INTERNATIONAL STANDARD

ISO/IEC 24824-1

First edition
2007-05-01

Information technology — Generic applications of ASN.1: Fast Infoset

Technologies de l'information — Applications génériques de ASN.1: Infoset rapide

Reference number
ISO/IEC 24824-1:2007(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

CONTENTS

		Page
1	Scope	1
2	Normative references	1
	2.1 Identical Recommendations | International Standards	2
	2.2 Additional references	2
3	Definitions	3
	3.1 ASN.1 terms	3
	3.2 ECN terms	3
	3.3 ISO/IEC 10646 terms	3
	3.4 Additional definitions	3
4	Abbreviations	4
5	Notation	4
6	Principles of vocabulary table construction and use	5
7	ASN.1 type definitions	6
	7.1 General	6
	7.2 The `Document` type	6
	7.3 The `Element` type	11
	7.4 The `Attribute` type	12
	7.5 The `ProcessingInstruction` type	12
	7.6 The `UnexpandedEntityReference` type	13
	7.7 The `CharacterChunk` type	13
	7.8 The `Comment` type	14
	7.9 The `DocumentTypeDeclaration` type	14
	7.10 The `UnparsedEntity` type	15
	7.11 The `Notation` type	15
	7.12 The `NamespaceAttribute` type	16
	7.13 The `IdentifyingStringOrIndex` type	16
	7.14 The `NonIdentifyingStringOrIndex` type	17
	7.15 The `NameSurrogate` type	18
	7.16 The `QualifiedNameOrIndex` type	19
	7.17 The `EncodedCharacterString` type	20
8	Construction and processing of a fast infoset document	21
	8.1 Conceptual ordering of components of an abstract value of the `Document` type	22
	8.2 The restricted alphabet table	22
	8.3 The encoding algorithm table	22
	8.4 The dynamic string tables	23
	8.5 The dynamic name tables and name surrogates	23
9	Built-in restricted alphabets	24
	9.1 The "numeric" restricted alphabet	24
	9.2 The "date and time" restricted alphabet	24
10	Built-in encoding algorithms	24
	10.1 General	24
	10.2 The "hexadecimal" encoding algorithm	25
	10.3 The "base64" encoding algorithm	25
	10.4 The "short" encoding algorithm	25
	10.5 The "int" encoding algorithm	26
	10.6 The "long" encoding algorithm	26
	10.7 The "boolean" encoding algorithm	26
	10.8 The "float" encoding algorithm	27
	10.9 The "double" encoding algorithm	27
	10.10 The "uuid" encoding algorithm	27

Page

10.11 The "cdata" encoding algorithm 28
11 Restrictions on the supported XML infosets and other simplifications 28
12 Bit-level encoding of the Document type 29
Annex A – ASN.1 module and ECN modules for fast infoset documents 31
A.1 ASN.1 module definition 31
A.2 ECN module definitions 33
Annex B – The MIME media type for fast infoset documents 53
Annex C – Description of the encoding of a fast infoset document 55
C.1 Fast infoset document 55
C.2 Encoding of the Document type 55
C.3 Encoding of the Element type 57
C.4 Encoding of the Attribute type 58
C.5 Encoding of the ProcessingInstruction type 58
C.6 Encoding of the UnexpandedEntityReference type 59
C.7 Encoding of the CharacterChunk type 59
C.8 Encoding of the Comment type 59
C.9 Encoding of the DocumentTypeDeclaration type 59
C.10 Encoding of the UnparsedEntity type 60
C.11 Encoding of the Notation type 60
C.12 Encoding of the NamespaceAttribute type 61
C.13 Encoding of the IdentifyingStringOrIndex type 61
C.14 Encoding of the NonIdentifyingStringOrIndex type starting on the first bit of an octet 61
C.15 Encoding of the NonIdentifyingStringOrIndex type starting on the third bit of an octet 62
C.16 Encoding of the NameSurrogate type 62
C.17 Encoding of the QualifiedNameOrIndex type starting on the second bit of an octet 62
C.18 Encoding of the QualifiedNameOrIndex type starting on the third bit of an octet 63
C.19 Encoding of the EncodedCharacterString type starting on the third bit of an octet 63
C.20 Encoding of the EncodedCharacterString type starting on the fifth bit of an octet 64
C.21 Encoding of the length of a sequence-of type 64
C.22 Encoding of the NonEmptyOctetString type starting on the second bit of an octet 64
C.23 Encoding of the NonEmptyOctetString starting on the fifth bit of an octet 65
C.24 Encoding of the NonEmptyOctetString type starting on the seventh bit of an octet 65
C.25 Encoding of integers in the range 1 to 2^{20} starting on the second bit of an octet 65
C.26 Encoding of integers in the range 0 to 2^{20} starting on the second bit of an octet 66
C.27 Encoding of integers in the range 1 to 2^{20} starting on the third bit of an octet 66
C.28 Encoding of integers in the range 1 to 2^{20} starting on the fourth bit of an octet 66
C.29 Encoding of integers in the range 1 to 256 67
Annex D – Examples of encoding XML infosets as fast infoset documents 68
D.1 Introduction of examples 68
D.2 Size of example documents (including redundancy-based compression) 68
D.3 UBL order example 69
D.4 UBL Order fast infoset document with an external vocabulary 71
D.5 UBL order fast infoset document without an initial vocabulary 79
Annex E – Assignment of object identifier values 90
BIBLIOGRAPHY 91

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. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

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.

ISO/IEC 24824-1 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 Rec. X.891.

ISO/IEC 24824 consists of the following parts, under the general title *Information technology — Generic applications of ASN.1*:

— *Part 1: Fast infoset*

— *Part 2: Fast Web Services*

The following part is under preparation:

— *Part 3: Fast infoset security*

Introduction

This Recommendation | International Standard specifies a representation of an instance of the W3C XML Information Set using binary encodings (specified using the ASN.1 notation and the ASN.1 Encoding Control Notation). The encoding specified in this edition of this Recommendation | International Standard is identified by the version number 1 (see 12.9).

The technology specified in this Recommendation | International Standard is named Fast Infoset. It provides an alternative to W3C XML syntax as a means of representing instances of the W3C XML Information Set. This representation generally provides smaller encoding sizes and faster processing than a W3C XML representation.

The representation of an instance of the W3C XML Information Set specified in this Recommendation | International Standard is called a fast infoset document. Each fast infoset document is an encoding of an abstract value of an ASN.1 data type (the **`Document`** type – see 7.2) representing an instance of the W3C XML Information Set.

This Recommendation | International Standard specifies the use of several techniques that minimize the size of a fast infoset document and that maximize the speed of creating and processing such documents.

These techniques are based on the use of vocabulary tables, which allow typically-small integer values (vocabulary table indexes) to be used instead of character strings that form (for example) the names of elements or attributes in an XML 1.0 serialization of an instance of the W3C XML Information Set.

There are a number of vocabulary tables (see clause 8), of which the most basic (the eight character string tables) map typically-small integers to strings of characters. There are, however, also vocabulary tables (the element name table and the attribute name table) that provide a further level of indirection, with a vocabulary table index mapping to a set of three vocabulary table indexes, identifying a prefix, a namespace name, and a local name.

Another important technique is the use of a restricted alphabet vocabulary table. This contains entries that list a subset of ISO/IEC 10646 characters. If a character string needs to be encoded for which there is an entry in this table, then it can be encoded by identifying that this vocabulary table is being used, giving the vocabulary table index, and then encoding each character in the minimum number of bits needed for that particular subset of ISO/IEC 10646 characters. There are a number of built-in restricted alphabets that always form the first few entries of this table, covering such commonly occurring strings as dates and times, and numeric values.

A further important optimization uses the encoding algorithm vocabulary table. This table identifies specialized encodings that can be employed for commonly occurring strings, again with a number of built-in algorithms. For example, if there is a string which looks like the decimal representation of an integer in the range –32768 to 32767, then that string can be encoded by identifying that this vocabulary table is being used, giving the vocabulary table index, and then encoding the integer as a two-octet signed integer. Floating-point numbers and arrays of such numbers are supported in the same way.

In order to ensure fast processing without sacrificing compactness, many components of a fast infoset document (such as character strings and components representing information items of the XML infoset) are octet-aligned, while other components (such as lengths and vocabulary table indexes) are not necessarily octet-aligned but always end on the last bit of an octet. To provide a formal specification of these optimized encodings, the ASN.1 Encoding Control Notation (defined in ITU-T Rec. X.692 | ISO/IEC 8825-3) is used (see A.2), but use of ECN tools for implementation is not necessary and a complete description of the encoding is provided (see Annex C).

The vocabulary tables for a particular fast infoset document can be initialized by information at the head of the document, and are normally added to dynamically, providing flexibility for an encoder. The initial vocabulary tables can be provided by a reference to the set of final vocabulary tables of some other identified fast infoset document (or by other means). This vocabulary reference can then be supplemented by further table additions to provide the initial vocabulary tables for this document. Further dynamic additions are normally made to the tables during the creation or the processing of the document.

Finally, a mechanism is provided for the generator of a fast infoset document to include data (called additional processing data) related to optional additional processing of the fast infoset document, together with a URI that identifies a complete specification of the form and semantics of that additional processing data. The optional additional processing data is ignored by any subsequent processor of the fast infoset document if the URI is not known, or the processing that it specifies is not supported or not required.

NOTE – An example of such additional processing data would be data that provides indexes that enable immediate access to parts of the fast infoset document, so that the whole document need not be processed if the only interest is in those parts of the fast infoset document that correspond to a specific XML tag.

Annex A forms an integral part of this Recommendation | International Standard, and contains an ASN.1 module (see ITU-T Rec. X.680 | ISO/IEC 8824-1) and two ECN modules (EDM and ELM – see ITU-T Rec. X.692 | ISO/IEC 8825-3) which together specify the abstract content and the bit-level encoding of a value of the **`Document`** type, which conveys the value of an instance of the W3C XML Information Set.

Annex B forms an integral part of this Recommendation | International Standard, and contains the specification of a MIME media type identifying a fast infoset document.

Annex C does not form an integral part of this Recommendation | International Standard, and provides a complete description of the encodings formally specified in clause 12 and A.2.

Annex D does not form an integral part of this Recommendation | International Standard, and provides examples of fast infoset documents generated from some XML documents. Annex D also gives the size of the XML representation and the Fast Infoset representation of these examples.

INTERNATIONAL STANDARD
ITU-T RECOMMENDATION

Information technology – Generic applications of ASN.1: Fast infoset

1 Scope

This Recommendation | International Standard specifies an ASN.1 type (see ITU-T Rec. X.680 | ISO/IEC 8824-1) whose abstract values represent instances of the W3C XML Information Set. It also specifies binary encodings for those values, using ASN.1 Encoding Control Notation (see ITU-T Rec. X.692 | ISO/IEC 8825-3).

NOTE – These encodings are called fast infoset documents.

This Recommendation | International Standard also specifies techniques that:

- minimize the size of fast infoset documents;
- maximize the speed of creating and processing fast infoset documents;
- allow the specification (by the generator of a fast infoset document) of additional processing data.

The first two techniques involve the use of conceptual vocabulary tables. The set of vocabulary tables and the nature of their entries is fully defined in this Recommendation | International Standard, but their representation in computer memory is outside the scope of this Recommendation | International Standard. Provision for transfer or storage of, or a formal notation for displaying or specifying, vocabulary tables to be used as an external vocabulary is also outside the scope of this Recommendation | International Standard.

The third technique involves the provision of additional processing data and a URI that identifies the form and semantics of that data. The specification of specific forms of additional processing data and their use is outside the scope of this Recommendation | International Standard.

URIs can be used to identify final vocabularies that can be used as either part or all of some new initial vocabulary, but the assignment of specific URIs to specific final vocabularies is outside the scope of this Recommendation | International Standard.

This Recommendation | International Standard specifies built-in restricted alphabets, the addition to vocabulary tables of further restricted alphabets by enumeration, and the use of these vocabulary tables for efficient encoding of character strings.

This Recommendation | International Standard further specifies built-in encoding algorithms for the optimum encoding of certain character strings, and the addition to vocabulary tables of further encoding algorithms identified by URIs, but the definition of these further encoding algorithms and their associated URIs is outside the scope of this Recommendation | International Standard.

In addition, this Recommendation | International Standard specifies a Multipurpose Internet Mail Extensions (MIME) media type that identifies a fast infoset document.

2 Normative references

The following Recommendations, International Standards and other references 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, International Standards and other references 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, International Standards and other references listed below. The Telecommunication Standardization Bureau of the ITU maintains a list of currently valid ITU-T Recommendations. Members of IEC and ISO maintain registers of currently valid International Standards. The IETF maintains a list of RFCs, together with those that have been obsoleted by later RFCs. The W3C maintains a list of currently valid W3C Recommendations. The reference to a document within this Recommendation | International Standard does not give it, as a stand-alone document, the status of a Recommendation or International Standard.

2.1 Identical Recommendations | International Standards

- ITU-T Recommendation X.667 (2004) | ISO/IEC 9834-8:2005, *Information technology – Open Systems Interconnection – Procedures for the operation of OSI Registration Authorities: Generation and registration of Universally Unique Identifiers (UUIDs) and their use as ASN.1 Object Identifier components.*
- ITU-T Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.*
- ITU-T Recommendation X.681 (2002) | ISO/IEC 8824-2:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Information object specification.* †
- ITU-T Recommendation X.682 (2002) | ISO/IEC 8824-3:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Constraint specification.* †
- ITU-T Recommendation X.683 (2002) | ISO/IEC 8824-4:2002, *Information technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 specifications.* †
- ITU-T Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).* †
- ITU-T Recommendation X.691 (2002) | ISO/IEC 8825-2:2002, *Information technology – ASN.1 encoding rules: Specification of Packed Encoding Rules (PER).* †
- ITU-T Recommendation X.692 (2002) | ISO/IEC 8825-3:2002, *Information technology – ASN.1 encoding rules: Specification of Encoding Control Notation (ECN).*
- ITU-T Recommendation X.693 (2001) | ISO/IEC 8825-4:2002, *Information technology – ASN.1 encoding rules: XML Encoding Rules (XER).* †

NOTE – The complete set of ASN.1 Recommendations | International Standards are listed above, as they can all be applicable in particular uses of this Recommendation | International Standard. Where these are not directly referenced in the body of this Recommendation | International Standard, a † symbol is added to the reference.

2.2 Additional references

- ISO 8601:2004, *Data elements and interchange formats – Information interchange – Representation of dates and times.*
- ISO/IEC 10646:2003, *Information technology – Universal Multiple-Octet Coded Character Set (UCS).*
- *The Unicode Standard, Version 4.0*, The Unicode Consortium *(Reading, MA, Addison-Wesley).*

  NOTE 1 – The graphics characters (and their encodings) defined by Unicode are identical to those defined by ISO/IEC 10646-1, but Unicode is included as a reference because it also specifies the names of control characters and defines the abbreviation UTF-16BE.
- W3C XML 1.0:2004, *Extensible Markup Language (XML) 1.0 (Third Edition), W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml-20040204/.*
- W3C XML 1.1:2004, *Extensible Markup Language (XML) 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2000/REC-xml11-20040204/.*

  NOTE 2 – References to both W3C XML 1.0 and W3C XML 1.1 are included as neither is a subset of the other. These references are used solely in 3.4.10.
- W3C XML Information Set:2004, *XML Information Set (Second Edition), W3C Recommendation, Copyright © [04 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2004/REC-xml-infoset-20040204/.*
- W3C XML Namespaces 1.0:1999, *Namespaces in XML, W3C Recommendation, Copyright © [14 January 1999] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/1999/REC-xml-names-19990114/.*
- W3C XML Namespaces 1.1:2004, *Namespaces in XML 1.1, W3C Recommendation, Copyright © [4 February 2004] World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University), http://www.w3.org/TR/2004/REC-xml-l-names11-20040204/.*

NOTE 3 – References to both W3C XML Namespaces 1.0 and W3C XML Namespaces 1.1 are included as neither is a subset of the other. These references are used solely in 3.4.10.

– IETF RFC 2045 (1996), *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.*

– IETF RFC 2396 (1998), *Uniform Resource Identifiers (URI): Generic Syntax.*

– IEEE 754-1985, *IEEE Standard for Binary Floating-Point Arithmetic.*

3 Definitions

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

3.1 ASN.1 terms

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

a) choice type;

b) sequence type;

c) sequence-of type.

3.2 ECN terms

This Recommendation | International Standard uses the following terms defined in ITU-T Rec. X.692 | ISO/IEC 8825-3:

a) Encoding Definition Modules (EDM);

b) Encoding Link Module (ELM).

3.3 ISO/IEC 10646 terms

This Recommendation | International Standard uses the following term defined in ISO/IEC 10646:

a) Basic Multilingual Plane.

3.4 Additional definitions

3.4.1 Base64: An encoding mechanism that represents an octet string value as a character string using a restricted alphabet of 65 characters (see 10.3 and IETF RFC 2045).

3.4.2 character string: A string of ISO/IEC 10646 abstract characters, without any implication on the way they are encoded.

3.4.3 encoding algorithm: A precise specification of how to efficiently encode a character string with specified characteristics into octets.

NOTE – An example is the encoding of a string such as "-32176" into a two's complement binary integer in two octets. The two-octet encoding would be accompanied by a vocabulary table index identifying this encoding algorithm.

3.4.4 external vocabulary: A set of vocabulary tables referenced by a URI (see 7.2.14).

3.4.5 fast infoset document: An XML infoset represented as specified in this Recommendation | International Standard.

3.4.6 final vocabulary: The content of the vocabulary tables at the end of the creation or of the processing of a fast infoset document.

3.4.7 information item: Each of the kinds of items that constitute an XML infoset.

3.4.8 initial vocabulary: The set of vocabulary tables established by information at the head of a fast infoset document that optionally references an external vocabulary and optionally provides additional table entries.

3.4.9 name surrogate: A set of three vocabulary table indexes (the first two optional) that are used to represent a qualified name (see 3.4.11).

3.4.10 namespace-well-formed XML document: Either a W3C XML 1.0 document that is well-formed according to W3C XML Namespaces 1.0, or a W3C XML 1.1 document that is well-formed according to W3C XML Namespaces 1.1.

3.4.11 qualified name: The set consisting of the **[prefix]**, **[namespace name]**, and **[local name]** properties of an **element** information item or **attribute** information item.

3.4.12 restricted alphabet: An ordered set of distinct ISO/IEC 10646 characters, which permits a compact encoding of any character string that consists entirely of characters from that set.

3.4.13 vocabulary table index: A positive integer value identifying an entry in a vocabulary table.

3.4.14 vocabulary tables: A set of conceptual tables (typically, but not necessarily, dynamically constructed) associated with a fast infoset document, which contain character strings or other information, and support the use of typically-small positive integer values (vocabulary table indexes) identifying their entries.

NOTE – Examples of vocabulary tables are those containing character strings that are the **[local name]** property of **attribute** or **element** information items, or character strings corresponding to sequences of **character** information items that are members of the **[children]** property of **element** information items.

3.4.15 XML declaration: The UTF-8 encoding of a specified character string (see also 12.3) that may be included at the beginning of a fast infoset document to identify the encoding as a fast infoset document and to distinguish it from a W3C XML 1.0 or W3C XML 1.1 document.

3.4.16 XML infoset: An abstract data set describing the information in a namespace-well-formed XML document, as specified in W3C XML Information Set.

3.4.17 XML whitespace: One or more of the characters HORIZONTAL TABULATION (9), LINE FEED (10), CARRIAGE RETURN (13), or SPACE (32) of Unicode.

NOTE – These characters are those that match the production "S" in both W3C XML 1.0 and W3C XML 1.1 (see W3C XML 1.0, 2.3 and W3C XML 1.1, 2.3). The characters NEXT LINE (133) and LINE SEPARATOR (8232), which may occur in a namespace-well-formed W3C XML 1.1 document (see W3C XML 1.1, 2.11), are converted to LINE FEED characters by end-of-line handling (see W3C XML 1.1, 2.11). If those characters occur in an XML infoset generated from a namespace-well-formed W3C XML 1.1 document, they are not XML whitespace.

4 Abbreviations

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

ASN.1	Abstract Syntax Notation One
BMP	Basic Multilingual Plane
ECN	Encoding Control Notation
MIME	Multipurpose Internet Mail Extensions
UBL	Universal Business Language
URI	Uniform Resource Identifier
UTF-8	Universal Transformation Function 8-bit (see ISO/IEC 10646, Annex D)
UTF-16BE	Universal Transformation Function 16-bit Big Endian (see Unicode, 2.6)
UUID	Universally Unique Identifier
XML	eXtensible Markup Language

5 Notation

5.1 This Recommendation | International Standard uses the ASN.1 notation defined by ITU-T Rec. X.680 | ISO/IEC 8824-1 for the formal definition of data types whose encodings are fast infoset documents.

NOTE – Clause 12 specifies the application of ITU-T Rec. X.692 | ISO/IEC 8825-3 to the ASN.1 type definitions, providing the bit-level encoding of a fast infoset document.

5.2 In this Recommendation | International Standard, **`bold Courier`** is used for ASN.1 notation and **bold Arial** is used for W3C XML syntax and for the names of information items of the XML Information Set.

5.3 The names of information items' properties are in **bold Arial** and enclosed between square brackets (for example, **[children]**).

5.4 The names of categories of character strings (see 8.4.2) and the names of categories of qualified names (see 8.5.4) are in UPPERCASE.

5.5 In this Recommendation | International Standard, bit positions within an octet are specified using the terminology first bit, second bit, etc., to eighth bit, where the first bit is the most significant bit of the octet, and the eighth bit is least significant bit of the octet.

6 Principles of vocabulary table construction and use

6.1 Vocabulary tables are conceptual tables mapping a vocabulary table index into a vocabulary table entry.

NOTE – The representation of vocabulary tables in computer memory is not defined, nor is the means by which an implementation maps a vocabulary table index into a vocabulary table entry for that table.

6.2 The creator of a fast infoset document from an XML infoset determines the contents of the vocabulary tables.

6.3 In the most general case, the head of a fast infoset document can reference a set of vocabulary tables (an external vocabulary), followed by the specification of additions to those vocabulary tables to form the initial vocabulary for this fast infoset document. Further additions to the vocabulary tables occur during the creation and during the processing of a fast infoset document, so that they incrementally grow to form the final vocabulary tables for that document.

6.4 Some vocabulary tables incrementally grow from an initial vocabulary to a final vocabulary during the creation and during the processing of a fast infoset document, and therefore have the word "dynamic" in the name of the vocabulary table. There are no mechanisms for entries to be removed from any table.

6.5 Vocabulary table indexes are implicitly assigned. The first entry to any vocabulary table has a vocabulary table index of one, and each subsequent entry to that table has the next higher integer value for the vocabulary table index. Where this Recommendation | International Standard specifies that something is to be added to a vocabulary table, this implies that the next available vocabulary table index shall be assigned.

NOTE – Vocabulary table indexes start at one and not zero because the value zero (when permitted) has the special meaning of "empty character string" in a field that might otherwise hold a vocabulary table index.

6.6 In order to support this implicit assignment of vocabulary table indexes, the conceptual order of processing the components (at any depth) of a fast infoset document is fully-defined (see 8.1).

NOTE – This order is the same as the order of the encodings of the components in a fast infoset document. It does not necessarily imply that the semantics carried by the document is processed in this order. The order is defined solely for the purposes of ensuring that the same vocabulary table index is assigned for any given vocabulary table entry by both the creator and the processor of a fast infoset document.

6.7 Vocabulary tables are used for many purposes (see clause 8), but their primary function is to enable the use of a vocabulary table index instead of a vocabulary table entry, where such indexes are smaller (and may be faster to process) than the table entry. A number of built-in entries for some vocabulary tables are specified in clause 9. These entries are always implicitly present in these vocabulary tables, with the vocabulary table indexes specified in clause 9.

6.8 For some categories of character string, the creator of a fast infoset document has the option of adding or not adding a string to a vocabulary table, depending on the expected (or known) number of occurrences of that character string in the XML infoset.

