
**Information technology — Multimedia
content description interface —**

**Part 1:
Systems**

*Technologies de l'information — Interface de description du contenu
multimédia —*

Partie 1: Systèmes

Reference number
ISO/IEC 15938-1:2002(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/IEC 2002

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

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.ch
Web www.iso.ch

Printed in Switzerland

Contents

Page

Foreword.....	iv
Introduction	v
1 Scope	1
2 Normative references	2
3 Terms and definitions	3
3.1 Conventions	3
3.2 Definitions	6
4 Symbols and abbreviated terms	10
4.1 Abbreviations	10
4.2 Mathematical operators	10
4.3 Mnemonics	12
5 System architecture	14
5.1 Terminal architecture	14
5.2 General characteristics of the decoder	14
5.3 Sequence of events during decoder initialisation	15
5.4 Decoder behaviour	16
5.5 Issues in encoding descriptions	16
5.6 Differences between the TeM and BiM	18
5.7 Characteristics of the delivery layer	19
6 Textual Format - TeM	21
6.1 Overview	21
6.2 Textual DecoderInit	21
6.3 Textual Access Unit	22
6.4 Textual Fragment Update Unit	23
6.5 Textual Fragment Update Command	23
6.6 Textual Fragment Update Context	24
6.7 Textual Fragment Update Payload	25
7 Binary format - BiM	27
7.1 Overview	27
7.2 Binary DecoderInit	27
7.3 Binary Access Unit	30
7.4 Binary Fragment Update Unit	30
7.5 Binary Fragment Update Command	32
7.6 Binary Fragment Update Context	33
8 Binary Fragment Update Payload	49
8.1 Overview	49
8.2 Definitions	49
8.3 Fragment Update Payload syntax and semantics	50
8.4 Element syntax and semantics	51
8.5 Element Content decoding process	60
Annex A (informative) Informative examples	78
Annex B (informative) FUContextType definition based on W3C XPath specification	81
Annex C (informative) Patent statements	83
Bibliography	85

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 3.

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 15938-1 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 15938 consists of the following parts, under the general title *Information technology — Multimedia content description interface*:

- *Part 1: Systems*
- *Part 2: Description definition language*
- *Part 3: Visual*
- *Part 4: Audio*
- *Part 5: Multimedia description schemes*
- *Part 6: Reference software*
- *Part 7: Conformance testing*
- *Part 8: Extraction and use of MPEG-7 descriptions*

Annexes A to C of this part of ISO/IEC 15938 are for information only.

Introduction

This standard, also known as "Multimedia Content Description Interface," provides a standardized set of technologies for describing multimedia content. The standard addresses a broad spectrum of multimedia applications and requirements by providing a metadata system for describing the features of multimedia content.

The following are specified in this standard:

- **Description Schemes (DS)** describe entities or relationships pertaining to multimedia content. Description Schemes specify the structure and semantics of their components, which may be Description Schemes, Descriptors, or datatypes.
- **Descriptors (D)** describe features, attributes, or groups of attributes of multimedia content.
- **Datatypes** are the basic reusable datatypes employed by Description Schemes and Descriptors.
- **Description Definition Language (DDL)** defines Description Schemes, Descriptors, and Datatypes by specifying their syntax, and allows their extension.
- **Systems tools** support delivery of descriptions, multiplexing of descriptions with multimedia content, synchronization, file format, and so forth.

This standard is subdivided into eight parts:

Part 1 – Systems: specifies the tools for preparing descriptions for efficient transport and storage, compressing descriptions, and allowing synchronization between content and descriptions.

Part 2 – Description definition language: specifies the language for defining the standard set of description tools (DSs, Ds, and datatypes) and for defining new description tools.

Part 3 – Visual: specifies the description tools pertaining to visual content.

Part 4 – Audio: specifies the description tools pertaining to audio content.

Part 5 – Multimedia description schemes: specifies the generic description tools pertaining to multimedia including audio and visual content.

