INTERNATIONAL STANDARD

ISO/IEC 23001-1

First edition
2006-04-01

Information technology — MPEG systems technologies —

Part 1: Binary MPEG format for XML

Technologies de l'information — Technologies des systèmes MPEG — Partie 1: Format binaire de MPEG pour XML

Reference number
ISO/IEC 23001-1:2006(E)

Contents

Page

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.

ISO/IEC 23001-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23001 consists of the following parts, under the general title *Information technology — MPEG systems technologies*:

— *Part 1: Binary MPEG format for XML*

Introduction

This International Standard provides a standardized set of generic technologies for encoding XML documents. It addresses a broad spectrum of applications and requirements by providing generic methods for transmitting and compressing XML documents.

Part 1 – Binary Format for XML: specifies the tools for preparing XML documents for efficient transport and storage and for compressing XML documents.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

The ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from the companies listed in Annex C.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights other than those identified in Annex C. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — MPEG systems technologies —

Part 1: Binary MPEG format for XML

1 Scope

This part of ISO/IEC 23001 provides a standardized set of technologies for encoding XML documents. It addresses a broad spectrum of applications and requirements by providing a generic method for transmitting and compressing XML documents.

This part of ISO/IEC 23001 specifies system level functionalities for the communication of XML documents. It provides a specification which will:

— enable the development of ISO/IEC 23001-1 receiving sub-systems, called ISO/IEC 23001-1 Terminal, or Terminal in short, to receive and assemble possibly partitioned and compressed XML documents

— provide rules for the preparation of XML documents for efficient transport and storage.

The decoding process within the ISO/IEC 23001-1 Terminal is normative. The rules mentioned provide guidance for the preparation and encoding of XML documents without leading to a unique encoded representation of such documents.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

- ISO/IEC 10646:2003, *Information technology — Universal Multiple-Octet Coded Character Set (UCS)*

Note: The UTF-8 encoding scheme is described in Annex R of ISO/IEC 10646-1:1993, published as Amendment 2 of ISO/IEC 10646-1:1993.

- XML, *Extensible Markup Language (XML) 1.0,* October 2000.
- *XML Schema, W3C Recommendation*, 2 May 2001.
- *XML Schema Part 0: Primer,* W3C Recommendation, 2 May 2001.
- *XML Schema Part 1: Structures*, W3C Recommendation, 2 May 2001.
- *XML Schema Part 2: Datatypes*, W3C Recommendation 2 May 2001.
- XPath, *XML Path Language,* W3C Recommendation, 16 November 1999.
- *Namespaces in XML*, W3C Recommendation, 14 January 1999.

Note: These documents are maintained by the W3C (http://www.w3.org). The relevant documents can be obtained as follows:

- *Extensible Markup Language (XML) 1.0 (Second Edition),*6 October 2000, http://www.w3.org/TR/2000/REC-xml-20001006
- *XML Schema: W3C Recommendation*, 2 May 2001, http://www.w3.org/XML/Schema
  - *XML Schema Part 0: Primer,* W3C Recommendation, 2 May 2001, http://www.w3.org/TR/xmlschema-0/
  - *XML Schema Part 1: Structures*, W3C Recommendation, 2 May 2001, http://www.w3.org/TR/xmlschema-1/
  - *XML Schema Part 2: Datatypes*, W3C Recommendation 2 May 2001, http://www.w3.org/TR/xmlschema-2/
- *xPath, XML Path Language*, W3C Recommendation, 16 November 1999, http://www.w3.org/TR/1999/REC-xpath-19991116
- *Namespaces in XML*, W3C Recommendation, 14 January 1999, http://www.w3.org/TR/1999/REC-xml-names-19990114

- *RFC 2396, Uniform Resource Identifiers (URI): Generic Syntax.*
- *RFC 1950, ZLIB Compressed Data Format Specification version 3.3.*
- *IEEE Standard for Binary Floating-Point Arithmetic*, Std 754-1985 Reaffirmed1990, http://standards.ieee.org/reading/ieee/std_public/description/busarch/754-1985_desc.html

3 Terms and definitions

3.1 Conventions

3.1.1 Naming convention

In order to specify data types and documents model, this part of ISO/IEC 23001 uses constructs specified in XML Schema, such as “element”, “attribute”, “simpleType” and “complexType”. The names associated with these constructs are created on the basis of the following conventions:

If the name is composed of various words, the first letter of each word is capitalized. The rule for the capitalization of the first word depends on the type of construct and is described below.