6.9 The precise form and meaning of vocabulary table entries is specified in clause 8, but they are in most cases variable length character strings, often short, but potentially as large as 2^{32} octets.

6.10 A conforming creator of a fast infoset document is required to do all the additions to the vocabulary tables as specified in 7.13.7, 7.14.6, 7.14.7, and 7.16.7. This ensures that the number of vocabulary table entries in each vocabulary table never exceeds 2^{20}.

NOTE – A vocabulary table entry may equal one or more other vocabulary table entries. This is in order to allow efficient creation of fast infoset documents. However, duplicate entries will decrease the efficiency of transfer. A processor is not affected by duplicate entries.

6.11 A conforming processor of a fast infoset document is required to do all the additions to the vocabulary tables as specified in 7.13.8, 7.14.11, and 7.16.8. This ensures that the restriction of 6.10 a has not been violated.

7 ASN.1 type definitions

7.1 General

7.1.1 This Recommendation | International Standard specifies a set of ASN.1 types supporting a representation of the XML Information Set. The root type of this set of types is the **Document** type.

7.1.2 Some restrictions are imposed on the content of the XML infosets and some simplifications are made in the representation (see clause 11) in order to improve the usability of the specification and the efficiency of the encodings produced with it.

NOTE – An XML infoset that does not meet those restrictions cannot be represented as a fast infoset document, nor can it normally be represented as a namespace-well-formed XML document.

7.1.3 For each kind of information item specified in W3C XML Information Set, a corresponding ASN.1 type definition is provided in this Recommendation | International Standard. This type definition is always a sequence type, with components corresponding to the properties of the information item.

7.1.4 Certain properties of information items are not included in the ASN.1 type definitions (see 11.4).

7.1.5 In some cases, the value of a property that is not included in the ASN.1 type definitions can be determined from the value of other properties of the same or other information items that are included. In these cases, the omission of that property simplifies the representation with no loss of information. There are, however, a few cases in which the value of a property that is not included cannot be determined from other properties. In all such cases, the omission of that property is a simplification that does not limit the utility of the specification for most practical use cases.

7.1.6 Clause 12 specifies the encoding of the **Document** type.

7.2 The Document type

7.2.1 The **Document** type is:

```
Document ::= SEQUENCE {
      additional-data          SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        -- If the initial-vocabulary component is present, at least
                        -- one of its components shall be present -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
```

```
character-encoding-scheme  NonEmptyOctetString OPTIONAL,
standalone                 BOOLEAN OPTIONAL,
version                    NonIdentifyingStringOrIndex OPTIONAL
                             -- OTHER STRING category --,
children                   SEQUENCE (SIZE(0..MAX)) OF
      CHOICE {
            element                    Element,
            processing-instruction     ProcessingInstruction,
            comment                    Comment,
            document-type-declaration DocumentTypeDeclaration }}
```

where the value **one-meg** is:

```
one-meg INTEGER ::= 1048576 -- Two to the power 20
```

The **NonEmptyOctetString** type is:

```
NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))
```

where the value **four-gig** is:

```
four-gig INTEGER ::= 4294967296 -- Two to the power 32
```

The **URI** type is:

```
URI ::= NonEmptyOctetString
```

7.2.2 The **EncodedCharacterString**, **NameSurrogate**, **Notation**, **UnparsedEntity**, **NonIdentifyingStringOrIndex**, **Element**, **ProcessingInstruction**, **Comment**, and **DocumentTypeDeclaration** types are defined in 7.17, 7.15, 7.11, 7.10, 7.14, 7.3, 7.5, 7.8, and 7.9 respectively.

7.2.3 The **URI** type shall be a URI as specified in IETF RFC 2396.

7.2.4 The component **restricted-alphabets** of **initial-vocabulary** (if present) shall carry one or more character strings, each holding the characters of a restricted alphabet. Each character string shall contain at least two characters, and all characters in the character string shall be distinct.

NOTE – The use of a restricted alphabet to optimize encodings of character strings is specified in 7.17.6.

7.2.5 The component **encoding-algorithms** of **initial-vocabulary** (if present) shall carry one or more URIs each identifying an encoding algorithm.

NOTE – There are built-in encoding algorithms defined in this Recommendation | International Standard (see clause 10), with specified vocabulary table indexes, but it is out of the scope of this Recommendation | International Standard to define further encoding algorithms and their associated URIs, nor is the means of defining such algorithms determined here. The information needed to define an encoding algorithm is specified in 8.3.3.

7.2.6 The **Document** type represents the **document** information item of an XML infoset. Since all other information items in an XML infoset are either properties of this information item or properties of an item that is a child or descendant of this item (at any depth), each **Document** represents a complete XML infoset.

NOTE – Each **Document** without a reference to an external vocabulary (see 7.2.13) also defines a final vocabulary that can be used as the external vocabulary of some other fast infoset document.

7.2.7 The **additional-data** component (if present) shall carry one or more **additional-datum** components to permit additional mechanisms for the processing of a fast infoset document.

NOTE 1 – An example would be data that enables a processor to access parts of a fast infoset document without requiring the processing of the whole document. The form of such data is not standardized.

NOTE 2 – The number of **additional-datum** components is restricted to 2^{20} components (see 7.2.1).

7.2.8 Each **additional-datum** shall consist of:

a) the **id** component (a value of the **URI** type); the URI shall reference a specification that defines the form and semantics of the **data** component; and

NOTE – The form of the **additional-datum** may be specified as an abstract type in conjunction with an encoding rule, or by any other suitable means.

b) the **data** component, which is an octet string that holds the additional processing data.

7.2.9 The use of an **additional-data** component is subject to the following:

a) an **additional-datum** component can be ignored by a processor unless the URI is recognized and the additional processing is considered relevant for the activity of that processor;

b) a processor that ignores all **additional-datum** components is nonetheless capable of generating an XML infoset that is equivalent to the XML infoset used to generate the fast infoset document.

7.2.10 Multiple **`additional-datum`** components with the same URI may be present, and will be processed in accordance with the specification associated with the URI.

7.2.11 The **`initial-vocabulary`** component provides data that (together with some built-in table entries) completely determines the initial content of the restricted alphabet table (see 8.2), the encoding algorithm table (see 8.3), the dynamic string tables (see 8.4), and the dynamic name tables (see 8.5) of this fast infoset document (the initial vocabulary of the fast infoset document). An initial vocabulary consists of the following data:

a) an ordered set of restricted alphabets (see 8.2.2), containing at least the built-in restricted alphabets (see clause 9);

b) an ordered set of encoding algorithms (see 8.3.2), containing at least the built-in encoding algorithms (see clause 10);

c) eight independent ordered sets of character strings, corresponding to the eight categories of character strings specified in this Recommendation | International Standard (see 8.4.2), with each set containing zero or more character strings of a single category; and

d) two independent ordered sets of name surrogates (see 8.5.2), corresponding to the two categories of qualified names specified in this Recommendation | International Standard (see 8.5.4), with each set containing zero or more name surrogates of a single category.

NOTE – An initial vocabulary cannot be totally empty, because it always contains (at least) the built-in restricted alphabets and the built-in encoding algorithms. However, it would not be unusual for a fast infoset document to have an initial vocabulary that contains only that data, as the decision (by the creator of a fast infoset document) on how to use the **`initial-vocabulary`** component is implementation-dependent, and some implementations may choose to add all vocabulary table entries dynamically (within the body of the fast infoset document).

7.2.12 The initial vocabulary of the fast infoset document shall be determined as follows:

a) If the **`initial-vocabulary`** component is absent, then the initial vocabulary shall consist solely of the built-in table entries specified in 7.2.21, 7.2.22, and clauses 9 and 10.

b) If the **`initial-vocabulary`** component is present, and the **`external-vocabulary`** component is absent, then the initial vocabulary shall consist of the built-in table entries specified in 7.2.21, 7.2.22, and clauses 9 and 10, with the added table entries (if any) determined by 7.2.16.

c) If the **`initial-vocabulary`** component is present, and the **`external-vocabulary`** component is present, then the initial vocabulary shall consist of the final vocabulary identified by the **`external-vocabulary`** component as specified in 7.2.13 and 7.2.14, with the added vocabulary table entries (if any) determined by 7.2.16.

7.2.13 The **`external-vocabulary`** component identifies a final vocabulary using one of the mechanisms specified in 7.2.14. The URI type (see 7.2.1) determines the final vocabulary to be used as the external vocabulary in one of three ways (see 7.2.14).

NOTE – This Recommendation | International Standard does not specify any external vocabularies and any URIs that reference external vocabularies. Such external vocabularies and URIs can be defined by any authority able to allocate those URIs, and may be privately agreed or may be the subject of standardization.

7.2.14 An external vocabulary can be specified in one of three ways:

a) as the final vocabulary of a fast infoset document, which shall not itself reference an external vocabulary; or

NOTE 1 – It is an implementation matter whether the final vocabulary is stored locally or whether only the fast infoset document is stored and the final vocabulary is generated by processing it.

NOTE 2 – The restriction that the final vocabulary of a fast infoset document with a reference to an external vocabulary cannot itself be used as an external vocabulary is imposed in order to simplify implementation and avoid circularity of references.

b) as a namespace-well-formed XML document, which is conceptually processed as follows:

1) the XML infoset of the namespace-well-formed XML document shall be determined; and

2) the fast infoset document for this XML infoset shall be created as specified in this Recommendation | International Standard, but it shall have no **`initial-vocabulary`**, the **`add-to-table`** component of the **`NonIdentifyingStringOrIndex`** (see 7.14) shall always be set to **`TRUE`** and multiple identical character strings shall not be present in any string table; and

3) the final vocabulary of this fast infoset document becomes the external vocabulary; or

NOTE 3 – It is an implementation matter whether the final vocabulary is stored locally or whether only the XML document is stored and the final vocabulary generated by processing it.

c) as a set of vocabulary tables specified using any other sufficiently precise mechanism or text, which shall include the built-in table entries of clauses 9 and 10 (with the vocabulary table indexes specified by those clauses).

NOTE 4 – It is outside the scope of this Recommendation | International Standard to specify a notation for the definition of vocabulary tables.

NOTE 5 – The requirement to include the built-in table entries when this mechanism is used ensures that all vocabulary tables include the built-in table entries.

7.2.15 For an external vocabulary specified according to 7.2.14 c, all string and name table entries, except those in the PREFIX table and NAMESPACE NAME table, shall be assigned consecutive indexes starting from 1. The PREFIX table and NAMESPACE NAME table entries shall be assigned consecutive indexes starting from 2. All restricted alphabets other than those that are built-in shall be assigned consecutive indexes starting from 16. All encoding algorithms other than those that are built-in shall be assigned consecutive indexes starting from 32.

7.2.16 Each **`NonEmptyOctetString`**, **`EncodedCharacterString`**, and **`NameSurrogate`** (if any) that is present in any of the remaining components of the **`initial-vocabulary`** shall be added in order (see 8.1) to a vocabulary table, as specified in Table 1.

Table 1 – Mapping of component identifiers to vocabulary tables

Component identifier	ASN.1 type of the entry	Vocabulary table (see clause 8)
`restricted-alphabets`	`NonEmptyOctetString`	The restricted alphabet table (see 8.2)
`encoding-algorithms`	`NonEmptyOctetString`	The encoding algorithm table (see 8.3)
`prefixes`	`NonEmptyOctetString`	The PREFIX table (see 8.4)
`namespace-names`	`NonEmptyOctetString`	The NAMESPACE NAME table (see 8.4)
`local-names`	`NonEmptyOctetString`	The LOCAL NAME table (see 8.4)
`other-ncnames`	`NonEmptyOctetString`	The OTHER NCNAME table (see 8.4)
`other-uris`	`NonEmptyOctetString`	The OTHER URI table (see 8.4)
`attribute-values`	`EncodedCharacterString`	The ATTRIBUTE VALUE table (see 8.4)
`content-character-chunks`	`EncodedCharacterString`	The CONTENT CHARACTER CHUNK table (see 8.4)
`other-strings`	`EncodedCharacterString`	The OTHER STRING table (see 8.4)
`element-name-surrogates`	`NameSurrogate`	The ELEMENT NAME table (see 8.5)
`Attribute-name-surrogates`	`NameSurrogate`	The ATTRIBUTE NAME table (see 8.5)

7.2.17 A value of the **`NonEmptyOctetString`** type shall carry the UTF-8 encoding (see ISO/IEC 10646, Annex D) of a character string.

7.2.18 The restricted alphabet table and the encoding algorithm table in an initial vocabulary shall have at most 256 entries. All the other tables shall have at most 2^{20} entries.

NOTE – The restriction on the number of entries is to ensure common upper bounds for table indexes. The restriction also applies if table entries are added dynamically (see 7.13.7, 7.14.6, 7.14.7 and 7.16.7). These restrictions do not prevent the encoding of any XML infoset as a fast infoset document.

7.2.19 The built-in restricted alphabets have vocabulary table indexes between 1 and 2 (see clause 9). The vocabulary table indexes of the restricted alphabets in the **`restricted-alphabets`** component of **`initial-vocabulary`** (if present) shall be assigned as follows:

a) if there is no external vocabulary, or the external vocabulary contains only built-in restricted alphabets, then the indexes shall be assigned starting from 16;

b) otherwise, the indexes shall be assigned starting from one plus the highest restricted alphabet index in the external vocabulary.

NOTE – This means that vocabulary table indexes of 3 to 15 are not used. These values are reserved for future versions of this Recommendation | International Standard.

7.2.20 The built-in encoding algorithms have vocabulary table indexes between 1 and 10 (see clause 10). The vocabulary table indexes of the encoding algorithms in the **`encoding-algorithms`** component of **`initial-vocabulary`** (if present) shall be assigned as follows:

a) if there is no external vocabulary, or the external vocabulary contains only built-in encoding algorithms, then the indexes shall be assigned starting from 32;

b) otherwise, the indexes shall be assigned starting from one plus the highest encoding algorithm index in the external vocabulary.

NOTE – This means that vocabulary table indexes of 11 to 31 are not used. These values are reserved for future versions of this Recommendation | International Standard.

7.2.21 The PREFIX table shall have a built-in prefix entry of "xml", assigned an index of 1. The vocabulary table indexes of the prefixes in the **`prefixes`** component of **`initial-vocabulary`** (if present) shall be assigned as follows:

a) if there is no external vocabulary, or the external vocabulary contains only the built-in prefix entry, then the indexes shall be assigned starting from 2;

b) otherwise, the indexes shall be assigned starting from one plus the highest prefix index in the external vocabulary.

7.2.22 The NAMESPACE NAME table shall have a built-in namespace name entry of:

http://www.w3.org/XML/1998/namespace

This shall be assigned an index of 1.

7.2.23 The vocabulary table indexes of the namespace names in the **`namespace-names`** component of **`initial-vocabulary`** (if present) shall be assigned as follows:

a) if there is no external vocabulary, or the external vocabulary contains only the built-in namespace name entry, then the indexes shall be assigned starting from 2;

b) otherwise, the indexes shall be assigned starting from one plus the highest namespace name index in the external vocabulary.

7.2.24 The component **`notations`** represents the **[notations]** property of the **document** information item**.** The type of this component is a sequence-of type, even though the **[notations]** property is specified in W3C XML Information Set as an unordered set (of **notation** information items).

NOTE – Here and elsewhere, a sequence-of type is used rather than a set-of type because the latter does not satisfy the need for a strict ordering of all the components of a fast infoset document (see 8.1).

7.2.25 The component **`unparsed-entities`** represents the **[unparsed entities]** property of the **document** information item. The type of this component is a sequence-of type, even though the **[unparsed entities]** property is specified in W3C XML Information Set as an unordered set (of **unparsed entity** information items).

7.2.26 The component **`character-encoding-scheme`** represents the **[character encoding scheme]** property of the document information item. The type of this component is **`NonEmptyOctetString`** and a value of this component shall carry the UTF-8 encoding (see ISO/IEC 10646 Annex D) of the **[character encoding scheme]** property. The absence of this component in an abstract value of the **`Document`** type indicates that the **[character encoding scheme]** property has a value of "UTF-8".

NOTE – The support of the **[character encoding scheme]** property enables round-tripping of XML documents to and from fast infoset documents with no change of character encoding scheme. A creator of a fast infoset document from an XML document may encode the **[character encoding scheme]** property obtained from the encoding declaration of the XML document (see W3C XML 1.0, 4.3.1 and W3C XML 1.1, 4.3.1). A processor of a fast infoset document can use the component **`character-encoding-scheme`** (if present) if it wishes to produce the original encoding.

7.2.27 The component **`standalone`** represents the **[standalone]** property of the **document** information item. The abstract value **`TRUE`** represents the value **yes** of this property and the abstract value **`FALSE`** represents the value **no**. The absence of this component in an abstract value of the **`Document`** type indicates that the **[standalone]** property has no value.

7.2.28 The component **`version`** represents the **[version]** property of the **document** information item. The type of this component is **`NonIdentifyingStringOrIndex`** (see 7.14), representing here a character string of the OTHER STRING category. The absence of this component in an abstract value of the **`Document`** type indicates that the **[version]** property has no value.

7.2.29 The component **`children`** represents the **[children]** property of the **document** information item. Exactly one of the items of the sequence-of (at any position) shall use the alternative **`element`** of the choice type, and at most one of the items (at any position) shall use the alternative **`document-type-declaration`**. Each of the other items (if any) shall use either the alternative **`processing-instruction`** or the alternative **`comment`**.

7.2.30 The **[document element]** property of the **document** information item is not included in the **`Document`** type. The value of this property is always the one and only **element** information item that is a member of the **[children]** property of the **document** information item.

7.2.31 The **[base URI]** property of the **document** information item is not included in the **Document** type and is not supported in this Recommendation | International Standard.

7.2.32 The **[all declarations processed]** property of the **document** information item is not included in the **Document** type, and is assumed to have the value **true** (see 11.3).

7.3 The Element type

7.3.1 The **Element** type is:

```
Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (SIZE(1..MAX)) OF
            NamespaceAttribute OPTIONAL,
      qualified-name          QualifiedNameOrIndex
                                    -- ELEMENT NAME category --,
      attributes              SEQUENCE (SIZE(1..MAX)) OF
            Attribute OPTIONAL,
      children                SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                         Element,
                  processing-instruction          ProcessingInstruction,
                  unexpanded-entity-reference     UnexpandedEntityReference,
                  character-chunk                 CharacterChunk,
                  comment                         Comment }}
```

7.3.2 The **NameSpaceAttribute**, **QualifiedNameOrIndex**, **Attribute**, **ProcessingInstruction**, **UnexpandedEntityReference**, **CharacterChunk**, and **Comment** types are defined in 7.12, 7.16, 7.4, 7.5, 7.6, 7.7, and 7.8 respectively.

7.3.3 The **Element** type represents the **element** information item of the XML Information Set.

7.3.4 The component **namespace-attributes** represents the **[namespace attributes]** property of the **element** information item. The type of this component is a sequence-of type, even though the **[namespace attributes]** property is specified in W3C XML Information Set as an unordered set (of **attribute** information items).

NOTE – The type of the component of the sequence-of is **NamespaceAttribute** (rather than **Attribute**), even though the **[namespace attributes]** property of the **element** information item is specified in W3C XML Information Set as a set of **attribute** information items. In a restricted XML infoset (see 11.3), the properties of a **namespace** information item can be determined from the properties of an **attribute** information item representing a namespace attribute. The reverse is only partially true, but this limitation is considered acceptable for the expected uses of this Recommendation | International Standard. (See also the note in 7.2.24.)

7.3.5 The component **qualified-name** represents the qualified name (see 3.4.11) of the **element** information item (that is, the set consisting of the **[prefix]**, **[namespace name]**, and **[local name]** properties of this information item). The type of this component is **QualifiedNameOrIndex** (see 7.16), representing here a qualified name of the ELEMENT NAME category.

7.3.6 The component **attributes** represents the **[attributes]** property of the **element** information item. The type of this component is a sequence-of type, even though the **[attributes]** property is specified in W3C XML Information Set as an unordered set (of **attribute** information items).

7.3.7 The component **children** represents the **[children]** property of the **element** information item**.** When two or more adjacent children are **character** information items, a single **CharacterChunk** item may be used to represent those adjacent **character** information items.

NOTE – If there is a sequence of N adjacent characters among the children of an **element** information item, then any grouping of those N characters into a series of consecutive character chunks is allowed. However, it is expected that the creator of a fast infoset document will make each character chunk as large as possible in order to produce efficient encodings.

7.3.8 The **[in-scope namespaces]** property of the **element** information item is not included in the **Element** type.

NOTE – In a restricted XML infoset (see 11.3), the **[in-scope namespaces]** property of an **element** information item can be determined from the **[namespace attributes]** property of the **element** information item, together with the **[namespace attributes]** property of all the **element** information items (if any) that contain (directly or indirectly) that **element** information item.

7.3.9 The **[base URI]** property of the **element** information item is not included in the **Element** type and is not supported in this Recommendation | International Standard.

7.3.10 The **[parent]** property of the **element** information item is not included in the **Element** type. The value of this property, for any given **element** information item, is the **document** or **element** information item that contains that information item as a member of its **[children]** property.

7.4 The `Attribute` type

7.4.1 The **`Attribute`** type is:

```
Attribute ::= SEQUENCE {
      qualified-name       QualifiedNameOrIndex
                                 -- ATTRIBUTE NAME category --,
      normalized-value     NonIdentifyingStringOrIndex
                                 -- ATTRIBUTE VALUE category -- }
```

7.4.2 The **`QualifiedNameOrIndex`** and **`NonIdentifyingStringOrIndex`** types are defined in 7.16 and 7.14 respectively.

7.4.3 The **`Attribute`** type represents the **attribute** information item of the XML Information Set.

7.4.4 The component **`qualified-name`** represents the qualified name (see 3.4.11) of the **attribute** information item (that is, the set consisting of the **[prefix]**, **[namespace name]**, and **[local name]** properties of this information item). The type of this component is **`QualifiedNameOrIndex`** (see 7.16), representing here a qualified name of the ATTRIBUTE NAME category.

7.4.5 The component **`normalized-value`** represents the **[normalized value]** property of the **attribute** information item. The type of this component is **`NonIdentifyingStringOrIndex`** (see 7.14), representing here a character string of the ATTRIBUTE VALUE category.

7.4.6 The length of the character string assigned to **`normalized-value`** cannot be greater than 2^{32}.

NOTE – This restriction is implied by the ASN.1 definition, which is designed for optimization of the encodings and for simplicity of implementation (see also 11.3 j).

7.4.7 The **[specified]** property of the **attribute** information item is not included in the **`Attribute`** type.

7.4.8 The **[attribute type]** property of the **attribute** information item is not included in the **`Attribute`** type.

7.4.9 The **[references]** property of the **attribute** information item is not included in the **`Attribute`** type.

NOTE – In a restricted XML infoset (see 11.3), the **[references]** property of an **attribute** information item can be determined from the **[normalized value]** property of the **attribute** information item, together with the properties of other information items in the XML infoset.

7.4.10 The **[owner element]** property of the **attribute** information item is not included in the **`Attribute`** type. The value of this property, for any given **attribute** information item, is the **element** information item that contains that information item as a member of its **[attributes]** property.

7.5 The `ProcessingInstruction` type

7.5.1 The **`ProcessingInstruction`** type is:

```
ProcessingInstruction ::= SEQUENCE {
      target          IdentifyingStringOrIndex
                            -- OTHER NCNAME category --,
      content         NonIdentifyingStringOrIndex
                            -- OTHER STRING category -- }
```

7.5.2 The **`IdentifyingStringOrIndex`** and **`NonIdentifyingStringOrIndex`** types are defined in 7.13 and 7.14 respectively.

7.5.3 The **`ProcessingInstruction`** type represents the **processing instruction** information item of the XML Information Set.

7.5.4 The component **`target`** represents the **[target]** property of the **processing instruction** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER NCNAME category.

7.5.5 The component **`content`** represents the **[content]** property of the **processing instruction** information item. The type of this component is **`NonIdentifyingStringOrIndex`** (see 7.14), representing here a character string of the OTHER STRING category.

