OER)

Technologies de l'information — Règles de codage ASN.1 —
Partie 7: Spécification des règles de codage des octets (OER)

Reference number
ISO/IEC 8825-7:2021(E)

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

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives or www.iec.ch/members_experts/refdocs)

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see www.iso.org/iso/foreword.html. In the IEC, see www.iec.ch/understanding-standards.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 6, *Telecommunications and information exchange between systems,* in collaboration with ITU-T. The identical text is published as ITU-T X.696 (02/2021).

This third edition cancels and replaces the second edition (ISO/IEC 8825-7:2015), which has been technically revised. It also incorporates ISO/IEC 8825-7:2015/Cor 2:2017, ISO/IEC 8825-7:2015/Cor 3:2018, ISO/IEC 8825-7:2015/Cor 4:2018.

A list of all parts in the ISO/IEC 8825 series can be found on the ISO and IEC websites.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html and www.iec.ch/national-committees.

CONTENTS

Page

Introduction

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

This Recommendation | International Standard defines encoding rules that may be applied to values of ASN.1 types which have been defined using the notation specified in the above-mentioned publications. 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 two sets of Octet Encoding Rules, so-called because the encoding of every type takes a whole number of octets. Encoding and decoding data with the Octet Encoding Rules is usually faster than encoding and decoding the same data with the Basic Encoding Rules (described in Rec. ITU-T X.690 | ISO/IEC 8825-1) or the Packed Encoding Rules (described in Rec. ITU-T X.691 | ISO/IEC 8825-2).

NOTE – The encoding rules specified in this Recommendation | International Standard derive from the Octet Encoding Rules (OER) published by American Association of State Highway and Transportation Officials (AASHTO), Institute of Transportation Engineers (ITE) and National Electrical Manufacturers Association (NEMA) as NTCIP 1102:2004. In most practical cases, an implementation of this Recommendation | International Standard can interoperate with an implementation of NTCIP 1102.

Clauses 8 to 30 specify the BASIC-OER encoding of ASN.1 types.

Clause 31 specifies the CANONICAL-OER encoding of ASN.1 types.

Annex A is informative and contains examples of BASIC-OER and CANONICAL-OER encodings.

Annex B is informative and addresses the Interoperability of the encoding rules with NTCIP 1102:2004.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – ASN.1 encoding rules: Specification of Octet Encoding Rules (OER)

1 Scope

This Recommendation | International Standard specifies a set of Basic Octet Encoding Rules (BASIC-OER) that may be used to derive a transfer syntax for values of the types defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4. This Recommendation | International Standard also specifies a set of Canonical Octet Encoding Rules (CANONICAL-OER) which provides constraints on the Basic Octet Encoding Rules and produces a unique encoding for any given ASN.1 value. It is implicit in the specification of these encoding rules that they are also to be used for decoding.

The encoding rules specified in this Recommendation | International Standard:

- are used at the time of communication;
- are intended for use in circumstances where encoding/decoding speed is the major concern in the choice of encoding rules;
- allow the extension of an abstract syntax by addition of extra values for all forms of extensibility described in Rec. ITU-T X.680 | ISO/IEC 8824-1.

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

NOTE – This Recommendation | International Standard is based on ISO/IEC 10646:2003 and the Unicode standard version 3.2.0:2002. It cannot be applied using later versions of these two standards.

2.1 Identical Recommendations | International Standards

- Recommendation ITU-T X.680 (2021) | ISO/IEC 8824-1:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- Recommendation ITU-T X.681 (2021) | ISO/IEC 8824-2:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification*.
- Recommendation ITU-T X.682 (2021) | ISO/IEC 8824-3:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.*
- Recommendation ITU-T X.683 (2021) | ISO/IEC 8824-4:2021, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications*.
- Recommendation ITU-T X.690 (2021) | ISO/IEC 8825-1:2021, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*.
- Recommendation ITU-T X.691 (2021) | ISO/IEC 8825-2:2021, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).*

NOTE – The references above shall be interpreted as references to the identified Recommendations | International Standards together with all their published amendments and technical corrigenda.

2.2 Additional references

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

3 Definitions

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

3.1 Specification of basic notation

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

3.2 Information object specification

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

3.3 Constraint specification

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

a) component relation constraint;

b) table constraint.

3.4 Parameterization of ASN.1 specification

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

– variable constraint.

3.5 Basic Encoding Rules (BER)

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

a) data value;

b) dynamic conformance;

c) encoding (of a data value);

d) receiver;

e) sender;

f) static conformance.

3.6 Packed Encoding Rules (PER)

This Recommendation | International Standard makes use of the following terms defined in Rec. ITU-T X.691 | ISO/IEC 8825-2:

a) canonical encoding;

b) composite type;

c) composite value;

d) known-multiplier character string type;

e) outermost type;

f) relay-safe encoding;

g) simple type;

h) textually dependent.

3.7 Additional definitions

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

3.7.1 abstract syntax value: A value of an abstract syntax (defined as a set of values of a single ASN.1 type) which is to be encoded by BASIC-OER or CANONICAL-OER, or which is generated by BASIC-OER or CANONICAL-OER decoding.

3.7.2 effective value constraint (of an integer type): The smallest integer range that includes all the values of the integer type that are permitted by the OER-visible constraints (see 8.2.7).

3.7.3 effective size constraint (of a string type): The smallest integer range that includes the lengths of all the values of the string type that are permitted by the OER-visible constraints (see 8.2.8).

3.7.4 fixed-size signed number: A word (see 3.7.13) representing a negative, zero or positive whole number encoded as a signed integer encoding (see 3.7.9).

NOTE 1 – The least significant bit of the whole number is stored in bit 1 of the last octet of the word.

NOTE 2 – The range of integers that can be encoded as fixed-size signed numbers is −128 to 127 for a one-octet word, −32768 to 32767 for a two-octet word, −2147483648 to 2147483647 for a four-octet word, and −9223372036854775808 to 9223372036854775807 for an eight-octet word.

3.7.5 fixed-size unsigned number: A word (see 3.7.13) representing a zero or positive whole number encoded as an unsigned integer encoding (see 3.7.10).

NOTE 1 – The least significant bit of the whole number is stored in bit 1 of the last octet of the word.

NOTE 2 – The smallest integer that can be encoded as fixed-size unsigned numbers of any size is 0. The largest integer that can be encoded as a fixed-size unsigned number is 255 for a one-octet word, 65535 for a two-octet word, 4294967295 for a four-octet word, and 18446744073709551615 for an eight-octet word.

3.7.6 length determinant: A group of one or more consecutive octets encoding the length of a series of octets (see 8.6).

3.7.7 octet: A group of eight consecutive bits, numbered from bit 8 (the most significant bit) to bit 1 (the least significant bit).

NOTE – Within an OER encoding, each octet starts at a location that is a whole multiple of eight bits from the first bit of the encoding.

3.7.8 OER-visible constraint: An instance of use of the ASN.1 constraint notation that affects the OER encoding of a value.

3.7.9 signed integer encoding: The encoding of a whole number into a group of consecutive octets of a specified length as a 2's-complement binary integer, which provides representations for whole numbers that are equal to, greater than or less than zero.

NOTE – The value of a signed integer encoding is derived by numbering the bits in the octets of the group, starting with bit 1 of the last octet 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 (starting from 0) in the above numbering sequence. The value of the signed integer encoding 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.

3.7.10 unsigned integer encoding: The encoding of a whole number into a group of consecutive octets of a specified length as an unsigned binary integer, which provides representations for whole numbers that are equal to or greater than zero.