Part 6 – Reference software: provides a software implementation of the standard.

Part 7 – Conformance testing: specifies the guidelines and procedures for testing conformance of implementations of the standard.

Part 8 – Extraction and use of MPEG-7 descriptions: provides guidelines and examples of the extraction and use of descriptions.

.....

Information technology — Multimedia content description interface —

Part 1: Systems

1 Scope

This International Standard defines a Multimedia Content Description Interface, specifying a series of interfaces from system to application level to allow disparate systems to interchange information about multimedia content. It describes the architecture for systems, a language for extensions and specific applications, description tools in the audio and visual domains, as well as tools that are not specific to audio-visual domains.

This part of ISO/IEC 15938 specifies system level functionalities for the communication of multimedia content descriptions. ISO/IEC 15938-1 provides a specification which will:

- enable development of ISO/IEC 15938 receiving sub-systems, called ISO/IEC 15938 Terminal, or Terminal in short, to receive and assemble possibly partitioned and compressed multimedia content descriptions
- provide rules for the preparation of multimedia content descriptions consisting of the tools specified in Parts 3, 4 and 5 of ISO/IEC 15938 for efficient transport and storage.

The decoding process within the ISO/IEC 15938 Terminal is normative. The rules mentioned provide guidance for the preparation and encoding of multimedia content descriptions without leading to a unique encoded representation of such descriptions.

This part of the MPEG-7 Standard is intended to be implemented in conjunction with other parts of the standard. In particular, MPEG-7 Part 1: Systems assumes some knowledge of Part 2: Description Definition Language (DDL) in its normative syntactic definitions of Descriptors and Description Schemes, as well as in the processing of schema and descriptions. The methods for obtaining the descriptions to which the encoding techniques in this part refer are defined in Parts 3, 4, and 5 of ISO/IEC 15938.

MPEG-7 is an extensible standard. The standard method of extending the standard beyond the Description Schemes provided in the standard is to define new ones in the DDL, and to make those DSs as accessible as the instantiated descriptions. Further details are available in Part 2.

2 Normative references

The following normative documents contain provisions which, through reference in this text, constitute provisions of this part of ISO/IEC 15938. For dated references, subsequent amendments to, or revisions of, any of these publications do not apply. However, parties to agreements based on this part of ISO/IEC 15938 are encouraged to investigate the possibility of applying the most recent editions of the normative documents indicated below. For undated references, the latest edition of the normative document referred to applies. Members of ISO and IEC maintain registers of currently valid International Standards. The Telecommunication Standardization Bureau maintains a list of currently valid ITU-T Recommendations.

- ISO/IEC 10646-1:2000, *Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane*

NOTE The UTF-8 encoding scheme is described in Annex D of ISO/IEC 10646-1:2000.

- XML, *Extensible Markup Language (XML) 1.0*, 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>>

NOTE These documents are maintained by the W3C (<http://www.w3.org>).

- RFC 2396, *Uniform Resource Identifiers (URI): Generic Syntax*.
- IEEE Standard for Binary Floating-Point Arithmetic, Std 754-1985 Reaffirmed 1990,
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, Descriptors and Description Schemes, this part of ISO/IEC 15938 uses constructs specified in ISO/IEC 15938-2, 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 ISO/IEC 15938-2. 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 description stream retrieved by the decoder is specified in Clause 7 and Clause 8. Each data item in the binary description 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 7 and 8 implement a complete decoding procedure. In particular, the binary syntax parsing in this specification assumes a correct and error-free binary description stream. Handling of erroneous binary description 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 description stream until but not including the next byte-aligned position in the binary description 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 description 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 description 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 description stream, for example to a byte boundary. The value of each of these bits in the binary description stream shall be "1".

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:

<i>Name</i>	<i>Definition</i>
ExampleType	Specifies an ...
element1	Describes the ...
attribute1	Describes the ...