7.5.6 The length of the character string assigned to **`content`** cannot be greater than 2^{32}.

NOTE – This restriction is implied by the ASN.1 definition, which is designed for optimization of the encodings and for simplicity of implementation (see also 11.3 j).

7.5.7 The **[notation]** property of the **processing instruction** information item is not included in the **ProcessingInstruction** type.

NOTE – In a restricted XML infoset (see 11.3), the **[notation]** property of a **processing instruction** information item can be determined from the **[target]** property of the **processing instruction** information item together with the **[notations]** property of the **document** information item.

7.5.8 The **[parent]** property of the **processing instruction** information item is not included in the **ProcessingInstruction** type. The value of this property, for any given **processing instruction** information item, is the **document**, **element,** or **document type definition** information item that contains that information item as a member of its **[children]** property.

7.6 The UnexpandedEntityReference type

7.6.1 The **UnexpandedEntityReference** type is:

```
UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category -- }
```

7.6.2 The **IdentifyingStringOrIndex** type is defined in 7.13.

7.6.3 The **UnexpandedEntityReference** type represents the **unexpanded entity reference** information item of the XML Information Set.

7.6.4 The component **name** represents the **[name]** property of the **unexpanded entity reference** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER NCNAME category.

7.6.5 The component **system-identifier** represents the **[system identifier]** property of the **unexpanded entity reference** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **UnexpandedEntityReference** type indicates that the **[system identifier]** property has no value.

7.6.6 The component **public-identifier** represents the **[public identifier]** property of the **unexpanded entity reference** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **UnexpandedEntityReference** type indicates that the **[public identifier]** property has no value.

7.6.7 The **[declaration base URI]** property of the **unexpanded entity reference** information item is not included in the **UnexpandedEntityReference** type and is not supported in this Recommendation | International Standard.

7.6.8 The **[parent]** property of the **unexpanded entity reference** information item is not included in the **UnexpandedEntityReference** type. The value of this property, for any given **unexpanded entity reference** information item, is the **element** information item that contains that information item as a member of its **[children]** property.

7.7 The CharacterChunk type

7.7.1 The **CharacterChunk** type is:

```
CharacterChunk ::= SEQUENCE {
      character-codes           NonIdentifyingStringOrIndex
                                    -- CONTENT CHARACTER CHUNK category -- }
```

7.7.2 The **NonIdentifyingStringOrIndex** type is defined in 7.14.

7.7.3 The **CharacterChunk** type corresponds to the **character** information item, but represents a series of adjacent **character** information items (members of the **[children]** of the parent **element** information item) rather than a single **character** information item.

7.7.4 The number of **character** information items represented by a value of the **CharacterChunk** type shall not be zero.

7.7.5 The component **character-codes** represents the **[character code]** property of the (multiple) **character** information item(s) in the chunk. The type of this component is **NonIdentifyingStringOrIndex** (see 7.14), representing here a character string of the CONTENT CHARACTER CHUNK category.

7.7.6 The length of the character string assigned to **character-codes** cannot be greater than 2^{32}.

NOTE – This restriction is implied by the ASN.1 definition, which is designed for optimization of the encodings and for simplicity of implementation. This restriction does not prevent the encoding of an **element** information item containing more than 2^{32} **character** information items, because multiple chunks can be used.

7.7.7 The **[element content whitespace]** property of the **character** information item(s) is not included in the **CharacterChunk** type.

7.7.8 The **[parent]** property of the **character** information item(s) is not included in the **CharacterChunk** type. The value of this property, for any given **character** information item, is the **element** information item that contains that information item as a member of its **[children]** property.

7.8 The Comment type

7.8.1 The **Comment** type is:

```
Comment ::= SEQUENCE {
      content      NonIdentifyingStringOrIndex -- OTHER STRING category --}
```

7.8.2 The **NonIdentifyingStringOrIndex** type is defined in 7.14.

7.8.3 The **Comment** type represents the **comment** information item of the XML Information Set.

7.8.4 The component **content** represents the **[content]** property of the **comment** information item. The type of this component is **NonIdentifyingStringOrIndex** type (see 7.14), representing here a character string of the OTHER STRING category.

7.8.5 The length of the character string assigned to **content** cannot be greater than 2^{32}.

NOTE – This restriction is implied by the ASN.1 definition, which is designed for optimization of the encodings and for simplicity of implementation (see also 11.3 j).

7.8.6 The **[parent]** property of the **comment** information item is not included in the **Comment** type. The value of this property, for any given **comment** information item, is the **document** or **element** information item that contains that information item as a member of its **[children]** property.

7.9 The DocumentTypeDeclaration type

7.9.1 The **DocumentTypeDeclaration** type is:

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier    IdentifyingStringOrIndex OPTIONAL
                                 -- OTHER URI category --,
      public-identifier    IdentifyingStringOrIndex OPTIONAL
                                 -- OTHER URI category --,
      children             SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }
```

7.9.2 The **IdentifyingStringOrIndex** type is defined in 7.13.

7.9.3 The **DocumentTypeDeclaration** type represents the **document type declaration** information item of the XML Information Set.

7.9.4 The component **system-identifier** represents the **[system identifier]** property of the **document type declaration** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **DocumentTypeDeclaration** type indicates that the **[system identifier]** property has no value.

7.9.5 The component **public-identifier** represents the **[public identifier]** property of the **document type declaration** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **DocumentTypeDeclaration** type indicates that the **[public identifier]** property has no value.

7.9.6 The component **children** represents the **[children]** property of the **document type declaration** information item.

7.9.7 The **[parent]** property of the **document type declaration** information item is not included in the **DocumentTypeDeclaration** type. The value of this property, for any given **document type declaration** information item, is the **document** information item that contains that information item as a member of its **[children]** property.

7.10 The `UnparsedEntity` type

7.10.1 The **`UnparsedEntity`** type is:

```
UnparsedEntity ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex
                                -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category --,
      notation-name         IdentifyingStringOrIndex
                                -- OTHER NCNAME category -- }
```

7.10.2 The **`IdentifyingStringOrIndex`** type is defined in 7.13.

7.10.3 The **`UnparsedEntity`** type represents the **unparsed entity** information item of the XML Information Set.

7.10.4 The component **`name`** represents the **[name]** property of the **unparsed entity** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER NCNAME category.

7.10.5 The component **`system-identifier`** represents the **[system identifier]** property of the **unparsed entity** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER URI category.

7.10.6 The component **`public-identifier`** represents the **[public identifier]** property of the **unparsed entity** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **`UnparsedEntity`** type indicates that the **[public identifier]** property has no value.

7.10.7 The component **`notation-name`** represents the **[notation name]** property of the **unparsed entity** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER NCNAME category.

7.10.8 The **[declaration base URI]** property of the **unparsed entity** information item is not included in the **`UnparsedEntity`** type and is not supported in this Recommendation | International Standard.

7.10.9 The **[notation]** property of the **unparsed entity** information item is not included in the **`UnparsedEntity`** type.

NOTE – In a restricted XML infoset (see 11.3), the **[notation]** property of an **unparsed entity** information item can be determined from the **[notation name]** property of the **unparsed entity** information item together with the **[notations]** property of the **document** information item.

7.11 The `Notation` type

7.11.1 The **`Notation`** type is:

```
Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category -- }
```

7.11.2 The **`IdentifyingStringOrIndex`** type is defined in 7.13.

7.11.3 The **`Notation`** type represents the **notation** information item of the XML Information Set.

7.11.4 The component **`name`** represents the **[name]** property of the **notation** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER NCNAME category.

7.11.5 The component **`system-identifier`** represents the **[system identifier]** property of the **notation** information item. The type of this component is **`IdentifyingStringOrIndex`** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **`Notation`** type indicates that the **[system identifier]** property has no value.

7.11.6 The component **public-identifier** represents the **[public identifier]** property of the **notation** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the OTHER URI category. The absence of this component in an abstract value of the **Notation** type indicates that the **[public identifier]** property has no value.

7.11.7 The **[declaration base URI]** property of the **notation** information item is not included in the **Notation** type and not supported in this Recommendation | International Standard.

7.12 The NamespaceAttribute type

7.12.1 The **NamespaceAttribute** type is:

```
NamespaceAttribute ::= SEQUENCE {
      prefix              IdentifyingStringOrIndex OPTIONAL
                                -- PREFIX category --,
      namespace-name      IdentifyingStringOrIndex OPTIONAL
                                -- NAMESPACE NAME category -- }
```

7.12.2 The **IdentifyingStringOrIndex** type is defined in 7.13.

7.12.3 The **NamespaceAttribute** type represents an **attribute** information item that is a member of the **[namespace attributes]** property of an **element** information item of the XML Information Set.

NOTE – In the XML Information Set, both attributes and namespace attributes are **attribute** information items. In this Recommendation | International Standard, different types are used for optimization.

7.12.4 There are two types of namespace attributes in the XML Information Set:

a) default namespace declarations: the **[prefix]** property of the **attribute** information item has no value, and the **[local name]** property is **"xmlns"**;

b) non-default namespace declarations: the **[prefix]** property of the **attribute** information item is **"xmlns"**, and the **[local name]** property provides the prefix of the namespace declaration.

In both cases, the **[normalized value]** property of the **attribute** information item provides the namespace name of the namespace declaration.

7.12.5 If the namespace attribute is a default namespace declaration (case a of 7.12.4), then the component **prefix** shall be absent, otherwise (case b of 7.12.4) it shall be present, representing the **[local name]** property of the **attribute** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the PREFIX category.

7.12.6 If the **[normalized value]** property of the **attribute** information item is an empty string, then the component **namespace-name** shall be absent; otherwise, it shall be present, representing the **[normalized value]** property of the **attribute** information item. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the NAMESPACE NAME category.

7.12.7 The **[namespace name]** property of the **attribute** information item is always **"http://www.w3.org/2000/xmlns/"** (see W3C XML Infoset) and is not included in the **NamespaceAttribute** type.

7.13 The IdentifyingStringOrIndex type

7.13.1 The **IdentifyingStringOrIndex** type is:

```
IdentifyingStringOrIndex ::= CHOICE {
      literal-character-string   NonEmptyOctetString,
      string-index               INTEGER (1..one-meg) }
```

7.13.2 The **NonEmptyOctetString** type and the **one-meg** value are defined in 7.2.1.

7.13.3 The **IdentifyingStringOrIndex** type represents a character string that carries identification information.

NOTE – Examples of such character strings are prefixes, namespace names, and local names of elements and attributes.

7.13.4 An abstract value of this ASN.1 type holds either a character string (of a given category) as a value of the **NonEmptyOctetString** type, or the vocabulary table index of a character string (of a given category) in the vocabulary table for that category of string (see 8.4.2), called the applicable string table.

NOTE 1 – The category of string is always specified in the associated text of earlier subclauses (see 7.5 to 7.12) whenever this type is used.

NOTE 2 – "Identifying" character strings are treated differently from "non-identifying" character strings (see 7.14). While a non-identifying character string may be encoded in one of many encoding formats, all identifying character strings are encoded in UTF-8. Also, while a non-identifying character string may or may not (as a creator's option) be added to the dynamic string table (see 7.14.6), identifying character strings are always added to the dynamic string table (see 7.13.7).

7.13.5 The **`literal-character-string`**, if present, shall carry the UTF-8 encoding (see ISO/IEC 10646, Annex D) of the character string (see 7.13.4).

7.13.6 The **`string-index`**, if present, shall contain the vocabulary table index of any of the entries of the applicable string table that are identical to the character string.

7.13.7 When creating an abstract value of this ASN.1 type (representing a given character string of a given category), then if an identical character string exists in the current content of the applicable string table, the creator shall perform either action a) or action b) below as an implementation option (but the first option should be selected, if possible, as it produces the least number of indexes pointing to the same character string), otherwise (there is no existing identical character string) the creator shall perform action b) below. Actions a) and b) are:

a) select the **`string-index`** alternative, and assign to **`string-index`** the vocabulary table index of any of the existing entries that are identical to the character string;

b) select the **`literal-character-string`** alternative, assign the given character string to **`literal-character-string`**, and add to the applicable string table an identical character string unless that table already contains 2^{20} entries.

NOTE – The choice of performing action b) will result in more than one identical character string in the string table (if it does not already contain 2^{20} entries). This does not affect the later processing of character strings (see 7.13.8).

7.13.8 When processing an abstract value of this ASN.1 type representing a character string (of a given category), the processor shall determine the character string represented by the abstract value as follows:

a) If the **`string-index`** alternative is present, then the character string represented by the abstract value shall be the character string in the current content of the applicable string table whose vocabulary table index is the value of **`string-index`**.

b) If the **`literal-character-string`** alternative is present, then the character string represented by the abstract value shall be the value of **`literal-character-string`**, and an identical character string shall be added to the applicable string table (but see 7.13.9), unless that table already contains 2^{20} entries.

NOTE – The choice of performing action b) will result in more than one identical character string in the string table (if it does not already contain 2^{20} entries). This does not affect the later processing of character strings (see 7.13.8).

7.13.9 If a processor is unable (for any reason including implementation-specific limits) to add a string to a vocabulary table containing less than 2^{20} entries when such an addition is required by 7.13.8 b, it shall stop processing the fast infoset document and shall issue an error.

7.14 The `NonIdentifyingStringOrIndex` type

7.14.1 The **`NonIdentifyingStringOrIndex`** type is:

```
NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table                BOOLEAN,
            character-string            EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }
```

7.14.2 The **`EncodedCharacterString`** type and the **`one-meg`** value are defined in 7.17 and 7.2.1 respectively.

7.14.3 The **`NonIdentifyingStringOrIndex`** type represents a character string that does not carry identification information.

NOTE – An example of such a character string is the value of an attribute.

7.14.4 An abstract value of the **`NonIdentifyingStringOrIndex`** type holds either a character string (of a given category) as a value of the **`EncodedCharacterString`** type (see 7.17), or the vocabulary table index of a character string of a given category in the vocabulary table for that category of string (see 8.4.2), called the applicable string table.

NOTE – The category of string is always specified in the associated text of earlier clauses whenever this type is used.

7.14.5 The **`string-index`**, if present, shall either be zero (denoting a zero-length character string – see 7.14.6) or shall contain the vocabulary table index of any of the entries in the applicable string table that are identical to the character string.

7.14.6 For a zero-length character string, the creator shall always use the **string-index** alternative with the integer value zero. A processor shall treat such a value as representing a zero-length character string.

7.14.7 When creating an abstract value of the **NonIdentifyingStringOrIndex** type (representing a given character string of a given category) of non-zero length, then if an identical character string exists in the current content of the applicable string table, the creator shall perform either action a) or action b) below as an implementation option (but the first option should be selected, if possible, as it produces the least number of indexes pointing to the same character string), otherwise (there is no existing identical character string) the creator shall perform action b) below. Actions a) and b) are:

a) select the **string-index** alternative and assign to **string-index** the vocabulary table index of any of the existing entries that are identical to the character string;

b) select the **literal-character-string** alternative, assign the given character string to the **character-string** component, and either:

1) add an identical character string to the applicable string table and set the **add-to-table** component to **TRUE** (this action b1 shall not be used if the applicable string table already contains 2^{20} entries); or

NOTE 1 – If the applicable string table already contains 2^{20} entries, then only action a or b2 is available.

2) set the **add-to-table** component to **FALSE**.

NOTE 2 – The choice of performing action b1 will result in more than one identical character string in the current content. This does not affect the later processing of character strings (see 7.14.8).

7.14.8 When processing an abstract value of this ASN.1 type representing a character string of a given category, the processor shall determine the character string represented by the abstract value as follows:

a) If the **string-index** alternative is present, then the character string represented by the abstract value shall be the character string in the applicable string table whose vocabulary table index is the value of **string-index**.

NOTE 1 – If the **string-index** exceeds the current size of that vocabulary table, the fast infoset document is in error.

b) If the **literal-character-string** alternative is present and the **add-to-table** component has the value **TRUE**, then the character string represented by the abstract value shall be the value of the **character-string** component. The processor shall add to the applicable string table an identical character string (but see 7.14.12).

NOTE 2 – If the applicable string table already contains 2^{20} strings, the fast infoset document is in error.

c) If the **literal-character-string** alternative is present and the **add-to-table** component has the value **FALSE**, then the character string represented by the abstract value shall be the value of the **character-string** component.

7.14.9 If a processor is unable (for any reason including implementation-specific limits) to add a string to a vocabulary table when such an addition is required by 7.14.8 b), it shall stop processing the fast infoset document and shall issue an error.

7.15 The NameSurrogate type

7.15.1 The **NameSurrogate** type is:

```
NameSurrogate ::= SEQUENCE {
      prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
      namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
      local-name-string-index       INTEGER(1..one-meg) }
      (CONSTRAINED BY  {-- prefix-string-index shall only be present if
                       -- namespace-name-string-index is present --})
```

7.15.2 The **one-meg** value is defined in 7.2.1.

7.15.3 The **NameSurrogate** type holds three positive integers (the first two optional) forming a name surrogate (see 8.5.2).

NOTE – This type occurs only in the **initial-vocabulary** component of the **Document** type.

7.15.4 The **prefix-string-index** (if present), **namespace-name-string-index** (if present), and **local-name-string-index** shall be greater than zero and not greater than the number of entries in the PREFIX, NAMESPACE NAME, and LOCAL-NAME tables of the initial vocabulary, respectively.

NOTE – If the processor of a fast infoset document chooses not to check that this restriction is satisfied, security vulnerabilities could result from further processing.

7.15.5 The **prefix-string-index** shall be absent unless the **namespace-name-string-index** is present.

7.16 The QualifiedNameOrIndex type

7.16.1 The **QualifiedNameOrIndex** type is:

```
QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex OPTIONAL
                                        -- PREFIX category --,
            namespace-name        IdentifyingStringOrIndex OPTIONAL
                                        -- NAMESPACE NAME category --,
            local-name            IdentifyingStringOrIndex
                                        -- LOCAL NAME category -- },
      name-surrogate-index    INTEGER (1..one-meg) }
```

7.16.2 The **IdentifyingStringOrIndex** type and the **one-meg** value are defined in 7.13 and 7.2.1 respectively.

7.16.3 The **QualifiedNameOrIndex** type represents a qualified name (see 3.4.11).

7.16.4 An abstract value of this ASN.1 type holds either three components corresponding to the prefix, namespace name, and local name of a qualified name (of a given category), or the vocabulary table index of a name surrogate of a given category in the vocabulary table for that category of qualified name (see 8.5.2), called the applicable name table.

NOTE – The category is always specified in the associated text of earlier clauses whenever this type is used.

7.16.5 The **name-surrogate-index**, if present, shall contain the vocabulary table index of a name surrogate in the applicable name table.

7.16.6 If the **namespace-name** is absent, then the **prefix** shall also be absent.

7.16.7 When creating an abstract value of this ASN.1 type representing a given qualified name (of a given category), a creator shall perform the actions specified in the following subclauses.

7.16.7.1 The following conditions shall be evaluated in order:

a) either the qualified name has no prefix or the prefix of the qualified name exists in the current content of the PREFIX table;

b) either the qualified name has no namespace name or the namespace name of the qualified name exists in the current content of the NAMESPACE NAME table;

c) the local name of the qualified name exists in the current content of the LOCAL NAME table;

d) the first three conditions are all satisfied and a name surrogate (see 8.5), consisting of the vocabulary table index(es) of the prefix (if any), namespace name (if any), and local name, exists in the current content of the applicable name table.

7.16.7.2 If all the conditions above are satisfied, then the **name-surrogate-index** alternative shall be selected, and shall be set to the vocabulary table index of the name surrogate determined in 7.16.7.1 d in the applicable name table, completing the procedures of 7.16.7.

7.16.7.3 Otherwise, the **literal-qualified-name** alternative shall be selected, and its components shall be assigned as follows:

a) if the qualified name has no prefix, then the **prefix** component shall be absent, otherwise the prefix shall be assigned to the **prefix** component by applying 7.13.7 with the restriction that action 7.13.7 b) shall not be performed if an identical character string exists in the current content of the applicable string table. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the PREFIX category;

b) if the qualified name has no namespace name, then the **namespace-name** component shall be absent, otherwise the namespace name shall be assigned to the **namespace-name** component by applying 7.13.7 with the restriction that action 7.13.7 b) shall not be performed if an identical character string exists in the current content of the applicable string table. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the NAMESPACE NAME category (see 8.4.2);

c) the local name of the qualified name shall be assigned (by applying 7.13.7) to the **local-name** component. The type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the LOCAL NAME category.

NOTE – The application of 7.13.7 in this subclause may cause the addition of the local name to the LOCAL NAME table.

7.16.7.4 If the application of 7.13.7 in subclause 7.16.7.3 to one or more of the three above character strings has not added the string to a vocabulary table because that table already contained 2^{20} entries (see 7.13.7 b), then a name surrogate for this qualified name cannot be created.

7.16.7.5 Otherwise, a name surrogate (see 8.5), consisting of the vocabulary table index(es) of the prefix (if any), namespace name (if any), and local name, shall be created. If this name surrogate does not exist in the current content of the applicable name table, then it shall be added to that table, unless the table already contains 2^{20} entries.

7.16.8 When processing an abstract value of the **QualifiedNameOrIndex** type representing a qualified name of a given category, the processor shall determine the qualified name represented by the abstract value according to the following subclauses.

7.16.8.1 If the **name-surrogate-index** alternative is present, then the qualified name represented by the abstract value shall be the one represented by the name surrogate of the given category (see 8.5.2) in the applicable name table whose vocabulary table index is the value of **name-surrogate-index**.

7.16.8.2 If the **literal-qualified-name** alternative is present, then:

a) The qualified name represented by the abstract value shall be determined as follows:

1) the prefix of the qualified name shall be determined (by applying 7.13.8) from the **prefix** component (if this is present); the type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the PREFIX category;

2) the namespace name of the qualified name shall be determined (by applying 7.13.8) from the **namespace-name** component (if this is present); the type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the NAMESPACE NAME category (see 8.4.2);

3) the local name of the qualified name shall be determined (by applying 7.13.8) from the **local-name** component; the type of this component is **IdentifyingStringOrIndex** (see 7.13), representing here a character string of the LOCAL NAME category.

NOTE – These actions may require an addition to the corresponding vocabulary tables, as specified in 7.13.8 b.

b) If, after possible additions as a result of processing the components of the **literal-qualified-name**, vocabulary table indexes are available for all of the components that are present, then a name surrogate consisting of those vocabulary indexes shall be added to the applicable name table (but see 7.16.9), unless that vocabulary table already contains 2^{20} name surrogates.

7.16.9 If a processor is unable (for any reason including implementation-specific limits) to add a name surrogate to a vocabulary table containing less than 2^{20} entries when such an addition is required by 7.16.8.2 b, it shall stop processing the fast infoset document and shall issue an error.

7.17 The EncodedCharacterString type

7.17.1 The **EncodedCharacterString** type is:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                   NULL,
            utf-16                  NULL,
            restricted-alphabet     INTEGER(1..256),
            encoding-algorithm      INTEGER(1..256) },
      octets                NonEmptyOctetString }
```

7.17.2 The **EncodedCharacterString** type contains an encoding of a character string. It specifies an octet string that is a reversible mapping of a character string into an octet string. A creator of a fast infoset document specifies in the **encoding-format** the encoding that has been used, and the processor uses this information to decode the octets in the **octets** component into a character string.

7.17.3 The component **octets** shall carry an octet string value which is an encoding of the character string specified by the **encoding-format** component.

NOTE – In general, there are multiple encodings that can be used for a given character string. A creator of a fast infoset document may choose an encoding based on certain criteria (for example, it can try to optimize size or processing speed), but it may also prefer the convenience and universal applicability of UTF-8 or UTF-16BE. It is also the case that some encodings will only be able to encode character strings that contain only a subset of the ISO/IEC 10646 characters.

7.17.4 The **`utf-8`** encoding format can be used for any character string. This encoding format shall be applied by producing a UTF-8 encoding (see ISO/IEC 10646) of the character string and assigning that encoding to the **`octets`** component.

NOTE – This encoding format is most suitable for character strings in which ISO/IEC 10646 characters in the early part of the ISO/IEC 10646 Basic Multilingual Plane are the most common, and where none of the other encoding formats is applicable or more useful.