NOTE – The value of an unsigned integer encoding is derived by numbering the bits in the octets of the group, starting with bit 1 of the last octet 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 (starting from 0) in the above numbering sequence. The value of the unsigned integer encoding is obtained by summing the numerical values assigned to each bit for those bits which are set to one.

3.7.11 variable-size signed number: A group of one or more consecutive octets containing a negative, zero, or positive whole number encoded as a signed integer encoding, with the least significant bit of the binary number stored in bit 1 of the last octet of the variable-size signed number.

NOTE – There are no restrictions to the length of such a group of octets. In particular, the Basic Octet Encoding Rules (but not the Canonical Octet Encoding Rules) allow the presence of redundant octets set to 0 (for zero or positive values) or 255 (for negative values) at the beginning of the group.

3.7.12 variable-size unsigned number: A group of one or more consecutive octets containing a zero or positive whole number encoded as an unsigned integer encoding, with the least significant bit of the binary number stored in bit 1 of the last octet of the variable-size unsigned number.

NOTE – There are no restrictions to the length of such a group of octets. In particular, the Basic Octet Encoding Rules (but not the Canonical Octet Encoding Rules) allow the presence of redundant octets set to 0 at the beginning of the group.

3.7.13 word: A group of one, two, four or eight consecutive octets containing the encoding of a whole number, where the first octet contains the most significant part of the number and the last octet contains the least significant part of the number.

NOTE 1 – A single octet is also a word according to this definition. The octet ordering of words consisting of 2, 4 or 8 octets is big-endian.

NOTE 2 – Within an OER encoding, a word can start at any location within the encoding that is a whole number of octets from the beginning of the encoding (that is, there is no requirement that a word should start on a word boundary).

4 Abbreviations

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

ASN.1	Abstract Syntax Notation One
BER	Basic Encoding Rules of ASN.1
ITS	Intelligent Transportation Systems
NTCIP	National Transportation Communications for ITS Protocol
OER	Octet Encoding Rules of ASN.1
PER	Packed Encoding Rules of ASN.1
PDU	Protocol Data Unit

5 Convention

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

6 Encodings specified by this Recommendation | International Standard

6.1 This Recommendation | International Standard specifies two sets of 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.

6.2 Without knowledge of the type of the value encoded, it is not possible to determine the structure of the encoding. In particular, the end of the encoding cannot be determined from the encoding itself without knowledge of the type being encoded.

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

6.4 The most general set of encoding rules specified in this Recommendation | International Standard is BASIC-OER, which does not in general produce a canonical encoding.

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

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

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

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

6.7 OER encodings are self-delimiting only with knowledge of the type of the encoded value. Encodings are always a whole 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 OER, in which case the value may be encoded as a single ASN.1 type (i.e., an open type). When carried in an OSI presentation protocol, the "full encoding" (as defined in Rec. ITU-T X.226 | ISO/IEC 8823-1) with the **`OCTET STRING`** alternative shall be used.

6.8 CANONICAL-OER provides an alternative to both the Distinguished Encoding Rules (DER) and Canonical Encoding Rules (CER) specified in Rec. ITU-T X.690 | ISO/IEC 8825-1 where a canonical and relay-safe encoding is required.

7 Conformance

7.1 Dynamic conformance for the Basic Octet Encoding Rules is specified by clauses 8 to 30. Dynamic conformance for the Canonical Octet Encoding Rules is specified by clause 31.

7.2 Static conformance is specified by those standards which specify the application of these encoding rules.

7.3 Alternative encodings are permitted by the Basic Octet Encoding Rules as encoder's options. Decoders that claim conformance to BASIC-OER shall support all BASIC-OER encoding alternatives. No alternative encodings are permitted by the CANONICAL-OER for the encoding of an ASN.1 value.

7.4 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 octet 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.

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

NOTE – When CANONICAL-OER 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-OER as the transformation from the abstract syntax value to an initial bitstring (which is then hashed).

8 General provisions

8.1 Use of the type notation

8.1.1 These encoding rules make specific use of the ASN.1 type notation as specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, Rec. ITU-T X.681 | ISO/IEC 8824-2, Rec. ITU-T X.682 | ISO/IEC 8824-3, Rec. ITU-T X.683 | ISO/IEC 8824-4, and can only be applied to encode the values of a single ASN.1 type specified using that notation.

8.1.2 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 numeric 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 OER-visible constraints;

f) whether the type of a component is an open type.

8.2 Constraints

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

8.2.1 In general, the constraint on a type will consist of individual constraints combined using some or all of set arithmetic, contained subtype constraints, and serial application of constraints.

The following constraints are OER-visible:

a) non-extensible single value constraints and value range constraints on integer types;

b) non-extensible single value constraints on real types where the single value is either plus zero or minus zero or one of the special real values **`PLUS-INFINITY`**, **`MINUS-INFINITY`** and **`NOT-A-NUMBER`**;

c) non-extensible size constraints on known-multiplier character string types, octetstring types, and bitstring types;

d) non-extensible property settings constraints on the time type or on the useful and defined time types;

e) inner type constraints applying OER-visible constraints to real types when used to restrict the mantissa, base, or exponent;

f) inner type constraints applied to CHARACTER STRING or EMBEDDED-PDV types when used to restrict the value of the syntaxes component to a single value, or when used to restrict identification to the fixed alternative;

g) contained subtype constraints in which the constraining type carries an OER-visible constraint.

8.2.2 All other constraints are not OER-visible. In particular, the following constraints are not OER-visible:

a) constraints that are expressed in human-readable text or in ASN.1 comment;

b) variable constraints (see Rec. ITU-T X.683 | ISO/IEC 8824-4, 10.3 and 10.4);

c) user-defined constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 9.1);

d) table constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3);

e) component relation constraints (see Rec. ITU-T X.682 | ISO/IEC 8824-3, 10.7);

f) constraints whose evaluation is textually dependent on a table constraint or a component relation constraint (see Rec. ITU-T X.682 | ISO/IEC 8824-3);

g) extensible subtype constraints;

h) size constraints on restricted character string types which are not known-multiplier character string types (see clause 27);

i) single value subtype constraints applied to a character string type;

j) permitted alphabet constraints;

k) pattern constraints;

l) constraints on real types except those specified in 8.2.1 (b) and (e);

m) inner type constraints applied to components of unrestricted character string, embedded-pdv or external types, except those specified in 8.2.1 (f);

n) constraints on the useful types.

8.2.3 If a type is specified using a serial application of constraints, each of those constraints may or may not be individually OER-visible. If the last subtype constraint of the series of constraints is OER-visible and contains an extension marker, then that subtype constraint is extensible for the purposes of these encoding rules. Any other constraint is not extensible for the purposes of these encoding rules, even if it contains an extension marker.

NOTE – In a serial application of constraints, each subtype constraint removes the extensibility specified in earlier constraints of the series of constraints (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 50.8).

8.2.4 If a constraint that is OER-visible is part of an **`INTERSECTION`** construction, then the resulting constraint is OER-visible, and consists of the **`INTERSECTION`** of all the OER-visible parts (with the non-OER-visible parts ignored).

8.2.5 If a constraint that is not OER-visible is part of a **`UNION`** construction, then the resulting constraint is not OER-visible.

8.2.6 If a constraint has an **`EXCEPT`** clause, the **`EXCEPT`** keyword and the following value set is completely ignored, whether the value set following the **`EXCEPT`** keyword is OER-visible or not.