3.2 Definitions

3.2.1

access unit

An entity within a description stream 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

application

An abstraction of any entity that makes use of the decoded description stream.

3.2.3

binary access unit

An access unit in binary format as specified in Clause 7 and 8.

3.2.4

binary description stream

A concatenation of binary access units as specified in Clause 7 and 8.

3.2.5

binary format description tree

The internal binary decoder model.

3.2.6

byte-aligned

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

3.2.7

composition time

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

3.2.8

content particle

A particle is a term in the XML Schema grammar for element content, consisting of either an element declaration, a wildcard or a model group, together with occurrence constraints. Refers to ISO/IEC 15938-2.

3.2.9

context mode

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

3.2.10

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.

3.2.11**context path**

Information that identifies and locates the context node and the operand node in the current description tree.

3.2.12**current context node**

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

3.2.13**current description**

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

3.2.14**current description tree**

The description tree that represents the current description.

3.2.15**DDL parser**

An application that is capable of validating description schemes (content and structure) and descriptor data types against their schema definition.

3.2.16**delivery layer**

An abstraction of any underlying transport or storage functionality.

3.2.17**derived type**

A type defined by the derivation of an other type.

3.2.18**described time**

A point in time or range of time, embedded in the description, that is related to the media described by the description. Note that there is no intrinsic relation between the described time and the composition time of an access unit. This information is e.g. carried by instances of the *MediaTimeType* defined in ISO/IEC 15938-5.

3.2.19**description**

Short term for multimedia content description.

3.2.20**description composer**

An entity that reconstitutes the current description tree from the fragment update units.

3.2.21**description fragment**

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

3.2.22**description stream**

The ordered concatenation of either binary or textual access units conveying a single, possibly time-variant, multimedia content description.

3.2.23**description tree**

A model that is used throughout this specification in order to represent descriptions. A description tree consists of nodes, which represent elements or attributes of a description. Each node may have zero, one or more child nodes. Simple content are considered as child nodes in Clause 7 of the specification.

3.2.24

effective content particle

The particle of a complexType used for the validation process.

3.2.25

fragment update command

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

3.2.26

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.

3.2.27

fragment update context

Information in a fragment update unit that specifies on which node in the current description 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.

3.2.28

fragment update payload

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

3.2.29

fragment update payload decoder

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

3.2.30

fragment update unit

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

3.2.31

fragment update decoder parameters

Configuration parameters conveyed in the `DecoderInit` (see 6.2 and 7.2) that are required to specify the decoding process of the fragment update decoder.

3.2.32

initial description

A description that initialises the current description tree without conveying it to the application (see 5.3). The initial description is part of the `DecoderInit` (see 6.2 and 7.2).

3.2.33

initialisation extractor

An entity that de-multiplexes the `DecoderInit` (see 6.2 and 7.2), resulting in its components initial description, fragment update decoder parameters and schema URI.

3.2.34

multimedia content description

The description of audiovisual data content in multimedia environments using the tools and elements provided by the parts 2, 3, 4 and 5 of ISO/IEC 15938.

3.2.35**operand node**

The node in the binary format description 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.

3.2.36**schema**

A schema is represented in XML by one or more "schema documents", that is, one or more "<schema>" element information items. A "schema document" contains representations for a collection of schema components, e.g. type definitions and element declarations, which have a common target namespace. A schema document which has one or more "<import>" element information items corresponds to a schema with components with more than one target namespace. Refer also to ISO/IEC 15938-2.

3.2.37**schema resolver**

An entity that is capable of resolving the schema identification provided in the `DecoderInit` (see 6.2 and 7.2), and to possibly retrieve the specified schemas.

3.2.38**schema URI**

A URI that uniquely identifies a schema.

3.2.39**schema valid**

A description that is *schema valid* satisfies the constraints embodied in the Schema to which it should conform.

3.2.40**selector node**

The parent node of the topmost node of a description tree. It artificially extends the description tree to allow the addressing of the topmost node.