7.17.5 The **`utf-16`** encoding format can also be used for any character string. This encoding format shall be applied by producing a UTF-16BE encoding (see Unicode, 2.6) of the character string and assigning that encoding to the **`octets`** component.

NOTE 1 – This encoding format is most suitable for character strings in which a wide range of ISO/IEC 10646 characters are present, and none of the other encoding formats is applicable or more useful.

NOTE 2 – The byte order of the UTF-16BE encoding specified in Unicode, 2.6 is most significant byte first (the earlier byte in the octet string).

7.17.6 The **`restricted-alphabet`** encoding format is based on the use of a restricted alphabet, selected from those present in the restricted alphabet table. The **`restricted-alphabet`** component shall contain the vocabulary table index of the restricted alphabet. This encoding format can only be used for a character string that consists entirely of the characters in the restricted alphabet in the restricted alphabet table entry indexed by the **`restricted-alphabet`** component. A **`restricted-alphabet`** encoding format shall be applied as specified in 7.17.6.1 to 7.17.6.6.

7.17.6.1 An integer value (starting from zero) shall be assigned to each character of the restricted alphabet, in order.

7.17.6.2 Each character in the character string shall be converted to an integer, which is the integer assigned to the character in the restricted alphabet.

7.17.6.3 Each integer shall be represented as an unsigned integer binary number in a bit field. The size of the bit-field shall be determined by the integer value assigned to the last character of the restricted alphabet. That integer value shall be incremented by 1 to produce an integer value (N, say). The bit-field size shall be the minimum number of bits to encode N as an unsigned integer encoding.

NOTE 1 – The incrementing is necessary because a value of all ones in the bit-field is interpreted as the end of the character string, and cannot be used to represent a character. This means that if the restricted alphabet contains a number of characters that is an exact power of two, the bit-field will have one more bit than might otherwise be expected.

NOTE 2 – For example, if there are 24 characters in the restricted alphabet, then each character will encode into five bits, but if there are 32 characters in the restricted alphabet, then each character will encode into six bits.

7.17.6.4 All these bit fields shall be concatenated (in order) into a bit string.

7.17.6.5 If the length of the resulting bit string is not an integral multiple of 8 bits, then '1' bits shall be appended to make the length of the bit-string an integral multiple of 8 bits.

7.17.6.6 The resulting bit string (which is now an integral multiple of eight bits), reinterpreted as an octet string, shall be assigned to the **`octets`** component.

7.17.7 The **`encoding-algorithm`** encoding format is specified by the encoding algorithm (see 8.3) that is the table entry in the encoding algorithm table whose vocabulary table index is the value of the **`encoding-algorithm`** component. The encoding algorithm vocabulary table index shall be assigned to the **`encoding-algorithm`** component, and the octet string resulting from the encoding shall be placed in the **`octets`** component.

8 Construction and processing of a fast infoset document

A fast infoset document makes heavy use of vocabulary table indexes into a variety of tables that are built at various stages in the construction and in the processing of that document. Subclause 8.1 specifies a conceptual ordering of the components of an abstract value of the **`Document`** type to ensure that the creator and the processors of fast infoset documents create identical vocabulary tables. Subsequent subclauses specify the vocabulary tables that are built and used in the creation and processing of a fast infoset document. The representation of these tables in a computer system is an implementation matter, and is not standardized. A vocabulary table provides a mapping from a vocabulary table index to information in an XML infoset (possibly indirectly).

NOTE – The vocabulary tables for a fast infoset document are built dynamically during the construction of the fast infoset document. They are built dynamically from the content of the fast infoset document during the processing of that fast infoset document. They are never exchanged in any other form.

8.1 Conceptual ordering of components of an abstract value of the `Document` type

8.1.1 In order to ensure that different implementations assign vocabulary table indexes in the same way when constructing and when processing a fast infoset document, a conceptual order is specified for the components of an abstract value of the **`Document`** type. The construction and the processing of such abstract values shall use this conceptual order when adding strings (see 7.13.7 and 7.14.6) and name surrogates (see 7.16.7) to the vocabulary tables.

NOTE – This order is the same as the order of the encodings of the components in a fast infoset document. It does not necessarily imply that the semantics carried by the document are processed in this order. The order is defined solely for the purposes of ensuring that the same vocabulary table index is assigned for any given vocabulary table entry by both the creator and the processor of a fast infoset document.

8.1.2 The conceptual order for the construction and processing of a fast infoset document is specified as follows: The components of an abstract value of the **`Document`** type shall be visited according to the algorithm specified in 8.1.2.1 to 8.1.2.5. The order in which the components are visited defines the conceptual order.

8.1.2.1 The top-level component of the abstract value (corresponding to the **`Document`** type) shall be visited first.

8.1.2.2 If the type of the component being visited is a sequence type, then the components of the sequence type that are present in the abstract value shall be visited in the order of their textual definition, from the first component that is present to the last component that is present, applying subclauses 8.1.2.1 to 8.1.2.5 recursively to each component being visited.

8.1.2.3 If the type of the component being visited is a sequence-of type, then the occurrences of the component of the sequence-of shall be visited in sequence-of order, from the first occurrence to the last occurrence of the component of the sequence-of, applying subclauses 8.1.2.1 to 8.1.2.5 recursively to each component being visited.

8.1.2.4 If the type of the component being visited is a choice type, then the alternative that is present in the abstract value shall be visited and subclauses 8.1.2.1 to 8.1.2.5 shall be applied recursively to that alternative.

8.1.2.5 If the type of the component being visited is any other ASN.1 type, then no further action is required for that component.

8.2 The restricted alphabet table

8.2.1 Each fast infoset document has a restricted alphabet table associated with it. The restricted alphabet table contains restricted alphabets that can be referenced through a vocabulary table index.

8.2.2 Each entry in the restricted alphabet table shall be an ordered set of distinct ISO/IEC 10646 characters of any size between 2 and 2^{20} characters.

NOTE – A restricted alphabet permits a compact encoding of any character string consisting entirely of characters from that set, through the assignment of progressive integers to the characters in the set and the use of those integers to encode the characters of the string (see 7.17.6).

8.3 The encoding algorithm table

8.3.1 Each fast infoset document has an encoding algorithm table associated with it. The encoding algorithm table contains definitions of encoding algorithms that can be referenced through a vocabulary table index.

8.3.2 Each entry in this table specifies the encoding of a character string with some defined characteristics into an octet string (see 7.17.7).

NOTE – The defined characteristics may refer to the length of the string, to the characters appearing in it, or to an arbitrarily complex pattern of characters. In general, a given encoding algorithm applies only to a special and defined subset of the ISO/IEC 10646 character strings.

8.3.3 Encoding algorithms are subject to the following restrictions:

a) the encoding algorithm shall have a URI associated with it, unless it is a built-in encoding algorithm, so that it can be referenced for addition to the table;

b) the encoding algorithm shall specify precisely what kinds of character strings it can be applied to; that specification shall include a restricted alphabet (if any), a length range (if any), and any additional constraints upon the length and the content of the character strings (such as a pattern);

c) for any character string to which it can be applied, the encoding algorithm shall provide a reversible mapping from that character string to an octet string.

NOTE 1 – The above implies that there cannot be any character string S for which an encoding step from S to E, followed by a decoding step from E to S', results in S' ≠ S, even if the difference between S and S' is small (for example, an extra SPACE). On the other hand, it is not required that any character string S be encodable, nor is it required that the encodings be canonical.

NOTE 2 – An application that creates a fast infoset document from in-memory data, such as floating point numbers, is not required to produce a lexical representation of that data and then to apply an encoding algorithm to that representation. The application can instead create an octet string directly from that data, provided that the octet string is one that could have been produced by applying that encoding algorithm to a character string that represents the in-memory data and is a character string to which that encoding algorithm can be applied.

NOTE 3 – Encoding algorithms (other than the built-in ones) may be specified in other standards or be the subject of mutual agreement between the creator and the processors of a fast infoset document.

8.4 The dynamic string tables

8.4.1 Each fast infoset document has eight dynamic string tables associated with it. Each dynamic string table contains character strings that can be referenced through a vocabulary table index.

8.4.2 This Recommendation | International Standard classifies all the character strings that can occur in a fast infoset document into the eight following categories, each of which has a dynamic string table:

a) PREFIX: This category comprises character strings that are the **[prefix]** property of an **element**, **attribute**, or **namespace** information item.

b) NAMESPACE NAME: This category comprises character strings that are the **[namespace name]** property of an **element**, **attribute**, or **namespace** information item.

c) LOCAL NAME: This category comprises character strings that are the **[local name]** property of an **element** or **attribute** information item.

d) OTHER NCNAME: This category comprises character strings that are the **[target]** property of a **processing instruction** information item; or the **[name]** property of an **unexpanded entity reference**, **unparsed entity**, or **notation** information item; or the **[notation name]** property of an **unparsed entity** information item.

e) OTHER URI: This category comprises character strings that are the **[system identifier]** property or the **[public identifier]** property of an **unexpanded entity reference**, **document type declaration**, **unparsed entity**, or **notation** information item.

f) ATTRIBUTE VALUE: This category comprises character strings that are the **[normalized value]** property of an **attribute** information item.

g) CONTENT CHARACTER CHUNK: This category comprises character strings that are the **[character code]** property of a chunk of **character** information items that are consecutive children of a given **element** information item.

h) OTHER STRING: This category comprises character strings that are the **[version]** property of a **document** information item; or the **[content]** property of a **processing instruction** or **comment** information item.

8.5 The dynamic name tables and name surrogates

8.5.1 Each fast infoset document has two dynamic name tables associated with it. Each dynamic name table contains name surrogates that can be referenced through a vocabulary table index and are used to identify a qualified name, which may be either prefixed or unprefixed (and may or may not have a namespace name).

8.5.2 A name surrogate is a set of up to three ordered vocabulary table indexes:

a) (optionally) the index of a string in the PREFIX table;

b) (optionally) the index of a string in the NAMESPACE NAME table; and

c) the index of a string in the LOCAL NAME table.

The first vocabulary table index shall not be present unless the second is present.

8.5.3 Three cases can occur:

a) all three indexes are present, in which case the name surrogate represents a prefixed qualified name;

b) only the second and third indexes are present, in which case the name surrogate represents an unprefixed qualified name that has a namespace name;

c) only the third index is present, in which case name surrogate represents an unprefixed qualified name that has no namespace name.

8.5.4 This Recommendation | International Standard classifies all the qualified names that can occur in a fast infoset document (and therefore also the name surrogates that represent them) into the two following categories, each of which has a dynamic name table:

a) ELEMENT NAME: This category comprises name surrogates representing the qualified name of an **element** information item.

b) ATTRIBUTE NAME: This category comprises name surrogates representing the qualified name of an **attribute** information item.

8.5.5 The qualified name that is represented by a given name surrogate shall be determined as follows, given a dynamic string table:

a) the first vocabulary table index (if present) shall be interpreted as the vocabulary table index of a character string in the PREFIX table; and

b) the second vocabulary table index of the name surrogate (if present) shall be interpreted as the vocabulary table index of a character string in the NAMESPACE NAME table; and

c) the third integer shall be interpreted as the vocabulary table index of a character string in the LOCAL NAME table.

9 Built-in restricted alphabets

9.1 The "numeric" restricted alphabet

9.1.1 This restricted alphabet has a vocabulary table index of 1, and consists of the following fifteen ISO/IEC 10646 characters (in this order):

DIGIT ZERO to DIGIT NINE

HYPHEN-MINUS

PLUS SIGN

FULL STOP

LATIN SMALL LETTER E

SPACE

9.1.2 This restricted alphabet is suitable for encoding character strings representing several types of numbers, including floating-point numbers in scientific notation. A single character string may contain multiple numbers separated by spaces.

9.2 The "date and time" restricted alphabet

9.2.1 This restricted alphabet has a vocabulary table index of 2 and consists of the following fifteen ISO/IEC 10646 characters (in this order):

DIGIT ZERO to DIGIT NINE

HYPHEN-MINUS

COLON

LATIN CAPITAL LETTER T

LATIN CAPITAL LETTER Z

SPACE

9.2.2 This restricted alphabet is suitable for encoding character strings representing the most common expressions of date and time based on ISO 8601. A single character string may contain multiple such expressions separated by spaces.

10 Built-in encoding algorithms

10.1 General

10.1.1 This clause specifies the built-in encoding algorithms. Built-in encoding algorithms have no URI associated with them.

NOTE – URIs are necessary for encoding algorithms that are to be explicitly identified in an initial vocabulary, but built-in encoding algorithms are always added implicitly to the encoding algorithm table and therefore do not need URIs.

10.1.2 In this clause, the term "word" indicates any group of consecutive characters within a given character string, which:

a) contains no SPACE; and

b) is either at the beginning of the character string or is preceded by a SPACE; and

c) is either at the end of the character string or is followed by a SPACE.

NOTE – A "word" is not restricted to alphabetic characters.

10.2 The "hexadecimal" encoding algorithm

10.2.1 This encoding algorithm has a vocabulary table index of 1, and can only be applied to a character string that consists of the following sixteen ISO/IEC 10646 characters:

DIGIT ZERO to DIGIT NINE

LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER F

and contains an even number of characters (including zero).

NOTE – No embedded XML whitespace is allowed.

10.2.2 The character string shall be interpreted as the hexadecimal encoding of an octet string, with the first character of the string corresponding to the most significant nibble of the first octet, and so on.

10.3 The "base64" encoding algorithm

10.3.1 This encoding algorithm has a vocabulary table index of 2, and can only be applied to a character string that:

a) consists entirely of the characters LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z, LATIN SMALL LETTER A to LATIN SMALL LETTER Z, DIGIT ZERO to DIGIT NINE, PLUS SIGN, SOLIDUS, and EQUALS SIGN; and

NOTE – This does not allow the presence of XML whitespace in the character string.

b) either is a valid instance of the Content Transfer Encoding specified in IETF RFC 2045, 6.8, or would become a valid instance of that encoding through the insertion of XML whitespace wherever required by IETF RFC 2045.

10.3.2 The character string shall be interpreted as the Base64 encoding (see IETF RFC 2045) of an octet string (assuming that additional XML whitespace is present where required by IETF RFC 2045). The resulting octet string is the octet string specified by that Base64 encoding.

10.3.3 This encoding algorithm is suitable for encoding character strings that contain no XML whitespace, and are Base64 strings (of any length) or would become Base64 strings by the addition of XML whitespace.

10.4 The "short" encoding algorithm

10.4.1 This encoding algorithm has a vocabulary table index of 3, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string consists entirely of the characters DIGIT ZERO to DIGIT NINE, HYPHEN-MINUS, and SPACE; and

b) neither the first nor the last character in the character string is a SPACE, and there is no pair of adjacent SPACEs; and

c) the character string contains at least one word (see 10.1.2); and

d) each HYPHEN-MINUS that is present is the first character of a word; and

e) each HYPHEN-MINUS is followed by at least one character in the range DIGIT ONE to DIGIT NINE; and

f) each DIGIT ZERO is either the only character in a word or is preceded by a character in the range DIGIT ONE to DIGIT NINE; and

g) each word in the character string, if interpreted as a signed integer base-10 numeric character string, yields a value in the range –32768 to 32767.

10.4.2 Each word (see 10.1.2) in the character string shall be interpreted as a signed integer base-10 numeric character string and shall be represented as a 16-bit two's complement integer.

10.4.3 Each group of 8 bits in the 16-bit two's complement integer for a word shall produce an octet of the octet string, beginning with the highest-order group of 8 bits. The highest-order bit in each group of 8 bits shall become the most significant bit of the corresponding octet. If there are multiple words in the character string, they shall be encoded in order, and the octets produced by the multiple 16-bit two's complement integers shall be concatenated in that order.

10.4.4 This encoding algorithm is suitable for encoding character strings representing a single integer in the range –32768 to 32767 (representable as a 16-bit two's complement integer) or a list of such integers.

10.5 The "int" encoding algorithm

10.5.1 This encoding algorithm has a vocabulary table index of 4, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string satisfies the conditions specified under 10.4.1 a to f; and

b) each word (see 10.1.2) in the character string, if interpreted as a signed integer base-10 numeric character string, yields a value in the range –2147483648 to 2147483647.

10.5.2 Each word (see 10.1.2) in the character string shall be interpreted as a signed integer base-10 numeric character string and shall be represented as a 32-bit two's complement integer.

10.5.3 Each group of 8 bits in the 32-bit two's complement integer for a word shall produce an octet of the octet string, beginning with the highest-order group of 8 bits. The highest-order bit in each group of 8 bits shall become the most significant bit of the corresponding octet. If there are multiple words in the character string, they shall be encoded in order, and the octets produced by the multiple 32-bit two's complement integers shall be concatenated in that order.

10.5.4 This encoding algorithm is suitable for encoding character strings representing a single integer in the range –2147483648 to 2147483647 (representable as a 32-bit two's complement integer) or a list of such integers.

10.6 The "long" encoding algorithm

10.6.1 This encoding algorithm has a vocabulary table index of 5, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string satisfies the conditions specified under 10.4.1 a to f; and

b) each word (see 10.1.2) in the character string, if interpreted as a signed integer base-10 numeric character string, yields a value in the range –9223372036854775808 to 9223372036854775807.

10.6.2 Each word (see 10.1.2) in the character string shall be interpreted as a signed integer base-10 numeric character string and shall be represented as a 64-bit two's complement integer.

10.6.3 Each group of 8 bits in the 64-bit two's complement integer for a word shall produce an octet of the octet string, beginning with the highest-order group of 8 bits. The highest-order bit in each group of 8 bits shall become the most significant bit of the corresponding octet. If there are multiple words in the character string, they shall be encoded in order, and the octets produced by the multiple 64-bit two's complement integers shall be concatenated in that order.

10.6.4 This encoding algorithm is suitable for encoding character strings representing a single integer in the range –9223372036854775808 to 9223372036854775807 (representable as a 64-bit two's complement integer) or a list of such integers.

10.7 The "boolean" encoding algorithm

10.7.1 This encoding algorithm has a vocabulary table index of 6, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string consists entirely of one or more of the word "false" or the word "true", and the character SPACE; and

b) neither the first nor the last character in the character string is a SPACE, and there is no pair of adjacent SPACEs; and

c) the character string contains at least one word (see 10.1.2).

10.7.2 Each word "false" or "true" in the character string shall be encoded as a single bit (set to zero or one, respectively) of the octet string being produced, starting from the fifth bit of the first octet proceeding to the eighth bit of the first octet. Subsequent bits are placed in subsequent octets proceeding from the first bit of each octet to the eighth bit of that octet, using only as many octets as required. Any unused bits in the last octet shall be set to zero.

10.7.3 The first four bits of the first octet shall contain the number of unused bits in the last octet, encoded as a 4-bit unsigned integer.

NOTE – The first octet may also be the last octet and may contain up to three unused bits. If there is more than one octet, then the last octet may contain up to seven unused bits.

10.8 The "float" encoding algorithm

10.8.1 This encoding algorithm has a vocabulary table index of 7, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string consists entirely of the characters DIGIT ZERO to DIGIT NINE, HYPHEN-MINUS, FULL STOP, LATIN CAPITAL LETTER E, and SPACE; and

NOTE – LATIN SMALL LETTER E is not allowed, as the encoding would then not be reversible.

b) neither the first nor the last character in the character string is a SPACE, and there is no pair of adjacent SPACEs; and

c) the character string contains at least one word (see 10.1.2); and

d) each word in the character string matches the canonical lexical representation of a float as specified in W3C XML Schema, Part 2, 3.2.4; and

e) each word in the character string, if interpreted as a floating-point base-10 numeric character string, yields a value that is representable as a 32-bit IEEE 754 float.

10.8.2 Each word (see 10.1.2) in the character string shall be interpreted as a floating-point base-10 numeric character string and shall be represented as a 32-bit IEEE 754 float.

10.8.3 Each group of 8 bits in the 32-bit IEEE 754 float for a word shall produce an octet of the octet string, beginning with the leading group of 8 bits. The leading bit in each group of 8 bits shall become the most significant bit of the corresponding octet. If there are multiple words in the character string, they shall be encoded in order, and the octets produced by the multiple 32-bit IEEE 754 floats shall be concatenated in that order.

10.8.4 This encoding algorithm is suitable for encoding character strings representing a single floating-point number representable as a 32-bit IEEE 754 float or a list of such floating-point numbers.

10.9 The "double" encoding algorithm

10.9.1 This encoding algorithm has a vocabulary table index of 8, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string satisfies the conditions specified under 10.8.1 a to c;

b) each word in the character string matches the canonical lexical representation of a double as specified in W3C XML Schema, Part 2, 3.2.5; and

c) each word (see 10.1.2) in the character string, if interpreted as a floating-point base-10 numeric character string, yields a value that is representable as a 64-bit IEEE 754 double.

10.9.2 Each word (see 10.1.2) in the character string shall be interpreted as a floating-point base-10 numeric character string and shall be represented as a 64-bit IEEE 754 double.

10.9.3 Each group of 8 bits in the 64-bit IEEE 754 float for a word shall produce an octet of the octet string, beginning with the leading group of 8 bits. The leading bit in each group of 8 bits shall become the most significant bit of the corresponding octet. If there are multiple words in the character string, they shall be encoded in order, and the octets produced by the multiple 64-bit IEEE 754 floats shall be concatenated.

10.9.4 This encoding algorithm is suitable for encoding character strings representing a single floating-point number representable as a 64-bit IEEE 754 double or a list of such floating-point numbers.

10.10 The "uuid" encoding algorithm

10.10.1 This encoding algorithm has a vocabulary table index of 9, and can only be applied to a character string that satisfies all of the following conditions:

a) the character string consists entirely of the characters DIGIT ZERO to DIGIT NINE, LATIN SMALL LETTER A to LATIN SMALL LETTER F, HYPHEN-MINUS, and SPACE; and

b) neither the first nor the last character in the character string is a SPACE, and there is no pair of adjacent SPACEs; and

c) the character string contains at least one word (see 10.1.2);

d) each word contains exactly 36 characters; and

e) in each word, there are exactly four HYPHEN-MINUS characters, occupying the positions 9, 14, 19, and 24 (counting from one).

10.10.2 Each word (see 10.1.2) in the character string shall be interpreted as the hexadecimal representation of a UUID (see ITU-T Rec. X.667 | ISO/IEC 9834-8, 6.4), and shall be represented as a 16-octet unsigned integer as specified in ITU-T Rec. X.667 | ISO/IEC 9834-8, 6.3. If there are multiple words, then the multiple 16-octet unsigned integers shall be concatenated.

10.10.3 This encoding algorithm is suitable for encoding character strings representing a single UUID or a list of UUIDs.

10.11 The "cdata" encoding algorithm

10.11.1 This encoding algorithm has a vocabulary table index of 10, and can be applied to any character string.

10.11.2 The octet string produced shall be the UTF-8 encoding (see ISO/IEC 10646) of the character string.

10.11.3 This algorithm shall be used only with XML infosets created by parsing an XML document and where additional information identifies that the character string corresponds to an entire CDATA section (see W3C XML 1.0 and W3C XML 1.1). If this encoding algorithm is used within a fast infoset document, then all character strings that correspond to CDATA sections shall have this encoding algorithm applied.

11 Restrictions on the supported XML infosets and other simplifications

11.1 This Recommendation | International Standard supports most XML infosets that are likely to be encountered in practice, but does not support some XML infosets that are in theory possible, but do not normally arise.

11.2 The term "XML-self-consistent" is used in this clause with the following meaning: a set properties of one or more information items is "XML-self-consistent" if that set of properties could have been obtained by parsing a suitable namespace-well-formed XML document.