8.2.7 The effective value constraint of an integer type is an integer range determined as follows, taking into account all the OER-visible constraints present in the type definition and ignoring any constraints that are not OER-visible:

a) The lower bound of the effective value constraint is the least permitted value of the integer type, if such a value exists; otherwise, the effective value constraint has no finite lower bound.

b) The upper bound of the effective value constraint is the greatest permitted value of the integer type, if such a value exists; otherwise, the effective value constraint has no finite upper bound.

8.2.8 The effective size constraint of a string type (a known-multiplier character string type, an octetstring type, or a bitstring type) is a single integer range determined as follows, taking into account all the OER-visible constraints present in the type definition and ignoring any constraints that are not OER-visible:

a) The lower bound of the effective size constraint is the length of the shortest permitted value of the string type (possibly zero).

b) The upper bound of the effective size constraint is the length of the longest permitted value of the string type, if such length is finite; otherwise, the effective size constraint has no finite upper bound.

8.3 Type and value model used for encoding

8.3.1 An ASN.1 type is either a simple type or 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 18.2 and 20.1. The notation also permits the application of constraints and of error specifications. OER-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 OER type and value model.

8.3.2 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.

8.4 Types to be encoded

8.4.1 Clauses 9 to 30 specify the encoding of the following types: Boolean, integer, enumerated, real, bitstring, octetstring, null, sequence, sequence-of, set, set-of, choice, object identifier, relative object identifier, internationalized resource reference, relative internationalized resource reference, embedded-pdv, external, restricted character string, unrestricted character string, time, and open types.

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

8.4.2 This Recommendation | International Standard does not contain specific provisions for the encoding of tagged types, except as stated in 18.2 and 20.1, tagging is not visible in the type and value model used for these encoding rules. Tagged types are thus encoded according to the type which has been tagged.

8.4.3 An encoding prefixed type is encoded according to the type which has been prefixed.

8.4.4 The useful types **`GeneralizedTime`**, **`UTCTime`**, and **`ObjectDescriptor`** shall be encoded as if they had been replaced by their definitions given in Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 45. Constraints on the useful types are not OER-visible.

8.4.5 A type defined using a value set assignment shall be encoded as if the type had been defined using the production specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 16.8.

8.5 Production of a complete OER encoding

8.5.1 If an ASN.1 type is encoded using OER and the encoding is contained in:

a) an ASN.1 bitstring type or octetstring type; or

b) an ASN.1 open type; or

c) any part of an ASN.1 external or embedded-pdv type; or

d) any carrier protocol that is not defined using ASN.1,

then that ASN.1 type is defined as an outermost type, and clause 8.5.2 shall be applied to all the encodings of its values.

8.5.2 The series of words and groups of octets produced as a result of applying this Recommendation | International Standard to an abstract value of an outermost type shall be concatenated into a string of octets, which is the complete encoding of the abstract value of the outermost type.

8.6 Length determinant

8.6.1 A length determinant occurs at the beginning of the encoding of many types as specified in the respective clauses.

8.6.2 When a length determinant is included in the encoding of a type, it shall indicate the number of octets (zero or more) occupied by the remainder of the encoding of that type.

NOTE – The encoding of a sequence-of or set-of type does not utilize a length determinant, but a different determinant called quantity (see 17.2), which indicates the number of occurrences.

8.6.3 There are two forms of length determinant – a short form and a long form. The short form allows the specification of lengths between 0 and 127 octets. The long form allows the specifications of lengths between 0 and an effectively unlimited number ($2^{1016} – 1$) of octets. Values whose encoding would be longer than $2^{1016} – 1$ octets cannot be encoded in these encoding rules.

8.6.4 The short form of length determinant consists of a single octet. Bit 8 of this octet shall be set to '0', and bits 7 to 1 of this octet shall contain the length (0 to 127) encoded as an unsigned binary integer into 7 bits.

8.6.5 The long form of length determinant consists of an initial octet followed by one or more subsequent octets. Bit 8 of the initial octet shall be set to 1, and bits 7 to 1 of this octet shall indicate the number of subsequent octets (1 to 127). The length shall be encoded as a variable-size unsigned number into the subsequent octets.

NOTE – In CANONICAL-OER, the long form of length determinant is used only for lengths greater than 127, and the length is encoded in the smallest possible number of octets (see 31.2). These restrictions do not apply to BASIC-OER.

8.7 Encoding of tags

8.7.1 In these encoding rules, tags are encoded only as part of the encoding of a choice type, where the tag indicates which alternative of the choice type is the chosen alternative (see 20.1).

8.7.2 The encoding of a tag shall consist of one or more octets, as specified in 8.7.2.1 to 8.7.2.3.

8.7.2.1 Bits 8 and 7 of the first octet shall denote the tag class of the tag, as follows:

a) '00'B denotes the universal tag class;

b) '01'B denotes the application tag class;

c) '10'B denotes the context-specific tag class; and

d) '11'B denotes the private tag class.

8.7.2.2 If the tag number is less than 63, it shall be encoded into bits 6 to 1 of the first (and only) octet.

8.7.2.3 If the tag number is greater or equal to 63, it shall be encoded into an initial octet followed by one or more subsequent octets, as follows:

a) Bits 6 to 1 of the initial octet shall be set to '111111'B.

b) The tag number shall be encoded into bits 7 to 1 of each subsequent octet (seven bits in each octet), with bit 1 of the final subsequent octet containing the least significant bit of the tag number ("big-endian" encoding).

c) Bits 7 to 1 of the first subsequent octet shall not be all set to 0.

d) Bit 8 of each subsequent octet except the last shall be set to 1.

e) Bit 8 of the final subsequent octet shall be set to 0.

9 Encoding of Boolean values

The encoding of a Boolean value shall be a single octet. The octet value 0 denotes the Boolean value **`FALSE`** and a non-zero octet value denotes the Boolean value **`TRUE`**.

NOTE – In CANONICAL-OER, only the octet value 'FF'H can be used to encode the value **`TRUE`** (see 31.3). This restriction does not apply to BASIC-OER.

10 Encoding of integer values

10.1 The encoding of an integer value depends on the effective value constraint (see 8.2.7) of the integer type as determined by the OER-visible constraints present in the type definition.

10.2 There are two main cases:

a) The effective value constraint has a lower bound, and that lower bound is zero or positive.

b) The effective value constraint has either a negative lower bound or no lower bound.

10.3 In case (a) of 10.2, the encoding of the integer value depends on the upper bound of the effective value constraint, as follows:

a) If the upper bound is less than or equal to $2^8 – 1$ (255), then every value of the integer type shall be encoded as a fixed-size unsigned number in a one-octet word; else

b) if the upper bound is less than or equal to $2^{16} – 1$ (65535), then every value of the integer type shall be encoded as a fixed-size unsigned number in a two-octet word; else

c) if the upper bound is less than or equal to $2^{32} – 1$ (4294967295), then every value of the integer type shall be encoded as a fixed-size unsigned number in a four-octet word; else

d) if the upper bound is less than or equal to $2^{64} – 1$ (18446744073709551615), then every value of the integer type shall be encoded as a fixed-size unsigned number in an eight-octet word; else

e) (the effective value constraint has either an upper bound greater than 2^{64-1} or no upper bound) every value of the integer type shall be encoded as a length determinant (see 8.6) followed by a variable-size unsigned number (occupying at least as many whole octets as are necessary to carry the value).