— ***Element naming:*** the first letter of the first word is capitalized (e.g. *TimePoint* element of *TimeType*).

— ***Attribute naming:*** the first letter of the first word is **not** capitalized (e.g. *timeUnit* attribute of *IncrDurationType*).

— ***complexType naming:*** the first letter of the first word is capitalized, the suffix “Type” is used at the end of the name.

— ***simpleType naming:*** the first letter of the first word is not capitalized, the suffix “Type” may be used at the end of the name.

3.1.2 Documentation convention

3.1.2.1 Textual syntax

The syntax of each XML schema item is specified using the constructs specified in XML Schema. It is depicted in this document using a specific font and background, as shown in the example below:

```
<complexType name="ExampleType">
   <sequence>
      <element name="Element1" type="string"/>
   </sequence>
   <attribute name="attribute1" type="string" default="attrvalue1"/>
</complexType>
```

Non-normative XML examples are included in separate subclauses. They are depicted in this document using a separate font and background than the normative syntax specifications, as shown in the example below:

```
<Example attribute1="example attribute value">
   <Element1>example element content</Element1>
</Example>
```

3.1.2.2 Binary syntax

3.1.2.2.1 Overview

The binary document stream retrieved by the decoder is specified in Clause 6, Clause 7, and Clause 8. Each data item in the binary document stream is printed in bold type. It is described by its name, its length in bits, and by a mnemonic for its type and order of transmission. The construct "N+" in the length field indicates that the length of the element is an integer multiple of N.

The action caused by a decoded data element in a bitstream depends on the value of the data element and on data elements that have been previously decoded. The following constructs are used to express the conditions when data elements are present:

while (condition) {
data_element
. . .
}
	If the condition is true, then the group of data elements occurs next in the data stream. This repeats until the condition is not true.

do { **data_element** . . . } while (condition)	 The data element always occurs at least once. The data element is repeated until the condition is not true.

if (condition) {
data_element
. . .
} else {
data_element
. . .
}
	If the condition is true, then the first group of data elements occurs next in the data stream.
	
	If the condition is not true, then the second group of data elements occurs next in the data stream.

for (i = m; i < n; i++) {
data_element
. . .
}
	The group of data elements occurs (n-m) times. Conditional constructs within the group of data elements may depend on the value of the loop control variable i, which is set to m for the first occurrence, incremented by one for the second occurrence, and so forth.

/* comment */	Explanatory comment that may be deleted entirely without in any way altering the syntax.

This syntax uses the 'C-code' convention that a variable or expression evaluating to a non-zero value is equivalent to a condition that is true and a variable or expression evaluating to a zero value is equivalent to a condition that is false.

Use of function-like constructs in syntax tables

In some syntax tables, function-like constructs are used in order to pass the value of a certain syntax element or decoding parameter down to a further syntax table. In that table, the syntax part is then defined like a function in e.g. C program language, specifying in brackets the type and name of the passed syntax element or decoding parameter, and the returned syntax element type, as shown in the following example:

datatype Function(datatype parameter_name) {	**Number of bits**	**Mnemonic**
if (parameter_name == ...) {		
OtherFunction(parameter_name)		
} else if		
.....		
} else {		
.....		
}		
Return return_value		
}		

Here, the syntax table describing the syntax part called “Function” receives the parameter “parameter_name” which is of datatype “datatype”. The parameter “parameter_name” is used within this syntax part, and it can also be passed further to other syntax parts, in the table above e.g. to the syntax part “OtherFunction”.

The parsing of the binary syntax is expressed in procedural terms. However, it should not be assumed that Clause 6, 7 and 8 implement a complete decoding procedure. In particular, the binary syntax parsing in this specification assumes a correct and error-free binary document stream. Handling of erroneous binary document streams is left to individual implementations.