11.3 XML infosets that are supported meet all of the following conditions:

a) the **[all declarations processed]** property of the **document** information item has the value **true**;

b) the **[in-scope namespaces]** property of each **element** information item together with the **[namespace attributes]** property of all **element** information items form an XML-self-consistent set;

c) the **[namespace name]** property of each **element** information item together with the **[namespace attributes]** property of all **element** information items and the **[prefix]** property of that **element** information item form an XML-self-consistent set;

d) the **[namespace name]** property of each **attribute** information item together with the **[namespace attributes]** property of all **element** information items and the **[prefix]** property of that **attribute** information item form an XML-self-consistent set;

e) the **[references]** property of each **attribute** information item together with the **[normalized value]** property of the **attribute** information item form an XML-self-consistent set;

f) the **[notation]** property of each **processing instruction** information item together with the **[target]** property of the **processing instruction** information item together with the **[notations]** property of the **document** information item form an XML-self-consistent set;

g) the **[notation]** property of each **unparsed entity** information item together with the **[notation name]** property of the **unparsed entity** information item and the **[notations]** property of the **document** information item form an XML-self-consistent set;

h) the **[element content whitespace]** property of all **character** information items that do not represent whitespace has the value **false**;

i) the **[element content whitespace]** property of each **character** information item together with the **[character code]** property of the **character** information item form an XML-self-consistent set;

j) the **[normalized value]** property of all **attribute** information and the **[content]** property of all **comment** and **processing** instruction information items contain at most 2^{32} characters.

11.4 The following properties of information items of the XML Information Set are not included in the ASN.1 types representing those information items:

a) the **[document element]**, **[base URI]**, and **[all declarations processed]** properties of the **document** information item (see 7.2.30, 7.2.31, and 7.2.32);

b) the **[in-scope namespaces]**, **[base URI]**, and **[parent]** properties of the **element** information item (see 7.3.8, 7.3.9, and 7.3.10);

c) the **[specified]**, **[attribute type]**, **[references]**, and **[owner element]** properties of the **attribute** information item (see 7.4.7, 7.4.8, 7.4.9, and 7.4.10);

d) the **[notation]** and **[parent]** properties of the **processing instruction** information item (see 7.5.7 and 7.5.8);

e) the **[declaration base URI]** and **[parent]** properties of the **unexpanded entity reference** information item (see 7.6.7 and 7.6.8);

f) the **[element content whitespace]** property of the **character** information item (see 7.7.7);

g) the **[parent]** property of the **character** information item (see 7.7.8);

h) the **[parent]** property of the **comment** information item (see 7.8.6);

i) the **[parent]** property of the **document type declaration** information item (see 7.9.7);

j) the **[declaration base URI]** and **[notation]** properties of the **unparsed entity** information item (see 7.10.8 and 7.10.9);

k) the **[declaration base URI]** property of the **notation** information item (see 7.11.7).

12 Bit-level encoding of the `Document` type

12.1 This clause specifies special encodings of the `Document` type to form a fast infoset document.

NOTE – These special encodings are designed to optimize speed of processing and compactness, which are considered critical in many expected uses of this Recommendation | International Standard.

12.2 Encodings are specified in terms of actions to be performed by an encoder, resulting in bits being appended to a bit stream. The initial bit stream is either empty or consists of an XML declaration (see 12.3).

12.3 An XML declaration (see W3C XML 1.1, 2.8) may (as a creator's option) be included at the beginning of the bit stream. The XML declaration (if present) shall be one of the following character strings, encoded in UTF-8:

1) **<?xml encoding='finf'?>**
2) **<?xml encoding='finf' standalone='yes'?>**
3) **<?xml encoding='finf' standalone='no'?>**
4) **<?xml version='1.0' encoding='finf'?>**
5) **<?xml version='1.0' encoding='finf' standalone='yes'?>**
6) **<?xml version='1.0' encoding='finf' standalone='no'?>**
7) **<?xml version='1.1' encoding='finf'?>**
8) **<?xml version='1.1' encoding='finf' standalone='yes'?>**
9) **<?xml version='1.1' encoding='finf' standalone='no'?>**

12.4 The version number (if present) in the XML declaration shall be set to the corresponding **[version]** property of the **document** information item. The XML declaration shall not include a version number if the **[version]** property has no value.

12.5 The standalone declaration (if present) in the XML declaration shall be set to the corresponding **[standalone]** property of the **document** information item. The XML declaration shall not include a standalone declaration if the **[standalone]** property has no value.

12.6 The sixteen bits '1110000000000000' shall then be appended to the bit stream.

NOTE – These bits will either occur at the beginning of the fast infoset document or will follow the XML declaration. In the absence of an XML declaration, a parser can distinguish, by looking at the first 16 bits of an encoding, a potential fast infoset document from any well-formed W3C XML 1.0 or W3C XML 1.1 document, because those 16 bits can never occur at the beginning of a well-formed XML document.

12.7 A bit field of sixteen bits containing the version number of this Recommendation | International Standard (see 12.9) encoded as a 16-bit unsigned integer shall then be appended to the bit stream.

12.8 The bit '0' (padding) shall then be appended to the bit stream.

NOTE – This is done to ensure byte alignment in later parts of the encoding.

12.9 The version number of this edition of this Recommendation | International Standard is 1.

NOTE – Further editions of this Recommendation | International Standard are expected to increment the version number if interworking problems between the new edition and previous editions are likely to occur.

12.10 The ECN encoding (see ITU-T Rec. X.692 | ISO/IEC 8825-3) of the abstract value of the **`Document`** type, specified by the Encoding Link Module in A.2, shall be appended to the bit stream.

NOTE – Annex C provides an informal description of the encodings specified in A.2.

12.11 If the encoding of the abstract value of the **`Document`** type does not end on the last bit of an octet, then the four bits '0000' (padding) shall be appended to the bit stream, completing the last octet.

12.12 Once the steps above have been performed, the content of the bit stream is a fast infoset document.

Annex A

ASN.1 module and ECN modules for fast infoset documents

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

A.1 ASN.1 module definition

```
FastInfoset {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
modules(0) fast-infoset(0)}

DEFINITIONS AUTOMATIC TAGS ::= BEGIN

finf-doc-opt-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
            generic-applications(10) fast-infoset(0) encodings(1)
            optional-xml-declaration(0)}

finf-doc-no-decl OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) asn1(1)
            generic-applications(10) fast-infoset(0) encodings(1)
            no-xml-declaration(1)}

Document ::= SEQUENCE {
      additional-data          SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        -- If the initial-vocabulary component is present, at least
                        -- one of its components shall be present -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
      character-encoding-scheme     NonEmptyOctetString OPTIONAL,
      standalone                    BOOLEAN OPTIONAL,
      version                       NonIdentifyingStringOrIndex OPTIONAL
                                      -- OTHER STRING category --,
      children                      SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                     Element,
                  processing-instruction      ProcessingInstruction,
                  comment                     Comment,
                  document-type-declaration DocumentTypeDeclaration }}
```

```
one-meg INTEGER ::= 1048576 -- Two to the power 20

four-gig INTEGER ::= 4294967296 -- Two to the power 32

NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))

URI ::= NonEmptyOctetString

Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (SIZE(1..MAX)) OF
            NamespaceAttribute OPTIONAL,
      qualified-name          QualifiedNameOrIndex
                                    -- ELEMENT NAME category --,
      attributes              SEQUENCE (SIZE(1..MAX)) OF
            Attribute OPTIONAL,
      children                SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                       Element,
                  processing-instruction        ProcessingInstruction,
                  unexpanded-entity-reference   UnexpandedEntityReference,
                  character-chunk               CharacterChunk,
                  comment                       Comment }}

Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                                    -- ATTRIBUTE NAME category --,
      normalized-value    NonIdentifyingStringOrIndex
                                    -- ATTRIBUTE VALUE category -- }

ProcessingInstruction ::= SEQUENCE {
      target       IdentifyingStringOrIndex
                        -- OTHER NCNAME category --,
      content      NonIdentifyingStringOrIndex
                        -- OTHER STRING category -- }

UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category -- }

CharacterChunk ::= SEQUENCE {
      character-codes           NonIdentifyingStringOrIndex
                                    -- CONTENT CHARACTER CHUNK category -- }

Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- OTHER STRING category --}

DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category --,
      children              SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }

UnparsedEntity ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex
                                    -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category --,
      notation-name         IdentifyingStringOrIndex
                                    -- OTHER NCNAME category -- }

Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                    -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                    -- OTHER URI category -- }
```

```
NamespaceAttribute ::= SEQUENCE {
      prefix              IdentifyingStringOrIndex OPTIONAL
                                -- PREFIX category --,
      namespace-name      IdentifyingStringOrIndex OPTIONAL
                                -- NAMESPACE NAME category -- }

IdentifyingStringOrIndex ::= CHOICE {
      literal-character-string   NonEmptyOctetString,
      string-index               INTEGER (1..one-meg) }

NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table               BOOLEAN,
            character-string           EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }

NameSurrogate ::= SEQUENCE {
      prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
      namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
      local-name-string-index       INTEGER(1..one-meg) }
      (CONSTRAINED BY  {-- prefix-string-index shall only be present if
                       -- namespace-name-string-index is present -})

QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex OPTIONAL
                                        -- PREFIX category --,
            namespace-name        IdentifyingStringOrIndex OPTIONAL
                                        -- NAMESPACE NAME category --,
            local-name            IdentifyingStringOrIndex
                                        -- LOCAL NAME category -- },
      name-surrogate-index    INTEGER (1..one-meg) }

EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                 NULL,
            utf-16                NULL,
            restricted-alphabet   INTEGER(1..256),
            encoding-algorithm    INTEGER(1..256) },
      octets                NonEmptyOctetString }

END
```