10.4 In case (b) of 10.2, the encoding of the integer value depends on the lower bound and upper bound of the effective value constraint, as follows:

a) If the lower bound is greater than or equal to $–2^7$ (–128) and the upper bound is less than or equal to $2^7 – 1$ (127), then every value of the integer type shall be encoded as a fixed-size signed number in a one-octet word; else

b) if the lower bound is greater than or equal to $–2^{15}$ (–32768) and the upper bound is less than or equal to $2^{15} – 1$ (32767), then every value of the integer type shall be encoded as a fixed-size signed number in a two-octet word; else

c) if the lower bound is greater than or equal to $–2^{31}$ (–2147483648) and the upper bound is less than or equal to $2^{31} – 1$ (2147483647), then every value of the integer type shall be encoded as a fixed-size signed number in a four-octet word; else

d) if the lower bound is greater than or equal to $–2^{63}$ (–9223372036854775808) and the upper bound is less than or equal to $2^{63} – 1$ (9223372036854775807), then every value of the integer type shall be encoded as a fixed-size signed number in an eight-octet word; else

e) (the effective value constraint has a lower bound less than $–2^{63}$, no lower bound, an upper bound greater than $2^{63–1}$, or no upper bound) every value of the integer type shall be encoded as a length determinant (see 8.6) followed by a variable-size signed number (occupying at least as many whole octets as are necessary to carry the value).

NOTE 1 – In CANONICAL-OER, in case (e) of 10.3 and case (e) of 10.4, the integer value is encoded in the smallest possible number of octets (see 31.4). This restriction does not apply to BASIC-OER.

NOTE 2 – Unlike PER, OER does not add an extension bit at the beginning of the encoding of an integer type with an extensible OER-visible constraint. Such a type is encoded as an integer type with no bounds.

11 Encoding of enumerated values

11.1 The encoding of an enumerated value shall consist of the distinct numeric value associated with the enumerated value (see Rec. ITU-T X.680 | ISO/IEC 8824-1, clause 20) encoded into one or more octets.

11.2 There are two forms of enumerated type encoding – a short form and a long form. The short form allows the encoding of numeric values between 0 and 127. The long form allows the encoding of numeric values within an effectively unlimited range (between $–2^{1015}$ and $2^{1015} – 1$). Negative numeric values less than $–2^{1015}$ and positive numeric values greater than $2^{1015} – 1$ cannot be encoded in these encoding rules.

11.3 The short form of enumerated type encoding consists of a single octet. Bit 8 of this octet shall be set to '0', and bits 7 to 1 of this octet shall contain the numeric value (0 to 127) encoded as an unsigned binary integer into 7 bits.

11.4 The long form of enumerated type encoding consists of an initial octet followed by one or more subsequent octets. Bit 8 of the initial octet shall be set to 1, and bits 7 to 1 of this octet shall indicate the number of subsequent octets (1 to 127). The numeric value shall be encoded as a variable-size signed number into the subsequent octets.

NOTE 1 – This encoding differs from the encoding of a length determinant (see 8.6) in that the subsequent octets contain a variable-size unsigned number in the case of a length determinant but a variable-size signed number in the case of an enumerated type.

NOTE 2 – In CANONICAL-OER, the long form of enumerated type encoding is used only for numeric values outside the range 0..127, and the numeric value is encoded in the smallest possible number of octets (see 31.5). These restrictions do not apply to BASIC-OER.

11.5 The presence of an extension marker in the definition of an enumerated type does not affect the encoding of the values of the enumerated type.

12 Encoding of real values

12.1 The encoding of a real value depends on the effective value constraints of the mantissa, base, and exponent of the real type, which shall be determined as follows:

a) If there are no OER-visible constraints, the effective value constraints of the mantissa and exponent have no finite lower and upper bounds, and the effective value constraint of the base is the integer range 2..10.

b) If there is an inner type constraint that applies OER-visible constraints to one or more components of the real type (**mantissa**, **base**, and **exponent**), the effective value constraint of each component is the one resulting from the inner type constraint.

c) When two or more OER-visible constraints are combined into an **INTERSECTION** construction, they result in an OER-visible constraint (see 8.2.4); the effective value constraint of the mantissa in that OER-visible constraint is the intersection of all the effective value constraints of the mantissas in the members of the **INTERSECTION** construction; the effective value constraint of the base and the effective value constraint of the exponent are determined in the same way.

d) When two or more OER-visible constraints are combined into a **UNION** construction, they result in an OER-visible constraint (see 8.2.5); the effective value constraint of the mantissa in that OER-visible constraint is the smallest integer range that includes all the effective value constraints of the mantissas in the members of the **UNION** construction; the effective value constraint of the base and the effective value constraint of the exponent are determined in the same way.

e) When an **EXCEPT** clause is present, it is ignored (see 8.2.6).

12.2 If all of the following are true:

a) the lower bound of the effective value constraint of the mantissa is greater than or equal to $-2^{24} + 1$ (–16777215) and its upper bound is less than or equal to $2^{24} - 1$ (16777215);

b) the effective value constraint of the base is the fixed value 2; and

c) the lower bound of the effective value constraint of the exponent is greater than or equal to –149 and its upper bound is less than or equal to 104,

then the real value shall be encoded in the binary32 (single precision) floating-point format specified in IEEE 754.

12.3 Otherwise, if all of the following are true:

a) the lower bound of the effective value constraint of the mantissa is greater than or equal to $-2^{53} + 1$ (–9007199254740991) and its upper bound is less than or equal to $2^{53} - 1$ (9007199254740991)

b) the effective value constraint of the base is the fixed value 2; and

c) the lower bound of the effective value constraint of the exponent is greater than or equal to –1074 and its upper bound is less than or equal to 971,

then the real value shall be encoded in the binary64 (double precision) floating-point format specified in IEEE 754.

12.4 Otherwise, the encoding of a real value shall consist of a length determinant (see 8.6) followed by a series of octets, which are the contents octets of DER encoding of the real value (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 11.3).

EXAMPLES

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-99999..99999), base (2), exponent (-20..20)})** is encoded in the single precision floating-point format of IEEE 754.. This real type includes the real value 0 but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-999999999999..999999999999), base (2), exponent (-20..20)})** is encoded in the double precision floating-point format of IEEE 754. This real

type includes the real value positive zero but does not include the special real values **-0**, **MINUS-INFINITY**, **PLUS-INFINITY**, and **NOT-A-NUMBER**.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-999999..999999), base (2), exponent (0..2000)})** is encoded as specified in this clause (12.4) because the exponent is not sufficiently constrained to allow the use of either IEEE 754 floating-point format.

The real type denoted by **REAL (0 | WITH COMPONENTS { mantissa (-99999..99999), base (10), exponent (-20..20)})** is encoded as specified in this clause (12.4) because the base is not constrained to 2.

13 Encoding of bitstring values

13.1 General

The encoding of a bitstring value depends on the effective size constraint of the bitstring type (see 8.2.8). If the lower and upper bounds of the effective size constraint are identical, 13.2 applies, otherwise 13.3 applies.

13.2 Encoding of bitstring types with a fixed size

13.2.1 The encoding of a bitstring value shall consist of a series of octets (see 13.2.2) with no length determinant.

13.2.2 The series of octets shall contain the bits of the bitstring value. These bits, starting with the leading bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first subsequent octet, followed by bits 8 to 1 of the second subsequent octet, followed by bits 8 to 1 of each octet in turn, followed by as many bits as are needed of the final subsequent octet, starting with bit 8. In the final subsequent octet there may be up to 7 unused bits. All the unused bits shall be set to 0.

NOTE – The terms leading bit and trailing bit are defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.2.

13.2.3 If the bitstring value is empty, the series of octets shall be empty.

13.2.4 When Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.7 applies (i.e., the bitstring type is defined with a "NamedBitList"), the bitstring value shall be encoded with trailing 0 bits added or removed as necessary to satisfy the effective size constraint.