Syntax elements and data elements are depicted in this document using a specific font such as the following example: `FragmentUpdatePayload`.

boolean

In some syntax tables, the “true” and “false” constructs are used. If present in the stream “true” shall be represented with a single bit of value “1” and “false” shall be represented with a single bit of value “0”.

3.1.2.2.2 Arrays

Arrays of data elements are represented according to the C-syntax as described below. It should be noted that each index of an array starts with the value “0”.

data_element[n]	is the n+1th element of an array of data.
data_element[m][n]	is the m+1, n+1th element of a two-dimensional array of data**.**
data_element[l][m][n]	is the l+1, m+1, n+1th element of a three-dimensional array of data.

3.1.2.2.3 Functions

3.1.2.2.3.1 nextByteBoundary()

The function "nextByteBoundary()" reads and consumes bits from the binary document stream until but not including the next byte-aligned position in the binary document stream.

3.1.2.2.4 Reserved values and forbidden values

The terms "reserved" and "forbidden" are used in the description of some values of several code and index tables.

The term "reserved" indicates that the value shall not occur in a binary document stream. It may be used in the future for ISO/IEC defined extensions.

The term "forbidden" indicates a value that shall not occur in a binary document stream.

3.1.2.2.5 Reserved bits and stuffing bits

ReservedBits: a binary syntax element whose length is indicated in the syntax table. The value of each bit of this element shall be "1". These bits may be used in the future for ISO/IEC defined extensions.

Stuffing bits: bits inserted to align the binary document stream, for example to a byte boundary. The value of each of these bits in the binary document stream shall be "1".

ReservedBitsZero: a binary syntax element whose length is indicated in the syntax table. The value of each bit of this element shall be "0". These bits may be used in the future for ISO/IEC defined extensions.

3.1.2.3 Textual and binary semantics

The semantics of each schema or binary syntax component, is specified using a table format, where each row contains the name and a definition of that schema or binary syntax component:

Name	*Definition*
ExampleType	Specifies an ...
element1	Describes the …
attribute1	Describes the …

3.2 Definitions

3.2.1
access unit
An entity within an XML document that is atomic in time, i.e., to which a composition time can be attached. An access unit is composed of one or more fragment update units.

3.2.2
additional schema
A schema that can be updated after the start of the decoding process.

3.2.3
advanced optimised decoder
An optimised decoder used to decode a simple type. Advanced optimised decoders parameters and their mappings to types can be modified during binary document stream lifetime.

3.2.4
advanced optimised decoder instance
An advanced optimised decoder initialised and ready to be used for the decoding of some data types.

Note - There can be several instances of the same advanced optimised decoder with different or identical parameters.

3.2.5
advanced optimised decoder instances table
A table of all the advanced optimised decoders available at a certain instant in time.

3.2.6
advanced optimised decoder parameters
The parameters of an advanced optimised decoder.

3.2.7
advanced optimised decoder type
The type, identified by a URI, of an advanced optimised decoder.

3.2.8
application
An abstraction of any entity that makes use of the decoded document stream.

3.2.9
binary access unit
An access unit in binary format as specified in Clause 6 and 7.

3.2.10
binary document stream
A concatenation of binary access units as specified in Clause 6 and 7.

3.2.11
binary format document tree
The internal binary decoder model.

3.2.12
byte-aligned
A bit in a binary document stream is byte-aligned if its position is a multiple of 8-bits from the first bit in the binary document stream.

3.2.13
composition time
The point in time when a specific access unit becomes known to the application.

3.2.14
content particle
A particle is a term in the XML Schema grammar for element content, consisting of an element declaration, a wildcard or a model group, together with occurrence constraints. Refers to XML SCHEMA.

3.2.15
context mode
Information in the fragment update context specifying how to interpret the subsequent context path information.

3.2.16
context node
The context node is specified by the context path of the current fragment update context. It is the parent of the operand node.

3.2.17
context path
Information that identifies and locates the context node and the operand node in the current document tree.

3.2.18
contextual optimised decoder
“An optimised decoder which behavior is dependent on the current context of the decoding.

Note - For instance, the ZLib optimised decoder (see Clause 8) is a contextual optimised decoder.