A.2 ECN module definitions

```
FastInfosetEDM {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
modules(0) fast-infoset-edm(1)}
      ENCODING-DEFINITIONS ::= BEGIN
      EXPORTS FastInfosetEncodingSet;
      RENAMES
            #INTEGER AS #PositiveOrNonNegativeInteger
            IN #IdentifyingStringOrIndex.string-index,
               #NonIdentifyingStringOrIndex.string-index,
               #QualifiedNameOrIndex.name-surrogate-index,
               #NameSurrogate.namespace-name-string-index,
               #NameSurrogate.prefix-string-index,
               #NameSurrogate.local-name-string-index
            FROM FastInfoset;

      /* RENAMES automatically imports:
      #Document, #NonEmptyOctetString, #NameSurrogate, #ProcessingInstruction,
      #UnexpandedEntityReference, #Comment, #DocumentTypeDeclaration,
      #UnparsedEntity, #Notation, #Element, #Attribute, #CharacterChunk,
      #NamespaceAttribute, #IdentifyingStringOrIndex, #NonIdentifyingStringOrIndex,
      #QualifiedNameOrIndex, #EncodedCharacterString FROM FastInfoset;
      */
```

```
-- Useful encoding classes
#PositiveOrNonNegativeInteger ::= #INTEGER

#NonEmptySequenceOfLength ::= #INT(1..1048576)

#NonEmptyOctetStringLength ::= #INT(1..4294967296)

#TwoAlternativeDiscriminant ::= #INT(0..1)

#ThreeAlternativeDiscriminant ::= #INT(0..2)

#FourAlternativeDiscriminant ::= #INT(0..3)

#FiveAlternativeDiscriminant ::= #INT(0..4)

-- Used when encoding the length of a SEQUENCE OF (see C.21)

#NonEmptySequenceOfLengthAlternatives1 ::= #ALTERNATIVES {
      small         #INT(1..128),
      large         #INT(129..1048576) }
-- Used when encoding the length of a NonEmptyOctetString (see C.22)

#NonEmptyOctetStringLengthAlternatives2 ::= #ALTERNATIVES {
      small         #INT(1..64),
      medium        #INT(65..320),
      large         #INT(321..4294967296) }

-- Used when encoding the length of a NonEmptyOctetString (see C.23)

#NonEmptyOctetStringLengthAlternatives5 ::= #ALTERNATIVES {
      small         #INT(1..8),
      medium        #INT(9..264),
      large         #INT(265..4294967296) }

-- Used when encoding the length of a NonEmptyOctetString (see C.24)

#NonEmptyOctetStringLengthAlternatives7 ::= #ALTERNATIVES {
      small         #INT(1..2),
      medium        #INT(3..258),
      large         #INT(259..4294967296) }

-- Used when encoding a positive integer (see C.25)

#PositiveIntegerAlternatives2 ::= #ALTERNATIVES {
      small          #INT(1..64),
      medium         #INT(65..8256),
      large          #INT(8257..1048576) }

-- Used when encoding a positive integer (see C.27)

#PositiveIntegerAlternatives3 ::= #ALTERNATIVES {
      small          #INT(1..32),
      medium         #INT(33..2080),
      medium-large   #INT(2081..526368),
      large          #INT(526369..1048576) }

-- Used when encoding a positive integer (see C.28)

#PositiveIntegerAlternatives4 ::= #ALTERNATIVES {
      small          #INT(1..16),
      medium         #INT(17..1040),
      medium-large   #INT(1041..263184),
      large          #INT(263185..1048576) }

-- Used when encoding a non-negative integer (see C.26)

#NonNegativeIntegerAlternatives2 ::= #ALTERNATIVES {
      zero           #INT(0),
      small          #INT(1..64),
      medium         #INT(65..8256),
      large          #INT(8257..1048576) }

-- Used to insert pre-padding before an encoding in many cases

#PrecededByPrepadding{<#C>} ::= #CONCATENATION {
      prepadding       #PAD,
      original         #C }
```

```
-- Used to insert a two-alternative discriminant before an encoding

#PrecededByTwoAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #TwoAlternativeDiscriminant,
      original        #C }

-- Used to insert a three-alternative discriminant before an encoding

#PrecededByThreeAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      discriminant    #ThreeAlternativeDiscriminant,
      original        #C }

-- Used to insert a four-alternative discriminant before an encoding

#PrecededByFourAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FourAlternativeDiscriminant,
      original        #C }

-- Used to insert a five-alternative discriminant before an encoding

#PrecededByFiveAlternativeDiscriminant{<#C>} ::= #CONCATENATION {
      prepadding      #PAD,
      discriminant    #FiveAlternativeDiscriminant,
      original        #C }

-- Used to insert a length field before the encoding of a SEQUENCE OF

#PrecededByNonEmptySequenceOfLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptySequenceOfLength,
      original        #C }

-- Used to insert a length field before the encoding of a NonEmptyOctetString

#PrecededByNonEmptyOctetStringLength{<#C>} ::= #CONCATENATION {
      length          #NonEmptyOctetStringLength,
      original        #C }

-- Encodes an item of the notations component of the Document
-- type (see C.2.6.1)

eNotationDriver1 #Notation ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNotationPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Encodes an item of the unparsed-entities component of the Document
-- type (see C.2.7.1)

eUnparsedEntityDriver1 #UnparsedEntity ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eUnparsedEntityPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Encodes an item of the namespace-attributes component of the Element
-- type (see C.3.4.2)

eNamespaceAttributeDriver1 #NamespaceAttribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNamespaceAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Encodes an item of the attributes component of the Element
-- type (see C.3.6.1)

eAttributeDriver1 #Attribute ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eAttributePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Encodes an item of the components attribute-values,
-- content-character-chunks, and other-strings of the Document
-- type (see C.2.5.4)

eEncodedCharacterStringDriver1 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eEncodedCharacterStringPrepaddingAdder1 }
      WITH FastInfosetEncodingSet }
```

```
-- Encodes an item of the components element-name-surrogates and
-- attribute-name-surrogates of the Document type (see C.2.5.5)

eNameSurrogateDriver1 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          STRUCTURED WITH eNameSurrogatePrepaddingAdder1 }
      WITH FastInfosetEncodingSet }

-- Encodes the initial-vocabulary component of the Document
-- type (see C.2.5)

eInitialVocabularyPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eInitialVocabularyWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the
-- notations component of the Document type (see C.2.6.1)

eNotationPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNotationWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the
-- unparsed-entities component of the Document type (see C.2.7.1)

eUnparsedEntityPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eUnparsedEntityWithPrepadding1 }

-- Inserts pre-padding before the encoding of the standalone
-- component of the Document type (see C.2.9)

eStandalonePrepaddingAdder1 #BOOL ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eStandaloneWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the
-- children component of the DocumentTypeDeclaration type (see C.9.6)

eDocTypeDeclChildPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eDocTypeDeclChildWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the
-- namespace-attributes component of the Element type (see C.3.4.2)

eNamespaceAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNamespaceAttributeWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the
-- attributes component of the Element type (see C.3.6.1)

eAttributePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eAttributeWithPrepadding1 }

-- Inserts pre-padding before the encoding of the literal-qualified-name
-- component of the QualifiedNameOrIndex type. Used when the encoding
-- starts on the second bit of an octet (see C.17.3)

eLiteralQualifiedNamePrepaddingAdder2 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding2 }

-- Inserts pre-padding before the encoding of the literal-qualified-name
-- component of the QualifiedNameOrIndex type. Used when the encoding
-- starts on the third bit of an octet (see C.18.3)

eLiteralQualifiedNamePrepaddingAdder3 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eLiteralQualifiedNameWithPrepadding3 }
```

```
-- Inserts pre-padding before the encoding of an item of the components
-- restricted-alphabets, encoding-algorithms, prefixes, namespace-names,
-- local-names, other-ncnames, and other-uris of the Document
-- type (see C.2.5.3)

eNonEmptyOctetStringPrepaddingAdder1 #OCTET-STRING ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByPrepadding
          ENCODED BY eNonEmptyOctetStringWithPrepadding1 }}

-- Inserts pre-padding before the encoding of an item of the components
-- attribute-values, content-character-chunks, and other-strings of the
-- Document type (see C.2.5.4)

eEncodedCharacterStringPrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eEncodedCharacterStringWithPrepadding1 }

-- Inserts pre-padding before the encoding of an item of the components
-- element-name-surrogates and attribute-name-surrogates of the Document
-- type (see C.2.5.5)

eNameSurrogatePrepaddingAdder1 #SEQUENCE ::= {
      REPLACE STRUCTURE WITH #PrecededByPrepadding
      ENCODED BY eNameSurrogateWithPrepadding1 }

-- Inserts a discriminant before the encoding of an item of the
-- children component of the Document type (see C.2.11.2 to C.2.11.5)

eDocumentChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eDocumentChildWithDiscriminant1or5 }

-- Inserts a discriminant before the encoding of an item of the
-- children component of the Element type (see C.3.7.2 to C.3.7.6)

eElementChildDiscriminantAdder1or5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFiveAlternativeDiscriminant
      ENCODED BY eElementChildWithDiscriminant1or5 }

-- Inserts a discriminant before the encoding of the length of a SEQUENCE OF,
-- identifying one of the two ways of encoding the length (see C.21)

eNonEmptySequenceOfLengthDiscriminantAdder1 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByTwoAlternativeDiscriminant
      ENCODED BY eNonEmptySequenceOfLengthWithDiscriminant1 }

-- Inserts a discriminant before the encoding of the length of a
-- NonEmptyOctetString, identifying one of the three ways of encoding the
-- length. Used when the encoding starts on the second bit of an octet (see C.22)

eNonEmptyOctetStringLengthDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant2 }

-- Inserts a discriminant before the encoding of the length of a
-- NonEmptyOctetString, identifying one of the three ways of encoding the
-- length. Used when the encoding starts on the fifth bit of an octet (see C.23)

eNonEmptyOctetStringLengthDiscriminantAdder5 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant5 }

-- Inserts a discriminant before the encoding of the length of a
-- NonEmptyOctetString, identifying one of the three ways of encoding the
-- length. Used when the encoding starts on the seventh bit of an octet
-- (see C.24)

eNonEmptyOctetStringLengthDiscriminantAdder7 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY eNonEmptyOctetStringLengthWithDiscriminant7 }
```

```
-- Inserts a discriminant before the encoding of a positive integer,
-- identifying one of the three ways of encoding it. Used when the encoding
-- starts on the second bit of an octet (see C.25)

ePositiveIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByThreeAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant2 }

-- Inserts a discriminant before the encoding of a positive integer,
-- identifying one of the four ways of encoding it. Used when the encoding
-- starts on the third bit of an octet (see C.27)

ePositiveIntegerDiscriminantAdder3 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant3 }

-- Inserts a discriminant before the encoding of a positive integer,
-- identifying one of the four ways of encoding it. Used when the encoding
-- starts on the fourth bit of an octet (see C.28)

ePositiveIntegerDiscriminantAdder4 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY ePositiveIntegerWithDiscriminant4 }

-- Inserts a discriminant before the encoding of a non-negative integer,
-- identifying one of the three ways of encoding it (see C.26)

eNonNegativeIntegerDiscriminantAdder2 #CHOICE ::= {
      REPLACE STRUCTURE WITH #PrecededByFourAlternativeDiscriminant
      ENCODED BY eNonNegativeIntegerWithDiscriminant2 }

-- Sets the prepadding that has been added before the initial-vocabulary
-- component of the Document type and encodes the component (see C.2.5)

eInitialVocabularyWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 3
              PAD-PATTERN bits:'000'B },
          original {
              ENCODE STRUCTURE {
                  restricted-alphabets {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  encoding-algorithms {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  prefixes {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  namespace-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  local-names {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
      }
                      WITH FastInfosetEncodingSet }
                          OPTIONAL-ENCODING USE-SET,
                  other-ncnames {
                      ENCODE STRUCTURE {
```

```
                        STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
        }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-uris {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNonEmptyOctetString1
        }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-values {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                content-character-chunks {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                other-strings {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH
                                eRepetitionWithLengthEncodedCharacterString1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                element-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET,
                attribute-name-surrogates {
                    ENCODE STRUCTURE {
                        STRUCTURED WITH eRepetitionWithLengthNameSurrogate1 }
                    WITH FastInfosetEncodingSet }
                        OPTIONAL-ENCODING USE-SET }
            WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the notations
-- component of the Document type and encodes the item (see C.2.6.1)

eNotationWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'110000'B },
          original eNotation7 }
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the
-- unparsed-entities component of the Document type and encodes the item
-- (see C.2.7.1)

eUnparsedEntityWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'1101000'B },
          original eUnparsedEntity8 }
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before the standalone component of
-- the Document type and encodes the component (see C.2.9)

eStandaloneWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 7
              PAD-PATTERN bits:'0000000'B },
          original USE-SET }
```

```
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the
-- namespace-attributes component of the Element type and encodes the item
-- (see C.3.4.2)

eNamespaceAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'110011'B
              EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
              AS bits:'110011'B },
          original eNamespaceAttribute7
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B
      }}
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the attributes
-- component of the Element type and encodes the item (see C.3.6.1)

eAttributeWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 1
              PAD-PATTERN bits:'0'B },
          original eAttribute2 }
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the
-- children component of the DocumentTypeDeclaration type and encodes the
-- item (see C.9.6)

eDocTypeDeclChildWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 8
              PAD-PATTERN bits:'11000001'B },
          original eProcessingInstruction1 }
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the components
-- restricted-alphabets, encoding-algorithms, prefixes, namespace-names,
-- local-names, other-ncnames, and other-uris of the Document type and encodes
-- the item (see C.2.5.3)

eNonEmptyOctetStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 1
              PAD-PATTERN bits:'0'B },
          original USE-SET }
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before each item of the components
-- attribute-values, content-character-chunks, and other-strings of the
-- Document type and encodes the item (see C.2.5.4)

eEncodedCharacterStringWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 2
              PAD-PATTERN bits:'00'B },
          original USE-SET }
      WITH FastInfosetEncodingSet }


eNameSurrogateWithPrepadding1{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 6
              PAD-PATTERN bits:'000000'B },
          original eNameSurrogate7 }
```

```
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before the literal-qualified-name
-- component of the QualifiedNameOrIndex type and encodes the component.
-- Used when the encoding starts on the second bit of an octet (see C.17.3)

eLiteralQualifiedNameWithPrepadding2{<#C>} #PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 5
              PAD-PATTERN bits:'11110'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }

-- Sets the prepadding that has been added before the literal-qualified-name
-- component of the QualifiedNameOrIndex type and encodes the component. Used
-- when the encoding starts on the third bit of an octet (see C.18.3)

eLiteralQualifiedNameWithPrepadding3{<#C>}
#PrecededByPrepadding{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ENCODING-SPACE SIZE 4
              PAD-PATTERN bits:'1111'B
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B },
          original {
              ENCODE STRUCTURE {
                  prefix eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  namespace-name eIdentifyingStringOrIndex1
                          OPTIONAL-ENCODING USE-SET,
                  local-name eIdentifyingStringOrIndex1 }
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
              ENCODING-SPACE
              EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 } AS bits:'1111'B }}
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a SEQUENCE OF and encodes
-- the SEQUENCE OF NonEmptyOctetString (see C.21)

eNonEmptySequenceOfWithLengthNonEmptyOctetString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNonEmptyOctetStringPrepaddingAdder1
                  STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a SEQUENCE OF and encodes
-- the SEQUENCE OF EncodedCharacterString (see C.21)

eNonEmptySequenceOfWithLengthEncodedCharacterString1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eEncodedCharacterStringDriver1
                  STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
```

```
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a SEQUENCE OF and encodes
-- the SEQUENCE OF NameSurrogate (see C.21)

eNonEmptySequenceOfWithLengthNameSurrogate1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  eNameSurrogateDriver1
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a SEQUENCE OF and encodes
-- the SEQUENCE OF additional-datum (see C.21)

eNonEmptySequenceOfWithLengthAdditionalDatum1{<#C>}
#PrecededByNonEmptySequenceOfLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptySequenceOfLength1,
          original {
              ENCODE STRUCTURE {
                  additional-datum {
                       ENCODE STRUCTURE {
                            id eNonEmptyOctetStringPrepaddingAdder1,
                            data eNonEmptyOctetStringPrepaddingAdder1 }
                       WITH FastInfosetEncodingSet}
                 STRUCTURED WITH eRepetitionItems1{<length>}
              } WITH PER-BASIC-UNALIGNED }}
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a NonEmptyOctetString
-- and encodes the NonEmptyOctetString. Used when the encoding starts on the
-- second bit of an octet (see C.22)

eNonEmptyOctetStringWithLength2{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength2,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a NonEmptyOctetString
-- and encodes the NonEmptyOctetString. Used when the encoding starts on the
-- fifth bit of an octet (see C.23)

eNonEmptyOctetStringWithLength5{<#C>}
      #PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength5,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Encodes the length field that has been added before a NonEmptyOctetString
-- and encodes the NonEmptyOctetString. Used when the encoding starts on the
-- seventh bit of an octet (see C.24)

eNonEmptyOctetStringWithLength7{<#C>}
#PrecededByNonEmptyOctetStringLength{<#C>} ::= {
      ENCODE STRUCTURE {
          length eNonEmptyOctetStringLength7,
          original eOctetStringOctets1{<length>} }
      WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before an item of the children
-- component of the Document type and encodes the item (see C.2.11.2 to C.2.11.5)

eDocumentChildWithDiscriminant1or5{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          prepadding {
              ALIGNED TO NEXT octet
```

```
            ENCODING-SPACE SIZE 0 },
        discriminant {
            USE #BIT-STRING
             MAPPING TO BITS {
                0 TO '0'B,
                1 TO '11100001'B,
                2 TO '11100010'B,
                3 TO '110001'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                element eElement2,
                processing-instruction eProcessingInstruction1,
                comment eComment1,
                document-type-declaration eDocumentTypeDeclaration7
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before an item of the children
-- component of the Element type and encodes the item (see C.3.7.2 to C.3.7.6)

eElementChildWithDiscriminant1or5{<#C>}
#PrecededByFiveAlternativeDiscriminant{<#C>} ::= {
    ENCODE STRUCTURE {
        prepadding {
            ALIGNED TO NEXT octet
            ENCODING-SPACE SIZE 0 },
        discriminant {
            USE #BIT-STRING
            MAPPING TO BITS {
                0 TO '0'B,
                1 TO '11100001'B,
                2 TO '110010'B,
                3 TO '10'B,
                4 TO '11100010'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                element eElement2,
                processing-instruction eProcessingInstruction1,
                unexpanded-entity-reference eUnexpandedEntityReference7,
                character-chunk eCharacterChunk3,
                comment eComment1
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
                WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before the length of a
-- SEQUENCE OF (identifying one of the two ways of encoding the length)
-- and encodes the length (see C.21)

eNonEmptySequenceOfLengthWithDiscriminant1{<#C>}
#PrecededByTwoAlternativeDiscriminant{<#C>} ::= {
    ENCODE STRUCTURE {
        discriminant {
            USE #BIT-STRING
            MAPPING TO BITS {
                0 TO '0'B,
                1 TO '1000'B }
            WITH FastInfosetEncodingSet },
        original {
            ENCODE STRUCTURE {
                STRUCTURED WITH {
                    ALTERNATIVE DETERMINED BY field-to-be-set
                    USING discriminant }}
            WITH FastInfosetEncodingSet }}
    WITH FastInfosetEncodingSet }
```

```
-- Encodes the discriminant that has been added before the length of a
-- NonEmptyOctetString (identifying one of the three ways of encoding the
-- length) and encodes the length. Used when the encoding starts on the
-- second bit of an octet (see C.22)

eNonEmptyOctetStringLengthWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000000'B,
                  2 TO '1100000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before the length of a
-- NonEmptyOctetString (identifying one of the three ways of encoding the
-- length) and encodes the length. Used when the encoding starts on the fifth
-- bit of an octet (see C.23)

eNonEmptyOctetStringLengthWithDiscriminant5{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '1000'B,
                  2 TO '1100'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before the length of
-- a NonEmptyOctetString (identifying one of the three ways of encoding
-- the length) and encodes the length. Used when the encoding starts on
-- the seventh bit of an octet (see C.24)

eNonEmptyOctetStringLengthWithDiscriminant7{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '11'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }
```

```
-- Encodes the discriminant that has been added before a positive integer
-- (identifying one of the three ways of encoding the integer) and encodes
-- the integer. Used when the encoding starts on the second bit of an octet
-- (see C.25)

ePositiveIntegerWithDiscriminant2{<#C>}
#PrecededByThreeAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '10'B,
                  2 TO '110'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 12}}}  -- Less than '1110'B
      WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before a positive integer
-- (identifying one of the four ways of encoding the integer) and encodes
-- the integer. Used when the encoding starts on the third bit of an octet
-- (see C.27)

ePositiveIntegerWithDiscriminant3{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
                  STRUCTURED WITH {
                      ALTERNATIVE DETERMINED BY field-to-be-set
                      USING discriminant }}
              WITH FastInfosetEncodingSet }
          STRUCTURED WITH {
                  ENCODING-SPACE SIZE self-delimiting-values
                  EXHIBITS HANDLE "qn" AT { 0 | 1 | 2 | 3 }
                  AS range:{low 0, high 14}}}  -- Less than '1111'B
      WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before a positive integer
-- (identifying one of the four ways of encoding the integer) and encodes
-- the integer. Used when the encoding starts on the fourth bit of an octet
-- (see C.28)

ePositiveIntegerWithDiscriminant4{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
      ENCODE STRUCTURE {
          discriminant {
              USE #BIT-STRING
              MAPPING TO BITS {
                  0 TO '0'B,
                  1 TO '100'B,
                  2 TO '101'B,
                  3 TO '110000000'B }
              WITH FastInfosetEncodingSet },
          original {
              ENCODE STRUCTURE {
```

```
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

-- Encodes the discriminant that has been added before a non-negative integer
-- (identifying one of the three ways of encoding the integer) and encodes
-- the integer (see C.26)

eNonNegativeIntegerWithDiscriminant2{<#C>}
#PrecededByFourAlternativeDiscriminant{<#C>} ::= {
        ENCODE STRUCTURE {
            discriminant {
                USE #BIT-STRING
                MAPPING TO BITS {
                    0 TO '1111111'B,
                    1 TO '0'B,
                    2 TO '10'B,
                    3 TO '110'B }
                WITH FastInfosetEncodingSet },
            original {
                ENCODE STRUCTURE {
                    STRUCTURED WITH {
                        ALTERNATIVE DETERMINED BY field-to-be-set
                        USING discriminant }}
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

-- Encodes the Document type (see C.2)

eDocument2 #Document ::= {
        ENCODE STRUCTURE {
            additional-data {
                ENCODE STRUCTURE {
                    STRUCTURED WITH eRepetitionWithLengthAdditionalDatum1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            initial-vocabulary {
                ENCODE STRUCTURE {
                    STRUCTURED WITH eInitialVocabularyPrepaddingAdder1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            notations {
                ENCODE STRUCTURE {
                    eNotationDriver1
                    STRUCTURED WITH eRepetitionWithTerminator8bit1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            unparsed-entities {
                ENCODE STRUCTURE {
                    eUnparsedEntityDriver1
                    STRUCTURED WITH eRepetitionWithTerminator8bit1 }
                WITH FastInfosetEncodingSet }
                    OPTIONAL-ENCODING USE-SET,
            character-encoding-scheme eNonEmptyOctetStringPrepaddingAdder1
                OPTIONAL-ENCODING USE-SET,
            standalone eStandalonePrepaddingAdder1
                OPTIONAL-ENCODING USE-SET,
            version eNonIdentifyingStringOrIndex1
                OPTIONAL-ENCODING USE-SET,
            children {
                ENCODE STRUCTURE {
                    {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH eDocumentChildDiscriminantAdder1or5 }
                        WITH FastInfosetEncodingSet }
                    STRUCTURED WITH eRepetitionWithTerminator4bit1 }
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }
```

```
-- Encodes the Element type (see C.3)

eElement2 #Element ::= {
      ENCODE STRUCTURE {
          namespace-attributes {
              ENCODE STRUCTURE {
                  eNamespaceAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator10bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING eNamespaceAttributesOptionality3,
          qualified-name eQualifiedNameOrIndex3,
          attributes {
              ENCODE STRUCTURE {
                  eAttributeDriver1
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }
                  OPTIONAL-ENCODING USE-SET,
          children {
              ENCODE STRUCTURE {
                  {
                      ENCODE STRUCTURE {
                          STRUCTURED WITH eElementChildDiscriminantAdder1or5 }
                      WITH FastInfosetEncodingSet }
                  STRUCTURED WITH eRepetitionWithTerminator4bit1 }
              WITH FastInfosetEncodingSet }}
      WITH FastInfosetEncodingSet }

-- Encodes the Attribute type (see C.4)

eAttribute2 #Attribute ::= {
      ENCODE STRUCTURE {
          qualified-name eQualifiedNameOrIndex2,
          normalized-value eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Encodes the ProcessingInstruction type (see C.5)

eProcessingInstruction1 #ProcessingInstruction ::= {
      ENCODE STRUCTURE {
          target eIdentifyingStringOrIndex1,
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Encodes the UnexpandedEntityReference type (see C.6)

eUnexpandedEntityReference7 #UnexpandedEntityReference ::= {
      ENCODE STRUCTURE {
          name eIdentifyingStringOrIndex1,
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET }
      WITH FastInfosetEncodingSet }

-- Encodes the CharacterChunk type (see C.7)

eCharacterChunk3 #CharacterChunk ::= {
      ENCODE STRUCTURE {
          character-codes eNonIdentifyingStringOrIndex3 }
      WITH FastInfosetEncodingSet }

-- Encodes the Comment type (see C.8)

eComment1 #Comment ::= {
      ENCODE STRUCTURE {
          content eNonIdentifyingStringOrIndex1 }
      WITH FastInfosetEncodingSet }

-- Encodes the DocumentTypeDeclaration type (see C.9)

eDocumentTypeDeclaration7 #DocumentTypeDeclaration ::= {
      ENCODE STRUCTURE {
          system-identifier eIdentifyingStringOrIndex1
                  OPTIONAL-ENCODING USE-SET,
          public-identifier eIdentifyingStringOrIndex1
```

```
                OPTIONAL-ENCODING USE-SET,
            children {
                ENCODE STRUCTURE {
                    {
                        ENCODE STRUCTURE {
                            STRUCTURED WITH eDocTypeDeclChildPrepaddingAdder1 }
                        WITH FastInfosetEncodingSet }
                    STRUCTURED WITH eRepetitionWithTerminator4bit1 }
                WITH FastInfosetEncodingSet }}
        WITH FastInfosetEncodingSet }

    -- Encodes the UnparsedEntity type (see C.10)

    eUnparsedEntity8 #UnparsedEntity ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1,
            public-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            notation-name eIdentifyingStringOrIndex1 }
        WITH FastInfosetEncodingSet }

    -- Encodes the Notation type (see C.11)

    eNotation7 #Notation ::= {
        ENCODE STRUCTURE {
            name eIdentifyingStringOrIndex1,
            system-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            public-identifier eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }

    -- Encodes the NamespaceAttribute type (see C.12)

    eNamespaceAttribute7 #NamespaceAttribute ::= {
        ENCODE STRUCTURE {
            prefix eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET,
            namespace-name eIdentifyingStringOrIndex1
                    OPTIONAL-ENCODING USE-SET }
        WITH FastInfosetEncodingSet }

    -- Encodes the IdentifyingStringOrIndex type (see C.13)

    eIdentifyingStringOrIndex1 #IdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string eNonEmptyOctetString2,
            string-index ePositiveInteger2 }
        WITH FastInfosetEncodingSet }

    -- Encodes the NonIdentifyingStringOrIndex type. Used when the encoding starts
    -- on the first bit of an octet (see C.14)

    eNonIdentifyingStringOrIndex1 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString3 }
                WITH FastInfosetEncodingSet },
            string-index eNonNegativeInteger2 }
        WITH FastInfosetEncodingSet }

    -- Encodes the NonIdentifyingStringOrIndex type. Used when the encoding starts
    -- on the third bit of an octet (see C.15)

    eNonIdentifyingStringOrIndex3 #NonIdentifyingStringOrIndex ::= {
        ENCODE STRUCTURE {
            literal-character-string {
                ENCODE STRUCTURE {
                    add-to-table USE-SET,
                    character-string eEncodedCharacterString5 }
                WITH FastInfosetEncodingSet },
            string-index ePositiveInteger4 }
```

```
      WITH FastInfosetEncodingSet }

-- Encodes the NameSurrogate type (see C.16)

eNameSurrogate7 #NameSurrogate ::= {
      ENCODE STRUCTURE {
          prefix-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          namespace-name-string-index ePositiveInteger2
                  OPTIONAL-ENCODING USE-SET,
          local-name-string-index ePositiveInteger2 }
      WITH FastInfosetEncodingSet }

-- Encodes the QualifiedNameOrIndex type. Used when the encoding starts
-- on the second bit of an octet (see C.17)

eQualifiedNameOrIndex2 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder2 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger2
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }

-- Encodes the QualifiedNameOrIndex type. Used when the encoding starts
-- on the third bit of an octet (see C.18)

eQualifiedNameOrIndex3 #QualifiedNameOrIndex ::= {
      ENCODE STRUCTURE {
          literal-qualified-name {
              ENCODE STRUCTURE {
                  STRUCTURED WITH eLiteralQualifiedNamePrepaddingAdder3 }
              WITH FastInfosetEncodingSet },
          name-surrogate-index ePositiveInteger3
          STRUCTURED WITH eQualifiedNameAlternatives3 }
      WITH FastInfosetEncodingSet }

-- Encodes the EncodedCharacterString type. Used when the encoding starts
-- on the third bit of an octet (see C.19)

eEncodedCharacterString3 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString5 }
      WITH FastInfosetEncodingSet }

-- Encodes the EncodedCharacterString type. Used when the encoding starts
-- on the fifth bit of an octet (see C.20)

eEncodedCharacterString5 #EncodedCharacterString ::= {
      ENCODE STRUCTURE {
          encoding-format USE-SET,
          octets eNonEmptyOctetString7 }
      WITH FastInfosetEncodingSet }

-- Encodes a repetition (SEQUENCE OF NonEmptyOctetString) by inserting a length
-- field before it (see C.2.5.3 to C.2.5.5)

eRepetitionWithLengthNonEmptyOctetString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNonEmptyOctetString1 }}

-- Encodes a repetition (SEQUENCE OF EncodedCharacterString) by inserting a length
-- field before it (see C.2.5.3 to C.2.5.5)

eRepetitionWithLengthEncodedCharacterString1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthEncodedCharacterString1 }}
```

```
-- Encodes a repetition (SEQUENCE OF NameSurrogate) by inserting a length
-- field before it (see C.2.5.3 to C.2.5.5)

eRepetitionWithLengthNameSurrogate1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthNameSurrogate1 }}

-- Encodes a repetition (SEQUENCE OF additional-datum) by inserting a length
-- field before it (see C.2.5.3 to C.2.5.5)

eRepetitionWithLengthAdditionalDatum1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptySequenceOfLength
          ENCODED BY eNonEmptySequenceOfWithLengthAdditionalDatum1 }}

-- Encodes the NonEmptyOctetString type. Used when the encoding starts
-- on the second bit of an octet (see C.22)

eNonEmptyOctetString2 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength2 }}

-- Encodes the NonEmptyOctetString type. Used when the encoding starts
-- on the fifth bit of an octet (see C.23)

eNonEmptyOctetString5 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength5 }}

-- Encodes the NonEmptyOctetString type. Used when the encoding starts
-- on the seventh bit of an octet (see C.24)

eNonEmptyOctetString7 #NonEmptyOctetString ::= {
      REPETITION-ENCODING {
          REPLACE STRUCTURE WITH #PrecededByNonEmptyOctetStringLength
          ENCODED BY eNonEmptyOctetStringWithLength7 }}

-- Encodes the length field that has been inserted before the encoding of
-- a SEQUENCE OF (see C.21)

eNonEmptySequenceOfLength1 #NonEmptySequenceOfLength ::= {
      USE #NonEmptySequenceOfLengthAlternatives1
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptySequenceOfLengthDiscriminantAdder1 }
          WITH FastInfosetEncodingSet }}

-- Encodes the length field that has been inserted before the encoding of
-- a NonEmptyOctetString. Used when the encoding starts on the second bit
-- of an octet (see C.22)

eNonEmptyOctetStringLength2 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Encodes the length field that has been inserted before the encoding of a
-- NonEmptyOctetString. Used when the encoding starts on the fifth bit of
-- an octet (see C.23)

eNonEmptyOctetStringLength5 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives5
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder5 }
          WITH FastInfosetEncodingSet }}
```

```
-- Encodes the length field that has been inserted before the encoding of
-- a NonEmptyOctetString. Used when the encoding starts on the seventh bit
-- of an octet (see C.24)

eNonEmptyOctetStringLength7 #NonEmptyOctetStringLength ::= {
      USE #NonEmptyOctetStringLengthAlternatives7
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonEmptyOctetStringLengthDiscriminantAdder7 }
          WITH FastInfosetEncodingSet }}

-- Encodes a positive integer. Used when the encoding starts on the second bit
-- of an octet (see C.25)

ePositiveInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Encodes a positive integer. Used when the encoding starts on the third bit
-- of an octet (see C.27)

ePositiveInteger3 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives3
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder3 }
          WITH FastInfosetEncodingSet }}

-- Encodes a positive integer. Used when the encoding starts on the fourth bit
-- of an octet (see C.28)

ePositiveInteger4 #PositiveOrNonNegativeInteger ::= {
      USE #PositiveIntegerAlternatives4
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH ePositiveIntegerDiscriminantAdder4 }
          WITH FastInfosetEncodingSet }}

-- Encodes a non-negative integer (see C.26)

eNonNegativeInteger2 #PositiveOrNonNegativeInteger ::= {
      USE #NonNegativeIntegerAlternatives2
      MAPPING ORDERED VALUES
      WITH {
          ENCODE STRUCTURE {
              STRUCTURED WITH eNonNegativeIntegerDiscriminantAdder2 }
          WITH FastInfosetEncodingSet }}

-- Specifies how to determine the presence of the namespace-attributes component
-- of the Element type (see C.3.4.2)

eNamespaceAttributesOptionality3 #OPTIONAL ::= {
      PRESENCE DETERMINED BY handle
      HANDLE "nsa" }

-- Specifies how to determine the alternative of the QualifiedNameOrIndex type
-- (see C.17.3 and C.18.3)

eQualifiedNameAlternatives3 #ALTERNATIVES ::= {
      ALTERNATIVE DETERMINED BY handle
      HANDLE "qn"
      EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5}
      AS range:{low 0, high 50}}   -- Less than '110011'B

-- Specifies how to determine the termination of a repetition using a 4-bit
-- terminator '1111'(see C.2.12, C.3.6.2, C.3.8, and C.9.7)
```

```
eRepetitionWithTerminator4bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111'B }}

-- Specifies how to determine the termination of a repetition using an 8-bit
-- terminator '11110000'(see C.2.6.2 and C.2.7.2)

eRepetitionWithTerminator8bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'11110000'B }}

-- Specifies how to determine the termination of a repetition using a 10-bit
-- terminator '1111000000'(see C.3.4.3)

eRepetitionWithTerminator10bit1 #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant
          DETERMINED BY pattern PATTERN bits:'1111000000'B
          EXHIBITS HANDLE "nsa" AT { 0 | 1 | 2 | 3 | 4 | 5} AS bits:'110011'B }}

-- Encodes the items of a SEQUENCE OF, following the length field that has been
-- added (see C.2.5.3 to C.2.5.5)

eRepetitionItems1{<REFERENCE:len>} #REPETITION ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}

-- Encodes the octets of a NonEmptyOctetString, following the length field that
-- has been added (see C.22, C.23, and C.24)

eOctetStringOctets1{<REFERENCE:len>} #OCTETS ::= {
      REPETITION-ENCODING {
          REPETITION-SPACE SIZE variable-with-determinant MULTIPLE OF bit
          DETERMINED BY field-to-be-set USING len }}
empty-padding #PAD ::= {
       ENCODING-SPACE SIZE 0
}

FastInfosetEncodingSet #ENCODINGS::= { eDocument2 | empty-padding }
      COMPLETED BY PER-BASIC-UNALIGNED

END

FastInfosetELM
      {joint-iso-itu-t(2) asn1(1) generic-applications(10) fast-infoset(0)
      modules(0) fast-infoset-elm(2)}
      LINK-DEFINITIONS ::= BEGIN
      IMPORTS FastInfosetEncodingSet, Document FROM FastInfosetEDM;
      ENCODE #Document WITH FastInfosetEncodingSet

END
```

Annex B

The MIME media type for fast infoset documents

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

This annex defines the "application/fastinfoset" media type that describes fast infoset documents.

The MIME media type is specified below using the IETF MIME registration template, and has been registered in accordance with IETF procedures.

```
MIME media type name:
       application

MIME subtype name:
       fastinfoset

Required parameters:
       None.

Optional parameters:
       None.

Encoding considerations:
       XML infosets encoded as fast infoset documents will result in the production of
       binary data. This MIME media type may require further encoding on transports not
       capable of handling binary data.

Security considerations:
       Because XML infosets encoded as fast infoset documents can carry application
       defined data whose semantics is independent from that of any MIME wrapper (or
       context within which the MIME wrapper is used), one should not expect to be able
       to understand the semantics of the fast infoset document based on the semantics of
       the MIME wrapper alone. Therefore, whenever using the
       "application/fastinfoset" media type, it is strongly recommended that the security
       implications of the context within which the fast infoset document is used is
       fully understood.

Interoperability considerations:
       There are no known interoperability issues.

Published specification:
       ITU-T Rec. X.891 | ISO/IEC 24824-1

Applications which use this media type:
       No known applications currently use this media type.

Additional information:

       Magic number(s):
              A fast infoset document can begin with an optional XML declaration that
              shall be one of the following strings encoded in UTF-8:

              <?xml encoding='finf'?>
              <?xml encoding='finf' standalone='yes'?>
              <?xml encoding='finf' standalone='no'?>
              <?xml version='1.0' encoding='finf'?>
              <?xml version='1.0' encoding='finf' standalone='yes'?>
              <?xml version='1.0' encoding='finf' standalone='no'?>
              <?xml version='1.1' encoding='finf'?>
              <?xml version='1.1' encoding='finf' standalone='yes'?>
              <?xml version='1.1' encoding='finf' standalone='no'?>

              The first five octets of the XML declaration encoded in UTF-8 are
              hexadecimal 3C 3F 78 6D 6C. The four octets identifying a fast infoset
              document corresponding to the substring "finf" encoded in UTF-8 are
              hexadecimal 66 69 6E 66.

              A fast infoset document shall begin with an octet sequence of hexadecimal E0
              00 00 01 if the optional XML declaration is absent.


       File extension(s):
              *.finf
```

Person & email address to contact for further information:
ITU-T ASN.1 Rapporteur (contact via tsbmail@itu.int)
ISO/IEC JTC1/SC6 ASN.1 Rapporteur (contact via ittf@iso.org)

Intended usage:
COMMON

Author/Change controller:
Joint ITU-T | ISO/IEC balloting procedures in accordance with ITU-T Rec. A.23 *Collaboration with the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) on information technology*, Annex A and ISO/IEC JTC1 Directives, Annex K.