13.3 Encoding of bitstring types with a variable size

13.3.1 The encoding of a bitstring value shall consist of a length determinant (see 8.6) followed by an initial octet (see 13.3.2) and by zero or more subsequent octets (see 13.3.3). The length indicated by the length determinant shall comprise both the initial octet and the subsequent octets.

NOTE – The length determinant indicates the number of octets following the length determinant itself, not the number of bits in the bitstring value.

13.3.2 The initial octet shall indicate how many unused bits (between 0 and 7) there are in the final subsequent octet. This number shall be encoded as an unsigned binary integer into 8 bits.

13.3.3 The subsequent octets shall contain the bits of the bitstring value. These bits, starting with the leading bit and proceeding to the trailing bit, shall be placed in bits 8 to 1 of the first subsequent octet, followed by bits 8 to 1 of the second subsequent octet, followed by bits 8 to 1 of each octet in turn, followed by as many bits as are needed of the final subsequent octet, starting with bit 8. In the final subsequent octet there may be up to 7 unused bits. All the unused bits shall be set to 0.

NOTE – The terms leading bit and trailing bit are defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.2.

13.3.4 If the bitstring value is empty, the initial octet shall be set to 0, and there shall be no subsequent octets.

NOTE – In CANONICAL-OER, where Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.7 applies, the bitstring value is encoded with any number of trailing 0 bits added or removed as necessary to ensure that the size of the encoding is the smallest size capable of carrying this value and satisfies the effective size constraint (see 31.6). This restriction does not apply to BASIC-OER.

14 Encoding of octetstring values

14.1 For an octetstring type in which the lower and upper bounds of the effective size constraint are identical, the encoding shall consist of the octets of the octetstring value (zero or more octets), with no length determinant.

14.2 For any other octetstring type, the encoding shall consist of a length determinant (see 8.6) followed by the octets of the octetstring value (zero or more octets).

15 Encoding of the null value

The encoding of the null value shall be empty.

16 Encoding of sequence values

16.1 The encoding of a sequence value shall consist of the following parts, in order:

a) preamble;

b) encodings of the components in the extension root;

c) extension addition presence bitmap (optional); and

d) encodings of the extension additions (optional).

NOTE – Each of these parts occupies a whole number of octets.

16.2 The preamble is specified in 16.2.1 to 16.2.4.

16.2.1 The preamble shall consist of the following parts, in order:

a) extension bit (optional);

b) root component presence bitmap (zero or more bits); and

c) unused bits (zero or more bits).

NOTE – Each part of the preamble does not, in general, occupy a whole number of octets, but the preamble as a whole does.

16.2.2 The extension bit shall be present (as bit 8 of the first octet of the preamble) if, and only if, the sequence type definition contains an extension marker in the "ComponentTypeLists" or in the "SequenceType" productions. If the extension bit is present, it shall be set to 1 when this sequence value contains one or more extension additions, otherwise it shall be set to 0.

16.2.3 The root component presence bitmap shall have one bit for each component that is marked **OPTIONAL** or **DEFAULT** (optional component) in the extension root of the sequence type. These bits shall, taken in order (proceeding from higher to lower bit numbers for each octet of the preamble from the first to the last), encode the presence or absence of each optional component in this sequence value, taken in order (proceeding from the first to the last component of the sequence type). Each bit shall be set to 1 when the corresponding optional component is present in this sequence value, and shall be set to 0 when the corresponding optional component is absent. The root component presence bitmap shall be empty when there are no optional components in the extension root of the sequence type.

16.2.4 The unused bits shall be the minimum number (possibly zero) of additional bits that make the size of the preamble a whole multiple of 8 bits. All the unused bits shall be set to 0.

NOTE – For a sequence type definition that has no extension marker and no components marked **OPTIONAL** or **DEFAULT**, the preamble will be empty.

16.3 The preamble shall be followed by the encoding of each of the components of the extension root of the sequence type that are present in this sequence value, taken in order.

16.4 The extension addition presence bitmap is specified in 16.4.1 to 16.4.3.

16.4.1 The extension addition presence bitmap shall be present if, and only if, the sequence type definition contains an extension marker and the extension bit in the preamble is set to 1. If present, it shall consist of a length determinant (see 8.6) followed by an initial octet (see 16.4.2) and by zero or more subsequent octets (see 16.4.3). The length indicated by the length determinant shall comprise both the initial octet and the subsequent octets.

16.4.2 The initial octet shall indicate how many unused bits (between 0 and 7) there are in the final subsequent octet. This number shall be encoded as an unsigned binary integer into 8 bits.

16.4.3 The subsequent octets shall contain one bit for each extension addition specified in the sequence type definition (regardless of any **OPTIONAL** or **DEFAULT** keywords present on the extension additions). These bits shall, taken in order (proceeding from higher to lower bit numbers for each subsequent octet from the first to the last), encode the presence or absence of each extension addition in this sequence value, in order (proceeding from the first to the last extension addition in the sequence type definition). Each bit shall be set to 1 when the corresponding extension addition is present in this sequence value, and shall be set to 0 when the corresponding extension addition is absent. In the final subsequent octet there may be up to 7 unused bits. All the unused bits shall be set to 0.

16.5 The extension addition presence bitmap shall be followed by the encoding of each of the extension additions of the sequence type that are present in this sequence value, taken in order, as specified in 16.5.1 to 16.5.3.

16.5.1 The value of each extension addition that is a "ComponentType" (i.e., not an "ExtensionAdditionGroup") shall be encoded as if it were contained in an open type (see clause 30).

16.5.2 The value of each extension addition that is an "ExtensionAdditionGroup" shall be encoded as if the "ExtensionAdditionGroup" were a sequence type contained in an open type (see clause 30).

NOTE – If an "ExtensionAdditionGroup" contains components marked **OPTIONAL** or **DEFAULT**, then the encoding of any value of that "ExtensionAdditionGroup" begins with a preamble consisting of a root component presence bitmap for those optional components.

16.5.3 If all the components within an "ExtensionAdditionGroup" are marked **OPTIONAL** or **DEFAULT** and those components are all absent in this sequence value, the bit in the extension addition presence bitmap corresponding to the "ExtensionAdditionGroup" shall be set to 0 and the encoding of the "ExtensionAdditionGroup" shall be empty.

17 Encoding of sequence-of values

17.1 The encoding of a sequence-of value shall consist of a quantity field (see 17.2) followed by the encodings of the occurrences of the component (see 17.3).

17.2 The quantity field shall be a non-negative integer value indicating the number of occurrences. This number shall be encoded as a length determinant (see 8.6) followed by a variable-size unsigned number (occupying at least as many whole octets as are necessary to carry the value).

NOTE – In CANONICAL-OER, the quantity value is encoded in the smallest possible number of octets (see 31.7). This restriction does not apply to BASIC-OER.

17.3 The quantity field shall be followed by the encoding of each occurrence of the component of the sequence-of type in this sequence-of value (zero or more occurrences), taken in order.

18 Encoding of set values

18.1 The value of a set type shall be encoded as if the type had been declared a sequence type, except that the components in the "RootComponentTypeList" of the set type (as well as the preamble bits) shall be encoded in the order specified in 18.2.

18.2 The components in the "RootComponentTypeList" shall be sorted into the canonical order specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 8.6, which depends on the tags of the components of the set type and does not depend on the textual ordering of the components. For the purposes of determining the order in which components are encoded when one or more components are untagged choice types, each untagged choice type shall be temporarily assigned a tag equal to the smallest tag in the "RootAlternativeTypeList" of that untagged choice type or of any untagged choice types nested within.