Note - Upon certain events, the context must be reset. Upon a certain command or events they are flushed to release their contents. Only contextual optimised decoders are flushable.”

3.2.19
contextual optimised decoder reset
An operation that resets the optimised decoder to put it in a defined initial state. All contextual information is discarded.

3.2.20
current context node
The starting node for the context path in case of relative addressing.

3.2.21
current document
The document that is conveyed by the initial document and all access units up to a given composition time.

3.2.22
current document tree
The XML document tree that represents the current document.

3.2.23
deferred fragment reference
A fragment reference that can be resolved at any time by the application using the terminal.

3.2.24
deferred node
A node which is present in the document tree at encoder side and for which the following is true: No part of that node has been sent to the decoder but the existence of that node has been signalled to the decoder.

3.2.25
delivery layer
An abstraction of any underlying transport or storage functionality.

3.2.26
derived type
A type defined by the derivation of an other type.

3.2.28
document
Short term for a structured XML document.

3.2.29
document composer
An entity that reconstitutes the current document tree from the fragment update units.

3.2.30
document fragment
A contiguous part of a document attached at a single node. Using the representation model of a document tree, the document fragment is represented by a sub-tree of the document tree.

3.2.31
document fragment reference
A reference to a document fragment.

Note - For instance, a fragment reference can be a URI which serves to locate the fragment on the world wide web.

3.2.32
document stream
The ordered concatenation of either binary or textual access units conveying a single, possibly time-variant, document.

3.2.33
document tree
A model that is used throughout this specification in order to represent documents. A document tree consists of nodes, which represent elements or attributes of an XML document. Each node may have zero, one or more child nodes. Simple content are considered as child nodes in Clause 6 of the specification.

3.2.34
effective content particle
The particle of a complexType used for the validation process.

3.2.35
fixed optimised decoder
An optimised decoder used to decode either a complex type or a simple type. Fixed optimised decoders are set up at decoder initialisation phase and their mapping to types can't be modified during binary document stream lifetime.

3.2.36
fragment reference
short term for document fragment reference.

3.2.37
fragment reference format
An encoding format of fragment references.

3.2.38
fragment reference marker
A specific information used to describe a deferred fragment reference, which is present within the current document tree. It consists of a fragment reference, the name and type of the top most element of the referenced fragment.

3.2.39
fragment reference resolver
An entity that is capable of resolving the fragment reference provided in the fragment update payload.

3.2.40
fragment update command
A command within a fragment update unit expressing the type of modification to be applied to the part of the current document tree that is identified by the associated fragment update context.

3.2.41
fragment update component extractor
An entity that de-multiplexes a fragment update unit, resulting in the unit's components: fragment update command, fragment update context, and fragment update payload.

3.2.42
fragment update context
Information in a fragment update unit that specifies on which node in the current document tree the fragment update command shall be executed. Additionally, the fragment update context specifies the data type of the element encoded in the subsequent fragment update payload.

3.2.43
fragment update decoder parameters
Configuration parameters conveyed in the DecoderInit (see 6.2) that are required to specify the decoding process of the fragment update decoder.

3.2.44
fragment update payload
Information in a fragment update unit that conveys the information which is added to the current document or which replaces a part of the current document.

3.2.45
fragment update payload decoder
The entity that decodes the fragment update payload information of the fragment update.

3.2.46
fragment update unit
Information in an access unit, conveying a document or a portion thereof. Fragment update units provide the means to modify the current document. They are nominally composed of a fragment update command, a fragment update context and a fragment update payload.

3.2.47
initial document
A document that initialises the current document tree without conveying it to the application (see 5.3). The initial document is part of the DecoderInit (see 6.2).

3.2.48
initial schema
The schema that is known by the decoder before the decoding process starts.

3.2.49
initialisation extractor
An entity that de-multiplexes the DecoderInit (see 6.2), resulting in its components initial document, fragment update decoder parameters and schema URI.

3.2.51
non-deferred fragment reference
A fragment reference that shall be resolved by the terminal at the composition time of the access unit containing the fragment reference.

3.2.52
operand node
The node in the binary format document tree that is either added, deleted or replaced according to the current fragment update command and fragment update payload. The operand node is always a child node of the context node.