Annex C

Description of the encoding of a fast infoset document

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

C.1 Fast infoset document

C.1.1 This annex informally (but precisely and fully) describes the encodings that are specified in clause 12 and Annex A. For the convenience of implementers, all ASN.1 type definitions in the normative text are copied into this annex rather than simply being referenced.

C.1.2 Encodings are described in terms of actions to be performed by an encoder, resulting in bits being appended to a bit stream. The actions to be performed by a decoder are not described explicitly in this annex, but can be inferred from the encoder's actions described in this annex.

C.1.3 A fast infoset document may begin either with an XML declaration (see 12.3) followed by:

a) the sixteen bits '1111000000000000' (identification); followed by

b) the sixteen bits '0000000000000001' (version number); followed by

c) the bit '0' (padding),

or with the same thirty-three bits with no preceding XML declaration. The thirty-three bits are immediately followed by the encoding of an abstract value of the **Document** type, as described in C.2. This encoding ends either on the eighth or on the fourth bit of an octet, depending on the content of the fast infoset document. In the latter case, the four bits '0000' (padding) are appended to the bit stream.

C.2 Encoding of the Document type

C.2.1 The **Document** type is defined in 7.2 as follows:

```
Document ::= SEQUENCE {
      additional-data          SEQUENCE (SIZE(1..one-meg)) OF
            additional-datum SEQUENCE {
                  id                    URI,
                  data                  NonEmptyOctetString } OPTIONAL,
      initial-vocabulary       SEQUENCE {
            external-vocabulary         URI OPTIONAL,
            restricted-alphabets        SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            encoding-algorithms         SEQUENCE (SIZE(1..256)) OF
                  NonEmptyOctetString OPTIONAL,
            prefixes                    SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            namespace-names             SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            local-names                 SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-ncnames               SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            other-uris                  SEQUENCE (SIZE(1..one-meg)) OF
                  NonEmptyOctetString OPTIONAL,
            attribute-values            SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            content-character-chunks    SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            other-strings               SEQUENCE (SIZE(1..one-meg)) OF
                  EncodedCharacterString OPTIONAL,
            element-name-surrogates     SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL,
            attribute-name-surrogates   SEQUENCE (SIZE(1..one-meg)) OF
                  NameSurrogate OPTIONAL }
                  (CONSTRAINED BY {
                        --  If the initial-vocabulary component is present, at least
                        -- one of its components shall be present -- }) OPTIONAL,
      notations                     SEQUENCE (SIZE(1..MAX)) OF
                                          Notation OPTIONAL,
      unparsed-entities             SEQUENCE (SIZE(1..MAX)) OF
                                          UnparsedEntity OPTIONAL,
```

```
character-encoding-scheme  NonEmptyOctetString OPTIONAL,
standalone                 BOOLEAN OPTIONAL,
version                    NonIdentifyingStringOrIndex OPTIONAL
                             -- OTHER STRING category --,
children                   SEQUENCE (SIZE(0..MAX)) OF
      CHOICE {
            element                    Element,
            processing-instruction     ProcessingInstruction,
            comment                    Comment,
            document-type-declaration DocumentTypeDeclaration }}
```

C.2.2 A value of the **Document** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the second bit of an octet and ends on either the fourth or the eighth bit of another octet (which is the last bit of the terminator '1111' described in C.2.12).

C.2.3 For each of the seven optional components **additional-data**, **initial-vocabulary**, **notations**, **unparsed-entities**, **character-encoding-scheme**, **standalone**, and **version** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.2.4 If the optional component **additional-data** is present, then the number of **additional-datum** components is encoded as described in C.21, and each of the **additional-datum** components is encoded as described in the two following subclauses.

C.2.4.1 The bit '0' (padding) is appended to the bit stream and the **id** component is encoded as described in C.22.

C.2.4.2 The bit '0' (padding) is appended to the bit stream and the **data** component is encoded as described in C.22.

C.2.5 If the optional component **initial-vocabulary** is present, then the three bits '000' (padding) are appended to the bit stream, and the component is encoded as described in the five following subclauses.

C.2.5.1 For each of the thirteen optional components of **initial-vocabulary** (in textual order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.2.5.2 If the optional component **external-vocabulary** of **initial-vocabulary** is present, then the bit '0' (padding) is appended to the bit stream and the component is encoded as described in C.22.

C.2.5.3 For each of the components **restricted-alphabets**, **encoding-algorithms**, **prefixes**, **namespace-names**, **local-names**, **other-ncnames**, and **other-uris** (in this order) which is present, the number of **NonEmptyOctetString** items in the component is encoded as described in C.21, and then each item is encoded (in order) as follows: The bit '0' (padding) is appended to the bit stream, and the **NonEmptyOctetString** is encoded as described in C.22.

C.2.5.4 For each of the components **attribute-values**, **content-character-chunks**, and **other-strings** (in this order) which is present, the number of **EncodedCharacterString** items in the component is encoded as described in C.21, and then each item is encoded (in order) as follows: The two bits '00' (padding) are appended to the bit stream, and the **EncodedCharacterString** is encoded as described in C.19.

C.2.5.5 For each of the components **element-name-surrogates** and **attribute-name-surrogates** (in this order) which is present, the number of **NameSurrogate** items in the component is encoded as described in C.21, and then each item is encoded (in order) as follows: The six bits '000000' (padding) are appended to the bit stream, and the **NameSurrogate** is encoded as described in C.16.

C.2.6 If the optional component **notations** is present, it is encoded as described in the two following subclauses.

C.2.6.1 Each item of **notations** (in order) is encoded as follows: The six bits '110000' (identification) are appended to the bit stream, and the **Notation** is encoded as described in C.11.

C.2.6.2 The four bits '1111' (termination) and the four bits '0000' (padding) are appended to the bit stream.

NOTE – These bits are not appended if the component **notations** is absent.

C.2.7 If the optional component **unparsed-entities** is present, it is encoded as described in the two following subclauses.

C.2.7.1 Each item of **unparsed-entities** (in order) is encoded as follows: The seven bits '1101000' (identification) are appended to the bit stream, and the **UnparsedEntity** is encoded as described in C.10.

C.2.7.2 The four bits '1111' (termination) and the four bits '0000' (padding) are appended to the bit stream.

NOTE – These bits are not appended if the component **unparsed-entities** is absent.

C.2.8 If the optional component **character-encoding-scheme** is present, then the bit '0' (padding) is appended to the bit stream, and the **NonEmptyOctetString** is encoded as described in C.22.

C.2.9 If the optional component **standalone** is present, it is encoded as follows: The seven bits '0000000' (padding) are appended to the bit stream. If the value of **standalone** is **TRUE**, then the bit '1' is appended to the bit stream, otherwise the bit '0' is appended.

C.2.10 If the optional component **version** is present, then its value is encoded as described in C.14.

C.2.11 If the component **children** has one or more items, then each item is encoded (in order) as described in the five following subclauses.

C.2.11.1 The encoding of each item is required to start on the first bit of an octet. However, the latest bit appended may have been either the eight or the fourth bit of an octet. If it was the fourth bit of an octet, the bits '0000' (padding) are appended to the bit stream so that the encoding of the item starts on the first bit of the next octet.

C.2.11.2 If the alternative **element** is present, then the bit '0' (identification) is appended to the bit stream, then the **element** is encoded as described in C.3.

C.2.11.3 If the alternative **processing-instruction** is present, then the eight bits '11100001' (identification) are appended to the bit stream, and the **processing-instruction** is encoded as described in C.5.

C.2.11.4 If the alternative **comment** is present, then the eight bits '11100010' (identification) are appended to the bit stream, and the **comment** is encoded as described in C.8.

C.2.11.5 If the alternative **document-type-declaration** is present, then the six bits '110001' (identification) are appended to the bit stream, and the **document-type-declaration** is encoded as described in C.9.

C.2.12 The four bits '1111' (termination) are appended.

NOTE – These bits are appended even if the component **children** has no items.

C.3 Encoding of the Element type

C.3.1 The **Element** type is defined in 7.3 as follows:

```
Element ::= SEQUENCE {
      namespace-attributes    SEQUENCE (SIZE(1..MAX)) OF
            NamespaceAttribute OPTIONAL,
      qualified-name          QualifiedNameOrIndex
                                    -- ELEMENT NAME category --,
      attributes              SEQUENCE (SIZE(1..MAX)) OF
            Attribute OPTIONAL,
      children                SEQUENCE (SIZE(0..MAX)) OF
            CHOICE {
                  element                         Element,
                  processing-instruction          ProcessingInstruction,
                  unexpanded-entity-reference     UnexpandedEntityReference,
                  character-chunk                 CharacterChunk,
                  comment                         Comment }}
```

C.3.2 A value of the **Element** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the second bit of an octet and ends on either the fourth or the eighth bit of another octet (which is the last bit of the terminator '1111' described in C.3.8).

C.3.3 If the optional component **attributes** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.3.4 If the optional component **namespace-attributes** is present, it is encoded as described in the three following subclauses.

C.3.4.1 The four bits '1110' (presence) and the two bits '00' (padding) are appended to the bit stream.

C.3.4.2 Each item of **namespace-attributes** (in order) is encoded as follows: The six bits '110011' (identification) are appended to the bit stream, and the **NamespaceAttribute** is encoded as described in C.12.

C.3.4.3 The four bits '1111' (termination) and the six bits '000000' (padding) are appended.

NOTE – These bits are not appended if the component **namespace-attributes** is absent.

C.3.5 The value of the component **qualified-name** is encoded as described in C.18.

C.3.6 If the optional component **attributes** is present, it is encoded as described in the two following subclauses.

C.3.6.1 Each item of **attributes** (in order) is encoded as follows: The bit '0' (identification) is appended to the bit stream, and the **Attribute** is encoded as described in C.4.

C.3.6.2 The four bits '1111' (termination) are appended.

NOTE – These bits are not appended if the component **attributes** is absent.

C.3.7 If the component **children** has one or more items, then each item is encoded (in order) as described in the six following subclauses.

C.3.7.1 The encoding of each item is required to start on the first bit of an octet. However, the latest bit appended may have been either the eighth or the fourth bit of an octet. If it was the fourth bit of an octet, the bits '0000' (padding) are appended so that the encoding of the item starts on the first bit of the next octet.

C.3.7.2 If the alternative **element** is present, then the bit '0' (identification) is appended to the bit stream, and the **element** is encoded as described in this subclause C.3.

C.3.7.3 If the alternative **processing-instruction** is present, then the eight bits '11100001' (identification) are appended to the bit stream, and the **processing-instruction** is encoded as described in C.5.

C.3.7.4 If the alternative **unexpanded-entity-reference** is present, then the six bits '110010' (identification) are appended to the bit stream, and the **unexpanded-entity-reference** is encoded as described in C.6.

C.3.7.5 If the alternative **character-chunk** is present, then the two bits '10' (identification) are appended to the bit stream, and the **character-chunk** is encoded as described in C.7.

C.3.7.6 If the alternative **comment** is present, then the eight bits '11100010' (identification) are appended to the bit stream, and the **comment** is encoded as described in C.8.

C.3.8 The four bits '1111' (termination) are appended.

NOTE – These bits are appended even if the component **children** has no items.

C.4 Encoding of the Attribute type

C.4.1 The **Attribute** type is defined in 7.4 as follows:

```
Attribute ::= SEQUENCE {
      qualified-name      QualifiedNameOrIndex
                              -- ATTRIBUTE NAME category --,
      normalized-value    NonIdentifyingStringOrIndex
                              -- ATTRIBUTE VALUE category -- }
```

C.4.2 A value of the **Attribute** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the second bit of an octet and ends on the eighth bit of another octet.

C.4.3 The value of **qualified-name** is encoded as described in C.17.

C.4.4 The value of **normalized-value** is encoded as described in C.14.

C.5 Encoding of the ProcessingInstruction type

C.5.1 The **ProcessingInstruction** type is defined in 7.5 as follows:

```
ProcessingInstruction ::= SEQUENCE {
      target        IdentifyingStringOrIndex
                        -- OTHER NCNAME category --,
      content       NonIdentifyingStringOrIndex
                        -- OTHER STRING category -- }
```

C.5.2 A value of the **ProcessingInstruction** type is encoded by performing the following actions (in order).

NOTE - An encoding of this type always starts on the first bit of an octet and ends on the eighth bit of another octet.

C.5.3 The value of **target** is encoded as described in C.13.

C.5.4 The value of **content** is encoded as described in C.14.

C.6 Encoding of the `UnexpandedEntityReference` type

C.6.1 The **`UnexpandedEntityReference`** type is defined in 7.6 as follows:

```
UnexpandedEntityReference ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                  -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI category -- }
```

C.6.2 A value of the **`UnexpandedEntityReference`** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the seventh bit of an octet and ends on the eighth bit of another octet.

C.6.3 For each of the optional components **`system-identifier`** and **`public-identifier`** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream, otherwise the bit '0' (absence) is appended.

C.6.4 The value of **`name`** is encoded as described in C.13.

C.6.5 If the optional component **`system-identifier`** is present, it is encoded as described in C.13.

C.6.6 If the optional component **`public-identifier`** is present, it is encoded as described in C.13.

C.7 Encoding of the `CharacterChunk` type

C.7.1 The **`CharacterChunk`** type is defined in 7.7 as follows:

```
CharacterChunk ::= SEQUENCE {
      character-codes             NonIdentifyingStringOrIndex
                                        -- CONTENT CHARACTER CHUNK category -- }
```

C.7.2 A value of the **`CharacterChunk`** type is encoded by performing the following action.

NOTE – An encoding of this type always starts on the third bit of an octet and ends on the eighth bit of the same or another octet.

C.7.3 The value of **`character-codes`** is encoded as described in C.15.

C.8 Encoding of the `Comment` type

C.8.1 The **`Comment`** type is defined in 7.8 as follows:

```
Comment ::= SEQUENCE {
      content     NonIdentifyingStringOrIndex -- OTHER STRING category --}
```

C.8.2 A value of the **`Comment`** type is encoded by performing the following action.

NOTE – An encoding of this type always starts on the first bit of an octet and ends on the eighth bit of the same or another octet.

C.8.3 The value of **`content`** is encoded as described in C.14.

C.9 Encoding of the `DocumentTypeDeclaration` type

C.9.1 The **`DocumentTypeDeclaration`** type is defined in 7.9 as follows:

```
DocumentTypeDeclaration ::= SEQUENCE {
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                  -- OTHER URI category --,
      children              SEQUENCE (SIZE(0..MAX)) OF
            ProcessingInstruction }
```

C.9.2 A value of the **`DocumentTypeDeclaration`** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the seventh bit of an octet and ends on the fourth bit of another octet (which is the last bit of the terminator '1111' described in C.9.7).

C.9.3 For each of the optional components **`system-identifier`** and **`public-identifier`** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.9.4 If the optional component **system-identifier** is present, it is encoded as described in C.13.

C.9.5 If the optional component **public-identifier** is present, it is encoded as described in C.13.

C.9.6 If the component **children** has one or more items, then each item is encoded as follows: The eight bits '11100001' (identification) are appended to the bit stream, and the **ProcessingInstruction** is encoded as described in C.5.

C.9.7 The four bits '1111' (termination) are appended.

NOTE – These bits are appended even if the component **children** has no items.

C.10 Encoding of the UnparsedEntity type

C.10.1 The **UnparsedEntity** type is defined in 7.10 as follows:

```
UnparsedEntity ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex
                                -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category --,
      notation-name         IdentifyingStringOrIndex
                                -- OTHER NCNAME category -- }
```

C.10.2 A value of the **UnparsedEntity** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the eighth bit of an octet and ends on the eighth bit of another octet.

C.10.3 If the optional component **public-identifier** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.10.4 The value of **name** is encoded as described in C.13.

C.10.5 The value of **system-identifier** is encoded as described in C.13.

C.10.6 If the optional component **public-identifier** is present, it is encoded as described in C.13.

C.10.7 The value of **name** is encoded as described in C.13.

C.11 Encoding of the Notation type

C.11.1 The **Notation** type is defined in 7.11 as follows:

```
Notation ::= SEQUENCE {
      name                  IdentifyingStringOrIndex
                                -- OTHER NCNAME category --,
      system-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category --,
      public-identifier     IdentifyingStringOrIndex OPTIONAL
                                -- OTHER URI category -- }
```

C.11.2 A value of the **Notation** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the seventh bit of an octet and ends on the eighth bit of another octet.

C.11.3 For each of the optional components **system-identifier** and **public-identifier** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.11.4 The value of **name** is encoded as described in C.13.

C.11.5 If the optional component **system-identifier** is present, it is encoded as described in C.13.

C.11.6 If the optional component **public-identifier** is present, it is encoded as described in C.13.

C.12 Encoding of the NamespaceAttribute type

C.12.1 The **NamespaceAttribute** type is defined in 7.12 as follows:

```
NamespaceAttribute ::= SEQUENCE {
      prefix              IdentifyingStringOrIndex OPTIONAL
                                -- PREFIX category --,
      namespace-name      IdentifyingStringOrIndex OPTIONAL
                                -- NAMESPACE NAME category -- }
```

C.12.2 A value of the **NamespaceAttribute** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the eighth bit of an octet and ends on the eighth bit of another octet.

C.12.3 If the optional component **prefix** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.12.4 If the optional component **namespace-name** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.12.5 If the optional component **prefix** is present, it is encoded as described in C.13.

C.12.6 If the optional component **namespace-name** is present, it is encoded as described in C.13.

C.13 Encoding of the IdentifyingStringOrIndex type

C.13.1 The **IdentifyingStringOrIndex** type is defined in 7.13 as follows:

```
IdentifyingStringOrIndex ::= CHOICE {
      literal-character-string   NonEmptyOctetString,
      string-index               INTEGER (1..one-meg) }
```

C.13.2 A value of the **IdentifyingStringOrIndex** type is encoded by performing the following actions.

NOTE – An encoding of this type always starts on the first bit of an octet and ends on the eighth bit of the same or another octet.

C.13.3 If the alternative **literal-character-string** is present, then the bit '0' (discriminant) is appended to the bit stream, and the **literal-character-string** is encoded as described in C.22.

C.13.4 If the alternative **string-index** is present, then the bit '1' (discriminant) is appended to the bit stream, and the **string-index** is encoded as described in C.25

C.14 Encoding of the NonIdentifyingStringOrIndex type starting on the first bit of an octet

C.14.1 The **NonIdentifyingStringOrIndex** type is defined in 7.14 as follows:

```
NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table                BOOLEAN,
            character-string            EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }
```

C.14.2 This subclause C.14 is invoked to encode a value of the **NonIdentifyingStringOrIndex** type when the encoding is to start on the first bit of an octet (see also C.15). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of the same or another octet.

C.14.3 If the alternative **literal-character-string** is present, then the bit '0' (discriminant) is appended to the bit stream, and the **literal-character-string** is encoded as described in the two following subclauses.

C.14.3.1 If the value of the component **add-to-table** is **TRUE**, then the bit '1' is appended to the bit stream; otherwise, the bit '0' is appended.

C.14.3.2 The value of the component **character-string** is encoded as described in C.19.

C.14.4 If the alternative **string-index** is present, then the bit '1' (discriminant) is appended to the bit stream, and the **string-index** is encoded as described in C.26.

C.15 Encoding of the NonIdentifyingStringOrIndex type starting on the third bit of an octet

C.15.1 The **NonIdentifyingStringOrIndex** type is defined in 7.14 as follows:

```
NonIdentifyingStringOrIndex ::= CHOICE {
      literal-character-string    SEQUENCE {
            add-to-table                BOOLEAN,
            character-string            EncodedCharacterString },
      string-index                INTEGER (0..one-meg) }
```

C.15.2 This subclause C.15 is invoked to encode a value of the **NonIdentifyingStringOrIndex** type when the encoding is to start on the third bit of an octet (see also C.14). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of the same or another octet.

C.15.3 If the alternative **literal-character-string** is present, then the bit '0' (discriminant) is appended to the bit stream, and the **literal-character-string** is encoded as described in the two following subclauses.

C.15.3.1 If the value of the component **add-to-table** is **TRUE**, then the bit '1' is appended to the bit stream, otherwise the bit '0' is appended.

C.15.3.2 The value of the component **character-string** is encoded as described in C.20.

C.15.4 If the alternative **string-index** is present, then the bit '1' (discriminant) is appended to the bit stream, and the **string-index** is encoded as described in C.28.

C.16 Encoding of the NameSurrogate type

C.16.1 The **NameSurrogate** type is defined in 7.15 as follows:

```
NameSurrogate ::= SEQUENCE {
      prefix-string-index           INTEGER(1..one-meg) OPTIONAL,
      namespace-name-string-index   INTEGER(1..one-meg) OPTIONAL,
      local-name-string-index       INTEGER(1..one-meg) }
      (CONSTRAINED BY  {-- prefix-string-index shall only be present if
                       -- namespace-name-string-index is present --})
```

C.16.2 A value of the **NameSurrogate** type is encoded by performing the following actions (in order).

NOTE – An encoding of this type always starts on the seventh bit of an octet and ends on the eighth bit of another octet.

C.16.3 If the optional component **prefix-string-index** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.16.4 If the optional component **namespace-name-string-index** is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.16.5 If the optional component **prefix-string-index** is present, then the bit '0' (padding) is appended to the bit stream, and the component is encoded as described in C.25.

C.16.6 If the optional component **namespace-name-string-index** is present, then the bit '0' (padding) is appended to the bit stream, and the component is encoded as described in C.25.

C.16.7 The bit '0' (padding) is appended to the bit stream, and the component **local-name-string-index** is encoded as described in C.25.

C.17 Encoding of the QualifiedNameOrIndex type starting on the second bit of an octet

C.17.1 The **QualifiedNameOrIndex** type is defined in 7.16 as follows:

```
QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                  IdentifyingStringOrIndex OPTIONAL
                                          -- PREFIX category --,
            namespace-name          IdentifyingStringOrIndex OPTIONAL
                                          -- NAMESPACE NAME category --,
            local-name              IdentifyingStringOrIndex
                                          -- LOCAL NAME category -- },
      name-surrogate-index    INTEGER (1..one-meg) }
```

C.17.2 This subclause C.17 is invoked to encode a value of the **QualifiedNameOrIndex** type when the encoding is to start on the second bit of an octet (see also C.18). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of the same or another octet.

C.17.3 If the alternative **literal-qualified-name** is present, then the four bits '1111' (identification) and the bit '0' (padding) are appended to the bit stream, and the **literal-qualified-name** is encoded as described in the four following subclauses.

C.17.3.1 For each of the optional components **prefix** and **namespace-name** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.17.3.2 If the optional component **prefix** is present, it is encoded as described in C.13.

C.17.3.3 If the optional component **namespace-name** is present, it is encoded as described in C.13.

C.17.3.4 The component **local-name** is encoded as described in C.13.

C.17.4 If the alternative **name-surrogate-index** is present, it is encoded as described in C.25.

C.18 Encoding of the QualifiedNameOrIndex type starting on the third bit of an octet

C.18.1 The **QualifiedNameOrIndex** type is defined in 7.16 as follows:

```
QualifiedNameOrIndex ::= CHOICE {
      literal-qualified-name  SEQUENCE {
            prefix                IdentifyingStringOrIndex OPTIONAL
                                        -- PREFIX category --,
            namespace-name        IdentifyingStringOrIndex OPTIONAL
                                        -- NAMESPACE NAME category --,
            local-name            IdentifyingStringOrIndex
                                        -- LOCAL NAME category -- },
      name-surrogate-index    INTEGER (1..one-meg) }
```

C.18.2 This subclause C.18 is invoked to encode a value of the **QualifiedNameOrIndex** type when the encoding is to start on the third bit of an octet (see also C.17). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of the same or another octet.