NOTE 1 – The components in the "ExtensionAdditionList" of the set type are not reordered and hence are encoded in the order in which they are defined.

NOTE 2 – The content of this clause is equivalent to Rec. ITU-T X.691 | ISO/IEC 8825-2, clause 21. The example in that clause is also relevant to this clause.

19 Encoding of set-of values

A value of a set-of type shall be encoded as if the type had been declared a sequence-of type.

20 Encoding of choice values

20.1 The encoding of a value of a choice type shall consist of the encoding of the outermost tag of the type of the chosen alternative as specified in 8.7, followed by the encoding of the value of the chosen alternative.

NOTE 1 – Since the outermost tags of the alternatives of a choice type are required to be all different (see Rec. ITU-T X.680 | ISO/IEC 8824-1, 29.3), the outermost tag is sufficient to identify the chosen alternative.

NOTE 2 – If the type of the chosen alternative has more than one tag as a result of explicit tagging, the tags following the outermost tag are not encoded.

NOTE 3 – If the type of the choice alternative is an untagged choice type, the outermost tag for that alternative will appear more than once in the encoding. This is different from how BER works.

20.2 If the choice type contains an extension marker in the "AlternativeTypeLists" and the chosen alternative is one of the extension additions, then the value of the chosen alternative shall be encoded as if it were contained in an open type (see clause 30), otherwise it shall be encoded normally.

21 Encoding of object identifier values

The encoding of an object identifier value shall consist of a length determinant (see 8.6) followed by a series of octets, which are the contents octets of BER encoding of the object identifier value (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.19).

22 Encoding of relative object identifier values

The encoding of a relative object identifier value shall consist of a length determinant (see 8.6) followed by a series of octets, which are the contents octets of BER encoding of the relative object identifier value (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.20).

23 Encoding of values of the internationalized resource reference type

The encoding of a value of the internationalized resource reference type shall consist of a length determinant (see 8.6) followed by a series of octets, which are the contents octets of BER encoding of the value (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.21).

24 Encoding of values of the relative internationalized resource reference type

The encoding of a value of the internationalized resource reference type shall consist of a length determinant (see 8.6) followed by a series of octets, which are the contents octets of BER encoding of the value (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.22).

25 Encoding of values of the embedded-pdv type

25.1 There are two ways in which an embedded-pdv type can be encoded:

a) The **`syntaxes`** alternative of the embedded-pdv type is constrained with an OER-visible inner type constraint to a single value or **`identification`** is constrained with an OER-visible inner type constraint to the **`fixed`** alternative, in which case only the **`data-value`** shall be encoded; this is called the "predefined" case.

b) An inner type constraint is neither employed to constrain the **`syntaxes`** alternative to a single value, nor to constrain **`identification`** to the **`fixed`** alternative, in which case both the **`identification`** and **`data-value`** shall be encoded; this is called the "general" case.

25.2 In the "predefined" case, the encoding of the value of the embedded-pdv type shall consist of the encoding of a value of the **`OCTET STRING`** type. The value of the **`OCTET STRING`** shall be the octets which form the complete encoding of the single data value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 36.3 a).

25.3 In the "general" case, the encoding of a value of the embedded-pdv type shall consist of the encoding of the type defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 36.5, with the **`data-value-descriptor`** element removed (that is, there shall be no presence bitmap at the head of the encoding of the **`SEQUENCE`**). The value of the **`data-value`** component (of type **`OCTET STRING`**) shall be the octets which form the complete encoding of the single data value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 36.3 a).

26 Encoding of values of the external type

26.1 The encoding of a value of the external type shall consist of the encoding of the sequence type specified in Rec. ITU-T X.691 | ISO/IEC 8825-2, 29.1.

26.2 Rec. ITU-T X.691 | ISO/IEC 8825-2, clauses 29.2 to 29.11 apply, with the following modifications:

a) The reference to "this Recommendation | International Standard" (meaning Rec. ITU-T X.691 | ISO/IEC 8825-2) present in those clauses shall be read as a reference to this Recommendation | International Standard (meaning Rec. ITU-T X.696 | ISO/IEC 8825-7).

b) The reference to Rec. ITU-T X.691 | ISO/IEC 8825-2, 11.2 (encoding of open type fields) present in those clauses shall be read as a reference to clause 30 of this Recommendation | International Standard.

27 Encoding of values of the restricted character string types

27.1 The encoding of a restricted character string type depends on whether the type is a known-multiplier character string type or not. The following types are known-multiplier character string types: **`IA5String`**, **`VisibleString`**, **`ISO646String`**, **`PrintableString`**, **`NumericString`**, **`BMPString`**, and **`UniversalString`**.

NOTE – In a known-multiplier character string type, the encoding of each character occupies a fixed number of octets, and therefore the number of octets occupied by the encoding of each character string value depends solely on the size (number of characters) of the character string value.

27.2 For a known-multiplier character string type in which the lower and upper bounds of the effective size constraint are identical, the encoding shall consist of the series of octets specified in 27.4, with no length determinant.

27.3 For every other character string type, the encoding shall consist of a length determinant (see 8.6) followed by the series of octets specified in 27.4.

NOTE – The length determinant indicates the number of octets following the length determinant itself, not the number of characters in the character string value.

27.4 The series of octets that encode a character string value is determined as follows:

a) For the **`IA5String`**, **`ISO646String`**, or **`VisibleString`** type, each character of the character string value shall be encoded in one octet with bit 8 set to 0.

b) For the **`NumericString`** or **`PrintableString`** type, the octets shall be the same as those specified for the encoding of an **`IA5String`** value consisting of the same characters.

c) For the **`BMPString`** type, the octets shall be those specified in ISO/IEC 10646, using the 2-octet BMP form (see 13.1 of ISO/IEC 10646); signatures shall not be used; control functions may be used provided they satisfy the restrictions specified in Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.23.9.

d) For the **`UniversalString`** type, the octets shall be those specified in ISO/IEC 10646, using the 4-octet canonical form (see 13.2 of ISO/IEC 10646); signatures shall not be used; control functions may be used provided they satisfy the restrictions specified in Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.23.9.

e) For the **`UTF8String`** type, the octets shall be those specified in ISO/IEC 10646, Annex D; announcers and escape sequences shall not be used; each character shall be encoded in the smallest number of octets available for that character.

f) For the remaining restricted character string types (**`TeletexString`**, **`T61String`**, **`VideotexString`**, **`GraphicString`**, and **`GeneralString`**), the octets shall be those specified in Rec. ITU-T X.690 | ISO/IEC 8825-1, 8.23.5.

27.5 For all restricted character string types except **`BMPString`**, **`UniversalString`**, and **`UTF8String`**, escape sequences and character codings registered in accordance with ISO/IEC 2375 may be used. The restrictions specified in Rec. ITU-T X.690 | ISO/IEC 8825-1, clauses 8.23.5.1 to 8.23.5.4, including Table 3, also apply here (the example below those clauses is not relevant).

28 Encoding of values of the unrestricted character string type

28.1 There are two ways in which an unrestricted character string type can be encoded:

a) The **`syntaxes`** alternative of the unrestricted character string type is constrained with an OER-visible inner type constraint to a single value or **`identification`** is constrained with an OER-visible inner type constraint to the **`fixed`** alternative, in which case only the **`string-value`** shall be encoded; this is called the "predefined" case.

b) An inner type constraint is neither employed to constrain the **`syntaxes`** alternative to a single value, nor to constrain **`identification`** to the **`fixed`** alternative, in which case both the **`identification`** and **`string-value`** shall be encoded; this is called the "general" case.