C.18.3 If the alternative **literal-qualified-name** is present, then the four bits '1111' (identification) are appended to the bit stream, and the **literal-qualified-name** is encoded as described in the four following subclauses.

C.18.3.1 For each of the optional components **prefix** and **namespace-name** (in this order), if the component is present, then the bit '1' (presence) is appended to the bit stream; otherwise, the bit '0' (absence) is appended.

C.18.3.2 If the optional component **prefix** is present, it is encoded as described in C.13.

C.18.3.3 If the optional component **namespace-name** is present, it is encoded as described in C.13.

C.18.3.4 The component **local-name** is encoded as described in C.13.

C.18.4 If the alternative **name-surrogate-index** is present, it is encoded as described in C.27.

C.19 Encoding of the EncodedCharacterString type starting on the third bit of an octet

C.19.1 The **EncodedCharacterString** type is defined in 7.17 as follows:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                 NULL,
            utf-16                NULL,
            restricted-alphabet   INTEGER(1..256),
            encoding-algorithm    INTEGER(1..256) },
      octets                NonEmptyOctetString }
```

C.19.2 This subclause C.19 is invoked to encode a value of the **EncodedCharacterString** type when the encoding is to start on the third bit of an octet (see also C.20). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of another octet.

C.19.3 The value of the component **encoding-format** is encoded as described in the four following subclauses.

C.19.3.1 If the alternative **utf-8** is present, then the two bits '00' (discriminant) are appended to the bit stream.

C.19.3.2 If the alternative **utf-16** is present, then the two bits '01' (discriminant) are appended to the bit stream.

C.19.3.3 If the alternative **restricted-alphabet** is present, then the two bits '10' (discriminant) are appended to the bit stream, and the **restricted-alphabet** is encoded as described in C.29.

C.19.3.4 If the alternative **encoding-algorithm** is present, then the two bits '11' (discriminant) are appended to the bit stream, and the **encoding-algorithm** is encoded as described in C.29.

C.19.4 The component **octets** is encoded as described in C.23.

C.20 Encoding of the EncodedCharacterString type starting on the fifth bit of an octet

C.20.1 The **EncodedCharacterString** type is defined in 7.17 as follows:

```
EncodedCharacterString ::= SEQUENCE {
      encoding-format       CHOICE {
            utf-8                 NULL,
            utf-16                NULL,
            restricted-alphabet   INTEGER(1..256),
            encoding-algorithm    INTEGER(1..256) },
      octets                NonEmptyOctetString }
```

C.20.2 This subclause C.20 is invoked to encode a value of the **EncodedCharacterString** type when the encoding is to start on the fifth bit of an octet (see also C.19). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of another octet.

C.20.3 The value of the component **encoding-format** is encoded as described in the four following subclauses.

C.20.3.1 If the alternative **utf-8** is present, then the two bits '00' (discriminant) are appended to the bit stream.

C.20.3.2 If the alternative **utf-16** is present, then the two bits '01' (discriminant) are appended to the bit stream.

C.20.3.3 If the alternative **restricted-alphabet** is present, then the two bits '10' (discriminant) are appended to the bit stream, and the **restricted-alphabet** is encoded as described in C.29.

C.20.3.4 If the alternative **encoding-algorithm** is present, then the two bits '11' (discriminant) are appended to the bit stream, and the **encoding-algorithm** is encoded as described in C.29.

C.20.4 The component **octets** is encoded as described in C.24.

C.21 Encoding of the length of a sequence-of type

C.21.1 This subclause is invoked to encode the length of a sequence-of type that is encoded with a length field preceding the items of the sequence-of type.

NOTE – This encoding always starts on the first bit of an octet and ends on the eighth bit of the same or another octet.

C.21.2 If the value is in the range 1 to 128, then the bit '0' is appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of seven bits and appended.

C.21.3 If the value is in the range 129 to 2^{20}, the bit '1' and the three bits '000' (padding) are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of twenty bits and appended.

C.22 Encoding of the NonEmptyOctetString type starting on the second bit of an octet

C.22.1 The **NonEmptyOctetString** type is defined in 7.2 as follows:

```
NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))
```

C.22.2 This subclause C.22 is invoked to encode a value of the **NonEmptyOctetString** type when the encoding is to start on the second bit of an octet (see also C.23 and C.24). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of another octet.

C.22.3 The length of the octet string is encoded as described in the three following subclauses.

C.22.3.1 If the length is in the range 1 to 64, then the bit '0' is appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of six bits and appended.

C.22.3.2 If the length is in the range 65 to 320, the two bits '10' and the five bits '00000' (padding) are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of eight bits and appended.

C.22.3.3 If the length is in the range 321 to 2^{32}, the two bits '11' and the five bits '00000' (padding) are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of thirty-two bits and appended.

C.22.4 The bits forming the octets of the octet string are appended to the bit stream (in order).

C.23 Encoding of the `NonEmptyOctetString` starting on the fifth bit of an octet

C.23.1 The **`NonEmptyOctetString`** type is defined in 7.2 as follows:

```
NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))
```

C.23.2 This subclause C.23 is invoked to encode a value of the **`NonEmptyOctetString`** type when the encoding is to start on the fifth bit of an octet (see also C.22 and C.24). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of another octet.

C.23.3 The length of the octet string is encoded as described in the three following subclauses.

C.23.3.1 If the length is in the range 1 to 8, then the bit '0' is appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of three bits and appended.

C.23.3.2 If the length is in the range 9 to 264, the two bits '10' and the two bits '00' (padding) are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of eight bits and appended.

C.23.3.3 If the length is in the range 265 to 2^{32}, the two bits '11' and the two bits '00' (padding) are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of thirty-two bits and appended.

C.23.4 The bits forming the octets of the octet string are appended to the bit stream (in order).

C.24 Encoding of the `NonEmptyOctetString` type starting on the seventh bit of an octet

C.24.1 The **`NonEmptyOctetString`** is defined in 7.2 as follows:

```
NonEmptyOctetString ::= OCTET STRING (SIZE(1..four-gig))
```

C.24.2 This subclause C.24 is invoked to encode a value of the **`NonEmptyOctetString`** type when the encoding is to start on the seventh bit of an octet (see also C.22 and C.23). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of another octet.

C.24.3 The length of the octet string is encoded as described in the three following subclauses.

C.24.3.1 If the length is in the range 1 to 2, then the bit '0' is appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of one bit and appended.

C.24.3.2 If the length is in the range 3 to 258, the two bits '10' are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of eight bits and appended.

C.24.3.3 If the length is in the range 259 to 2^{32}, the two bits '11' are appended to the bit stream, and the length, minus the lower bound of the range, is encoded as an unsigned integer in a field of thirty-two bits and appended.

C.24.4 The bits forming the octets of the octet string are appended to the bit stream (in order).

C.25 Encoding of integers in the range 1 to 2^{20} starting on the second bit of an octet

C.25.1 This subclause C.25 is invoked to encode an integer value in the range 1 to 2^{20} when the encoding is to start on the second bit of an octet (see also C.26, C.27, and C.28). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of either the same or another octet.

C.25.2 If the value is in the range 1 to 64, then the bit '0' is appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of six bits and appended.

C.25.3 If the value is in the range 65 to 8256, the two bits '10' are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of thirteen bits and appended.

C.25.4 If the value is in the range 8257 to 2^{20}, the two bits '11' and the bit '0' (padding) are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of twenty bits and appended.

C.26 Encoding of integers in the range 0 to 2^{20} starting on the second bit of an octet

C.26.1 This subclause C.26 is invoked to encode an integer value in the range 0 to 2^{20} when the encoding is to start on the second bit of an octet (see also C.25, C.27, and C.28). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of either the same or another octet.

C.26.2 If the value is zero, then the seven bits '1111111' are appended to the bit stream. Otherwise, the value is encoded as described in C.25.

C.27 Encoding of integers in the range 1 to 2^{20} starting on the third bit of an octet

C.27.1 This subclause C.27 is invoked to encode an integer value in the range 1 to 2^{20} when the encoding is to start on the third bit of an octet (see also C.25, C.26, and C.28). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of either the same or another octet.

C.27.2 If the value is in the range 1 to 32, then the bit '0' is appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of five bits and appended.

C.27.3 If the value is in the range 33 to 2080, the three bits '100' are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of eleven bits and appended.

C.27.4 If the value is in the range 2081 to 526368, the three bits '101' are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of nineteen bits and appended.

C.27.5 If the value is in the range 526369 to 2^{20}, the three bits '110' and the seven bits '0000000' (padding) are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of twenty bits and appended.

C.28 Encoding of integers in the range 1 to 2^{20} starting on the fourth bit of an octet

C.28.1 This subclause C.28 is invoked to encode an integer value in the range 1 to 2^{20} when the encoding is to start on the fourth bit of an octet (see also C.25, C.26, and C.27). The value is encoded by performing the following actions (in order).

NOTE – An encoding of this type always ends on the eighth bit of either the same or another octet.

C.28.2 If the value is in the range 1 to 16, then the bit '0' is appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of four bits and appended.

C.28.3 If the value is in the range 17 to 1040, the three bits '100' are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of ten bits and appended.

C.28.4 If the value is in the range 1041 to 263184, the three bits '101' are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of eighteen bits and appended.

C.28.5 If the value is in the range 263185 to 2^{20}, the three bits '110' and the six bits '000000' (padding) are appended to the bit stream, and the value, minus the lower bound of the range, is encoded as an unsigned integer in a field of twenty bits and appended.

C.29 Encoding of integers in the range 1 to 256

C.29.1 This subclause C.29 is invoked to encode an integer value in the range 1 to 256.

NOTE – An encoding of this type always starts on either the fifth bit or the seventh bit of an octet and ends on the fourth bit or the sixth bit (respectively) of the following octet.

C.29.2 The value, minus the lower bound of the range, is encoded as an unsigned integer in a field of eight bits and appended to the bit stream.

Annex D

Examples of encoding XML infosets as fast infoset documents

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

D.1 Introduction of examples

D.1.1 This annex uses the following typographical conventions for numbers:

a) for a number represented in base ten **`bold Courier`** is used for the digits of the number, followed by the subscript "10" (for example, $\mathbf{11}_{10}$); and

b) for a number represented in base sixteen (a hexadecimal number) **`bold Courier`** is used for the digits of the number, followed by the subscript "16" (for example, $\mathbf{0b1f}_{16}$); and

c) if the base of a number is explicitly stated, then the subscript is omitted.

D.1.2 This annex presents two examples of possible encodings of a Universal Business Language (UBL) [1] order into a fast infoset document. UBL is designed to provide a universally understood and recognized commercial syntax for legally binding business documents.

D.1.3 The XML infoset for the example UBL order is presented in D.3.

D.1.4 The first fast infoset document has an initial vocabulary that references an external vocabulary. Subclause D.4 describes the content of the external vocabulary, the octets of the fast infoset document, and explanations of some octet sequences.

D.1.5 The second fast infoset has no initial vocabulary. Subclause D.5 describes the octets of that fast infoset document, and explanations of some octet sequences.

NOTE – The final vocabulary of this fast infoset document is the same as the final vocabulary of the fast infoset document described in D.4.

D.1.6 The octets of D.4 and D.5 are presented in a series of tables each with two columns. The first column lists the starting position in hexadecimal of 32 consecutive octets of the fast infoset document, and the second column lists the octets in hexadecimal notation. Those hexadecimal characters containing bits that correspond to the identification and termination of information items are underlined.

D.1.7 The explanations of some octet sequences of the fast infoset documents (in D.4 and D.5) are presented in tables with the following columns:

a) Column 1 presents the position, in hexadecimal, of the octet(s) listed in column 2.

b) Column 2 presents the octet(s) of the fast infoset document associated with a relevant information item and the item's properties. An octet is represented in base two followed by the same octet represented in base sixteen (hexadecimal) in brackets, for example, **`11110000 (f0)`**.

c) Column 3 presents, in detail, a description of the octets in column 2, and references subclauses in Annex C for further explanation and clarification.

d) Column 4 presents a portion of the XML infoset or a portion of the XML 1.0 document (if applicable) corresponding to the octet(s) in column 2.

D.1.8 In these examples, all chunks of **character** information items containing less than 6 characters are added to the CONTENT CHARACTER CHUNK table, and the **[normalized value]** property of all **attribute** information items containing less than 6 characters are added to the ATTRIBUTE VALUE table.

D.1.9 The sizes of the XML 1.0 document and of the fast infoset documents, and the compressed sizes (using GZIP) of those documents are listed in D.2

D.2 Size of example documents (including redundancy-based compression)

D.2.1 Table D.1 presents the sizes of all documents. Column 1 lists the UBL documents, column 2 lists the document sizes, and column 3 lists the GZIP (with default options) [2] compressed sizes of documents.

NOTE 1 – The UBL Order XML 1.0 document contains no white spaces (see D.3.1.2).

NOTE 2 – For each document all characters are encoded using the UTF-8 character encoding.

NOTE 3 – No XML declaration (see 12.3) is serialized for the fast infoset documents.

Table D.1 – Initial sizes and GZIP compressed sizes of documents

UBL document	Size	GZIP compressed size
XML 1.0 document	3311	893
Fast infoset document with an external vocabulary	684	546
Fast infoset document with no initial vocabulary	1322	860

D.2.2 The size of the fast infoset document with a reference to an external vocabulary is the smallest in size, and also the smallest in GZIP compressed size. The ratio of GZIP compressed size over the size of the fast infoset document implies that this fast infoset document has little redundant information.

D.2.3 In all cases the GZIP compressed sizes of the fast infoset documents are smaller than the GZIP compressed size of the XML 1.0 document. Furthermore, the size of the fast infoset document with a reference to an external vocabulary is smaller than the GZIP compressed size of the XML 1.0 document.

D.3 UBL order example

D.3.1 Joinery Order example

D.3.1.1 The UBL order example is taken from [1]. Specifically, the Joinery Order example has been chosen (see xml/joinery/UBL-Order-1.0-Joinery-Example.xml) for the following reasons:

a) it is a real world example developed independently of this Recommendation | International Standard with no particular bias towards Fast Infoset;

b) it is freely available; and

c) it makes extensive use of XML namespaces and thus is a good example to present how Fast Infoset supports XML namespaces.

D.3.1.2 The Joinery Order example has been modified with the following:

a) the last three **OrderLine** elements have been removed; and

NOTE 1 – This reduces the XML 1.0 document to reasonable size for presentation in this Recommendation | International Standard.

b) all white spaces have been removed.

NOTE 2 – This represents a more realistic use case for XML infosets that may be serialized, transmitted over a network, and parsed.

D.3.2 Joinery Order XML 1.0 document

The Joinery Order XML 1.0 document with the modifications as stated in D.3.1.2 a, but with white spaces retained for readability, is presented as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<Order xmlns:res="urn:oasis:names:tc:ubl:codelist:AcknowledgementResponseCode:1:0"
xmlns:cbc="urn:oasis:names:tc:ubl:CommonBasicComponents:1:0"
xmlns:cac="urn:oasis:names:tc:ubl:CommonAggregateComponents:1:0"
xmlns:cur="urn:oasis:names:tc:ubl:codelist:CurrencyCode:1:0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:oasis:names:tc:ubl:Order:1:0"
xsi:schemaLocation="urn:oasis:names:tc:ubl:Order:1:0 ../../xsd/maindoc/UBL-Order-1.0.xsd">
        <BuyersID>S03-034257</BuyersID>
        <cbc:IssueDate>2003-02-03</cbc:IssueDate>
        <cac:BuyerParty>
                <cac:Party>
                        <cac:PartyName>
                                <cbc:Name>Jerry Builder plc</cbc:Name>
                        </cac:PartyName>
                        <cac:Address>
                                <cbc:StreetName>Marsh Lane</cbc:StreetName>
                                <cbc:CityName>Nowhere</cbc:CityName>
                                <cbc:PostalZone>NR18 4XX</cbc:PostalZone>
                                <cbc:CountrySubentity>Norfolk</cbc:CountrySubentity>
                        </cac:Address>
                        <cac:Contact>
                                <cbc:Name>Eva Brick</cbc:Name>
                        </cac:Contact>
                </cac:Party>
        </cac:BuyerParty>
        <cac:SellerParty>
```

```
                <cac:Party>
                    <cac:PartyName>
                        <cbc:Name>Specialist Windows plc</cbc:Name>
                    </cac:PartyName>
                    <cac:Address>
                        <cbc:BuildingName>Snowhill Works</cbc:BuildingName>
                        <cbc:CityName>Little Snoring</cbc:CityName>
                        <cbc:PostalZone>SM2 3NW</cbc:PostalZone>
                        <cbc:CountrySubentity>Whereshire</cbc:CountrySubentity>
                    </cac:Address>
                </cac:Party>
            </cac:SellerParty>
            <cac:Delivery>
                <cbc:RequestedDeliveryDateTime>2003-02-24T00:00:00</cbc:RequestedDeliveryDateTime>
                <cac:DeliveryAddress>
                    <cbc:StreetName>Riverside Rd.</cbc:StreetName>
                    <cbc:BuildingName>Plot 17, Whitewater Estate</cbc:BuildingName>
                    <cbc:CityName>Whetstone</cbc:CityName>
                    <cbc:CountrySubentity>Middlesex</cbc:CountrySubentity>
                </cac:DeliveryAddress>
            </cac:Delivery>
            <cac:OrderLine>
                <cac:LineItem>
                    <cac:BuyersID>A</cac:BuyersID>
                    <cbc:Quantity quantityUnitCode="unit">2</cbc:Quantity>
                    <cac:Item>
                        <cac:SellersItemIdentification>
                            <cac:ID>236WV</cac:ID>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>wood</cac:AttributeID>
                                <cbc:Description>soft</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>finish</cac:AttributeID>
                                <cbc:Description>primed</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>fittings</cac:AttributeID>
                                <cbc:Description>satin</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>glazing</cac:AttributeID>
                                <cbc:Description>single</cbc:Description>
                            </cac:PhysicalAttribute>
                        </cac:SellersItemIdentification>
                    </cac:Item>
                </cac:LineItem>
            </cac:OrderLine>
            <cac:OrderLine>
                <cac:LineItem>
                    <cac:BuyersID>B</cac:BuyersID>
                    <cbc:Quantity quantityUnitCode="unit">3</cbc:Quantity>
                    <cac:Item>
                        <cac:SellersItemIdentification>
                            <cac:ID>340TW</cac:ID>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>hand</cac:AttributeID>
                                <cbc:Description>RH</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>wood</cac:AttributeID>
                                <cbc:Description>hard</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>finish</cac:AttributeID>
                                <cbc:Description>stain</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>fittings</cac:AttributeID>
                                <cbc:Description>brass</cbc:Description>
                            </cac:PhysicalAttribute>
                            <cac:PhysicalAttribute>
                                <cac:AttributeID>glazing</cac:AttributeID>
                                <cbc:Description>double</cbc:Description>
                            </cac:PhysicalAttribute>
```

```
                    </cac:SellersItemIdentification>
                </cac:Item>
            </cac:LineItem>
        </cac:OrderLine>
    </Order>
```

D.4 UBL Order fast infoset document with an external vocabulary

The external vocabulary of the fast infoset document is presented in D.4.1. The octets (as hexadecimal characters) of the fast infoset document are presented in D.4.2. Detailed explanations of some octet sequences in D.4.2 are presented in D.4.3. The fast infoset document cannot be considered self-describing because external information is required (the external vocabulary) to produce complete XML infoset.

NOTE – The fast infoset document can still be processed by a fast infoset parser that cannot obtain the vocabulary tables given the URI but vocabulary table indexes cannot be de-referenced to obtain the necessary information to generate properties of information items.

D.4.1 The UBL Order external vocabulary

D.4.1.1 The external vocabulary of the fast infoset document is specified to be the final vocabulary obtained from the example UBL order XML infoset (see D.3.1.2) that is further modified to contain:

a) no **character** information items; and

b) empty **[normalized value]** properties of the **attribute** information items.

NOTE 1 – This represents a realistic scenario where it is not known in advance what the application-defined content (**character** information items and or **[normalized value]** properties of the **attribute** information items) of an XML infoset will be.

NOTE 2 – In practice it is not expected that the document to be serialized will be used to generate the external vocabulary. It is anticipated that tools will make use of schema, and potentially XML infoset instances of the schema for frequency analysis of strings and qualified names such that smaller index values will be assigned to more frequently occurring information (for example, the frequency of **[local name]** properties in XML infosets may obey a power law series).

D.4.1.2 The URI of the external vocabulary is **urn:oasis:names:tc:ubl:Order:1.0:joinery:example**.

D.4.1.3 Table D.2 presents the vocabulary of the UBL Order XML infoset (the vocabulary tables). Column 1 lists the vocabulary table indexes of the vocabulary tables (index), column 2 lists the vocabulary table entries of the PREFIX table (prefix entry), column 3 lists the vocabulary table entries of the NAMESPACE NAME table (namespace name entry), column 4 lists the vocabulary table entries of the LOCAL NAME table (local name entry), column 5 lists the vocabulary table entries of the ELEMENT NAME table (element name entry), column 6 lists the vocabulary table entries of the ATTRIBUTE NAME table (attribute name entry). The index values for the name surrogate entries, of the ELEMENT NAME and ATTRIBUTE NAME tables, are presented in the order as specified for the components of the `NameSurrogate` type (`prefix-name-string-index`, `namespace-name-string-index` and `local-name-string-index`). A character of "_" specifies that the value is absent (which only occurs for values of the `prefix-name-string-index` and `namespace-name-string-index` components).

NOTE 1 – The first entry (index `1`) for the prefix and namespace name corresponding to the XML prefix, "xml", and the XML namespace name, "http://www.w3.org/XML/1998/namespace", are built-in (see 7.2.21 and 7.2.22).

NOTE 2 – Long namespaces name entries (URIs) have been truncated.

NOTE 3 – For the first element name entry (index `1`) there is no reference to a prefix (since the value is absent, represented by "_"), there is a reference to the seventh namespace name entry (index `7`) for the **[namespace name]** property ("urn:oasis:names:tc:ubl:Order:1:0"), and there is a reference to the first local name entry (index `1`) for the **[local name]** property ("Order").

Table D.2 – Vocabulary of the UBL Order XML infoset

Index	Prefix entry	Namespace name entry	Local name entry	Element name entry	Attribute name entry
1	xml	http://www.w3.org/XML/1998/namespace	Order	_ 7 1	6 6 2
2	res	….AcknowledgementResponseCode:1:0	schemaLocation	_ 7 3	_ _ 23
3	cbc	….CommonBasicComponents:1:0	BuyersID	3 3 4	
4	cac	….CommonAggregateComponents:1:0	IssueDate	4 4 5	
5	cur	….CurrencyCode:1:0	BuyerParty	4 4 6	
6	xsi	….XMLSchema-instance	Party	4 4 7	
7		….Order:1:0	PartyName	3 3 8	
8			Name	4 4 9	
9			Address	3 3 10	
10			StreetName	3 3 11	
11			CityName	3 3 12	
12			PostalZone	3 3 13	
13			CountrySubentity	4 4 14	
14			Contact	4 4 15	
15			SellerParty	3 3 16	
16			BuildingName	4 4 17	
17			Delivery	3 3 18	
18			RequestedDeliveryDateTime	4 4 19	
19			DeliveryAddress	4 4 20	
20			OrderLine	4 4 21	
21			LineItem	4 4 3	
22			Quantity	3 3 22	
23			quantityUnitCode	4 4 24	
24			Item	4 4 25	
25			SellersItemIdentification	4 4 26	
26			ID	4 4 27	
27			PhysicalAttribute	4 4 28	
28			AttributeID	3 3 29	
29			Description		

D.4.2 Octets (as hexadecimal characters) of the fast infoset document

Table D.3 presents the octets of the fast infoset document for the UBL order example presented in D.3.

NOTE – Hexadecimal characters containing bits that correspond to the identification and termination of information items are underlined.