28.2 In the "predefined" case, the encoding of the value of the **`CHARACTER STRING`** type shall consist of the encoding of a value of the **`OCTET STRING`** type. The value of the **`OCTET STRING`** type shall be the octets which form the complete encoding of the character string value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.3 a).

28.3 In the "general" case, the encoding of a value of the **CHARACTER STRING** type shall consist of the encoding of the type defined in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.5, with the **data-value-descriptor** element removed (that is, there shall be no bitmap at the head of the encoding of the **SEQUENCE**). The value of the **string-value** component (of type **OCTET STRING**) shall be the octets which form the complete encoding of the character string value referenced in Rec. ITU-T X.680 | ISO/IEC 8824-1, 44.3 a).

29 Encoding of values of the time types

29.1 General

29.1.1 The encoding of the useful time types, the defined time types, and the additional time types shall be determined by the property settings of these types. Property settings of the useful and defined time types are specified in Rec. ITU-T X.680 | ISO/IEC 8824-1, 38.4 and Annex B, respectively. The property settings of the additional time types are determined by the property settings of the parent type, restricted by any OER-visible constraints that are applied (see 8.2.1 (d)).

29.1.2 If all of the following are true:

a) the **Basic** property setting of all the abstract values of the type to be encoded is **Date**, and

b) their **Date** property setting is one of those listed in column 2 of Table 1 (the same setting for all abstract values),

then the type shall be encoded as if it had been replaced by the type specified in the row of Table 1 corresponding to the **Date** setting.

29.1.3 If all of the following are true:

a) the **Basic** property setting of all the abstract values of the type is **Time**,

b) their **Time** property setting is one of those listed in column 2 of Table 2 (the same setting, including the same value of **n** in **HMSFn**, for all abstract values), and

c) their **Local-or-UTC** property setting is one of those listed in column 3 of Table 2 (the same setting for all abstract values),

then the type shall be encoded as if it had been replaced by the type specified in the row of Table 2 corresponding to the combination of **Time** and **Local-or-UTC** settings.

29.1.4 If all of the following are true:

a) the **Basic** property setting of all the abstract values of the type is **Interval**, and

b) their **Interval-type** property setting is that listed in column 2 of Table 3 (the same setting for all abstract values),

then the type shall be encoded as if it had been replaced by the type specified in the row of Table 3 corresponding to the **Interval-type** setting.

29.1.5 If all of the following are true:

a) the **Basic** property setting of all the abstract values of the type is **DateTime**,

b) their **Date** property setting is one of those listed in column 2 of Table 1 (the same setting for all abstract values),

c) their **Time** property setting is one of those listed in column 2 of Table 2 (the same setting, including the same value of **n** in **HMSFn**, for all abstract values), and

d) their **Local-or-UTC** property setting is one of those listed in column 3 of Table 2 (the same setting for all abstract values),

then the type shall be encoded as if it had been replaced by a sequence of two types, where the first type (encoding the date components of the abstract value) is that specified in the row of Table 1 corresponding to the **Date** setting, and the second type (encoding the time components of the abstract value) is that specified in the row of Table 2 corresponding to the combination of the **Time** and **Local-or-UTC** settings.

29.1.6 If the type does not satisfy the conditions specified in any of 29.1.2 to 29.1.5, it shall be encoded as a length determinant (see 8.6) followed by a series of octets, which are the contents octets of DER encoding of the value of the time type (see Rec. ITU-T X.690 | ISO/IEC 8825-1, 11.9).

NOTE – All the useful and defined time types satisfy the conditions specified in one of 29.1.2 to 29.1.5, and hence have optimized encodings. Additional time types may satisfy the conditions, but they are otherwise encoded as specified in 29.1.6. The unconstrained **TIME** type is always encoded as specified in 29.1.6.

29.1.7 The types specified in Tables 1, 2 and 3 are defined (using the ASN.1 notation) in 29.2, 29.3 and 29.4, respectively, and are assumed to be defined in an ASN.1 module with **AUTOMATIC TAGS**.

NOTE – The use of these type reference names in the specification of OER encodings does not make them available for use within an ASN.1 module, nor are they reserved words.

Table 1 – Optimized encoding of time subtypes with the "Basic=Date" property setting

Basic	Date	ASN.1 type to be encoded
Date	Y	YEAR-ENCODING (see 29.2.1)
	YM	YEAR-MONTH-ENCODING (see 29.2.2)
	YMD	DATE-ENCODING (see 29.2.3)

Table 2 – Optimized encoding of time subtypes with the "Basic=Time" property setting

Basic	Time	Local-or-UTC	ASN.1 type to be encoded
Time	H	L or Z	HOURS-ENCODING (see 29.3.1)
		LD	HOURS-DIFF-ENCODING (see 29.3.2)
	HM	L or Z	HOURS-MINUTES-ENCODING (see 29.3.3)
		LD	HOURS-MINUTES-DIFF-ENCODING (see 29.3.4)
	HMS	L or Z	TIME-OF-DAY-ENCODING (see 29.3.5)
		LD	TIME-OF-DAY-DIFF-ENCODING (see 29.3.6)
	HMSFn	L or Z	TIME-OF-DAY-FRACTION-ENCODING (see 29.3.7)
		LD	TIME-OF-DAY-FRACTION-DIFF-ENCODING (see 29.3.8)

Table 3 – Optimized encoding of time subtypes with the "Basic=Interval" property setting

Basic	Interval-type	ASN.1 type to be encoded
Interval	D	DURATION-INTERVAL-ENCODING (see 29.4.1)

29.2 Optimized encoding of time subtypes with the Basic=Date property setting

This clause defines the ASN.1 types referenced in Table 1.

29.2.1 The **YEAR-ENCODING** type is defined as follows:

```
YEAR-ENCODING ::= SEQUENCE {
      year                    INTEGER
  }
```

where **year** encodes the year component of the abstract value (positive, zero, or negative).

29.2.2 The **YEAR-MONTH-ENCODING** type is defined as follows:

```
YEAR-MONTH-ENCODING ::= SEQUENCE {
      year                    INTEGER,
      month                   INTEGER (1..12)
  }
```

where **year** and **month** encode the year and month components of the abstract value, respectively.

29.2.3 The **DATE-ENCODING** type is defined as follows:

```
DATE-ENCODING ::= SEQUENCE {
      year                    INTEGER,
      month                   INTEGER (1..12),
      day                     INTEGER (1..31)
  }
```

where **year**, **month**, and **day** encode the year, month, and day components of the abstract value, respectively.

29.3 Optimized encoding of time subtypes with the Basic=Time property setting

This clause defines the ASN.1 types referenced in Table 2.

29.3.1 The **HOURS-ENCODING** type is defined as follows:

```
HOURS-ENCODING ::= SEQUENCE {
      hours                   INTEGER (0..24)
  }
```

where **hours** encodes the hours components of the abstract value.

29.3.2 The **HOURS-DIFF-ENCODING** type is defined as follows:

```
HOURS-DIFF-ENCODING ::= SEQUENCE {
      hours                   INTEGER (0..24),
      minutes-diff            INTEGER (-900..900)
  }
```

where **hours** encodes the hours (local time) component of the abstract value and **minutes-diff** encodes the time-zone difference expressed in minutes (positive, zero, or negative).

29.3.3 The **HOURS-MINUTES-ENCODING** type is defined as follows:

```
HOURS-MINUTES-ENCODING ::= SEQUENCE {
      hours                   INTEGER (0..24),
      minutes                 INTEGER (0..59)
  }
```

where **hours** and **minutes** encode the hours and minutes components of the abstract value, respectively.

29.3.4 The **HOURS-MINUTES-DIFF-ENCODING** type is defined as follows:

```
HOURS-MINUTES-DIFF-ENCODING ::= SEQUENCE {
      hours                   INTEGER (0..24),
      minutes                 INTEGER (0..59),
      minutes-diff            INTEGER (-900..900)
  }
```

where **hours** and **minutes** encode the hours and minutes (local time) components of the abstract value, respectively, and **minutes-diff** encodes the time-zone difference expressed in minutes (positive, zero, or negative).

29.3.5 The **TIME-OF-DAY-ENCODING** type is defined as follows:

```
TIME-OF-DAY-ENCODING ::= SEQUENCE {
      hours                   INTEGER (0..24),
```



```
        minutes                 INTEGER (0..59),
        seconds                 INTEGER (0..60)
    }
```

where **hours**, **minutes**, and **seconds** encode the hours, minutes, and seconds components of the abstract value, respectively.

29.3.6 The **TIME-OF-DAY-DIFF-ENCODING** type is defined as follows:

```
TIME-OF-DAY-DIFF-ENCODING ::= SEQUENCE {
        hours                   INTEGER (0..24),
        minutes                 INTEGER (0..59),
        seconds                 INTEGER (0..60),
        minutes-diff            INTEGER (-900..900)
    }
```

where **hours**, **minutes** and **seconds** encode the hours, minutes, and seconds (local time) components of the abstract value respectively, and **minutes-diff** encodes the time-zone difference expressed in minutes (positive, zero or negative).

29.3.7 The **TIME-OF-DAY-FRACTION-ENCODING** type is defined as follows:

```
TIME-OF-DAY-FRACTION-ENCODING ::= SEQUENCE {
        hours                   INTEGER (0..24),
        minutes                 INTEGER (0..59),
        seconds                 INTEGER (0..60),
        fractional-part         INTEGER (0..MAX)
    }
```

where **hours**, **minutes** and **seconds** encode the hours, minutes and seconds components of the abstract value respectively, and **fractional-part** encodes the fractional seconds multiplied by 10^n, where n is the number of digits specified in the **Time=HMSFn** property setting.

29.3.8 The **TIME-OF-DAY-FRACTION-DIFF-ENCODING** type is defined as follows:

```
TIME-OF-DAY-FRACTION-DIFF-ENCODING ::= SEQUENCE {
        hours                   INTEGER (0..24),
        minutes                 INTEGER (0..59),
        seconds                 INTEGER (0..60),
        fractional-part         INTEGER (0..MAX),
        minutes-diff            INTEGER (-900..900)
    }
```

where **hours**, **minutes** and **seconds** encode the hours, minutes and seconds (local time) components of the abstract value respectively, **fractional-part** encodes the fractional seconds multiplied by 10^n, where n is the number of digits specified in the **Time=HMSFn** property setting, and **minutes-diff** encodes the time-zone difference expressed in minutes (positive, zero, or negative).

29.4 Optimized encoding of time subtypes with the Basic=Interval property setting

29.4.1 The **DURATION-INTERVAL-ENCODING** type, referenced in Table 3, is defined as follows:

```
DURATION-INTERVAL-ENCODING ::= SEQUENCE {
        years                   INTEGER (0..MAX) OPTIONAL,
        months                  INTEGER (0..MAX) OPTIONAL,
        weeks                   INTEGER (0..MAX) OPTIONAL,
        days                    INTEGER (0..MAX) OPTIONAL,
        hours                   INTEGER (0..MAX) OPTIONAL,
        minutes                 INTEGER (0..MAX) OPTIONAL,
        seconds                 INTEGER (0..MAX) OPTIONAL,
        fractional-part             SEQUENCE {
            number-of-digits        INTEGER (0..MAX),
            fractional-value        INTEGER (0..MAX)
        } OPTIONAL
    }
```

29.4.2 Each of the fields **years**, **months**, **weeks**, **days**, **hours**, **minutes** and **seconds** encodes the corresponding component of the abstract value.

29.4.3 The **weeks** field shall be present if and only if the **years**, **months**, **days**, **hours**, **minutes** and **seconds** fields are all absent.

29.4.4 Each of the fields **years**, **months**, **weeks**, **days**, **hours** and **minutes** shall not be present with a value of zero unless all the less significant components are absent.

29.4.5 The **number-of-digits** field of **fractional-part** encodes the number of decimal digits after the decimal point that constitute the fractional part.

29.4.6 The field **fractional-value** of **fractional-part** contains the integer value that occupies the decimal digits after the decimal point. The actual value of the fractional part (the non-integral number obtained by dividing **fractional-value** by 10 to the power of **number-of-digits**) belongs to the least significant field (among **years**, **months**, **weeks**, **days**, **hours**, and **minutes)** that is present.

30 Encoding of open type values

NOTE – An open type is an ASN.1 type that can take any abstract value of any ASN.1 type. Each value of an open type consists of:

a) a contained type; and

b) a value of the contained type.

The encoding of an open type value shall consist of a length determinant (see 8.6) followed by a series of octets, which are the encoding of the value of the contained type.

31 Canonical Octet Encoding Rules

31.1 All the requirements specified in clauses 8 to 30 for the Basic Octet Encoding Rules (BASIC-OER) also apply to the Canonical Octet Encoding Rules (CANONICAL-OER). This clause specifies additional requirements that apply only to CANONICAL-OER.

NOTE – These requirements remove all the encoder's options and ensure that for each abstract value there is only one possible encoding.

31.2 In the encoding of a type that contains a length determinant, when the length is less than 128 the short form of length determinant (see 8.6.4) shall be used. When the long form is used (see 8.6.5), the length shall be encoded in the smallest number of octets that can carry the encoded length.

NOTE – The above implies that the first subsequent octet of the long form is never set to 0.

31.3 In the encoding of a Boolean type, the value **TRUE** shall be encoded as 255 (see clause 9).

31.4 In the encoding of an integer type, in the cases in which the integer type is encoded as a variable-size signed or unsigned number (see 10.3 (e) and 10.4 (e)), the value shall be encoded in the smallest number of octets that can carry the encoded value.

31.5 In the encoding of an enumerated type, when the numeric value is greater or equal to zero and less than 128 the short form of enumerated type encoding (see 11.3) shall be used. When the long form is used (see 11.4), the numeric value shall be encoded in the smallest number of octets that can carry the encoded numeric value.

NOTE – The above implies that the first subsequent octet of the long form is never set to 0, and is not set to 'FF'H unless it is the only subsequent octet.

31.6 In the encoding of a bitstring type with a variable length, where Rec. ITU-T X.680 | ISO/IEC 8824-1, 22.7 applies, the bitstring value shall be encoded with any number of trailing 0 bits added or removed as necessary to ensure that the size of the encoding is the smallest size capable of carrying this value and satisfies the effective size constraint.

31.7 In the encoding of a sequence-of or set-of type, the value of the quantity field (immediately following the length determinant) shall be encoded in the smallest number of octets that can carry the encoded quantity value (see 17.2).

31.8 In the encoding of a set-of type, the encodings of the component values shall appear in ascending order, the component encodings being compared as octet strings, with 0-octets added to the shorter one if necessary to make the length equal to that of the longer one.

NOTE – Any padding octets added for the sort do not appear in the actual encoding.

31.9 In the encoding of a sequence or set type, each component that is marked **DEFAULT** shall be encoded as absent if its value is identical to the default value.