3.2.41**super type**

The parent of a type in its type hierarchy.

3.2.42**systems layer**

An abstraction of the tools and processes specified by this part of ISO/IEC 15938.

3.2.43**terminal**

The entity that makes use of a coded representation of a multimedia content description.

3.2.44**textual access unit**

An access unit in textual format as specified in Clause 6.

3.2.45**textual description stream**

A concatenation of textual access units as specified in Clause 6.

3.2.46**topmost node**

The node specified by the first element in the description, instantiating one of the global elements declared in the schema.

3.2.47

type hierarchy

The hierarchy of type derivations.

3.3.48

validation

The process of parsing an XML document to determine whether it satisfies the constraints embodied in the Schema to which it should conform.

4 Symbols and abbreviated terms

4.1 Abbreviations

AU	Access Unit
BiM	Binary format for multimedia description streams
D	Descriptor
DDL	Description Definition Language
DL	Delivery Layer
DS	Description Scheme
FU	Fragment Update
FUU	Fragment Update Unit
FSAD	Finite State Automaton Decoder
MPC	Multiple element Position Code
SBC	Schema Branch Code
SPC	Single element Position Code
TBC	Tree Branch Code
TeM	Textual format for multimedia description streams
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	Universal Character Set Transformation Formats
XML	Extensible Markup Language
XPath	XML Path Language

4.2 Mathematical operators

The mathematical operators used to describe this part of ISO/IEC 15938 are similar to those used in the C programming language. However, integer divisions with truncation and rounding are specifically defined. Numbering and counting loops generally begin from zero.

4.2.1 Arithmetic operators

+	Addition.
-	Subtraction (as a binary operator) or negation (as a unary operator).
++	Increment. i.e. $x++$ is equivalent to $x = x + 1$
--	Decrement. i.e. $x--$ is equivalent to $x = x - 1$

* Multiplication.

^ Power.

sign() $sign(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases}$

abs() $abs(x) = x \cdot sign(x)$

log2(..) $\log 2(x) = \log_2(x)$

ceil(..) $ceil(x) = \begin{cases} int(x) + 1 & x \geq 0 \\ int(x) & x < 0 \end{cases}$

int(..) truncation of the argument to its integer value, e.g. 1.3 is truncated to 1 and –3.7 is truncated to –3.

$\sum_{i=a}^{i=b} f(i)$ the summation of the f(i) with i taking integral values from a up to, but not including b.

4.2.2 Logical operators

|| Logical OR.

&& Logical AND.

! Logical NOT.

4.2.3 Relational operators

> Greater than.

>= Greater than or equal to.

< Less than.

<= Less than or equal to.

== Equal to.

!= Not equal to.

max (, ...,) the maximum value in the argument list.

min (, ... ,) the minimum value in the argument list.

4.2.4 Assignment

= Assignment operator.

4.2.5 Character string comparison

Many phases of the fragment encoding rely on a string comparison method. This method is based on the Unicode value of each character in the strings. The following defines the notion of lexicographic ordering:

Two strings are different if they have different characters at some index that is a valid index for both strings, or if their lengths are different, or both.

If they have different characters at one or more index positions, let *k* be the smallest such index; then the string whose character at position *k* has the smaller value, as determined by using the < operator, lexicographically precedes the other string.

If there is no index position at which they differ, then the shorter string lexicographically precedes the longer string.

This string comparison is described by each method that is functionally equivalent to the following procedure:

```
compare_strings(string1, string2) {  
  
    len1 = length(string1);  
  
    len2 = length(string2);  
  
    n = min(len1, len2);  
  
    i = 0;  
  
    j = 0;  
  
    while (n-- != 0) {  
  
        c1 = string1[i++];  
  
        c2 = string2[j++];  
  
        if (c1 != c2) {  
  
            return c1 - c2;  
  
        }  
  
    }  
  
    return len1 - len2;  
  
}
```

4.3 Mnemonics

The following mnemonics are defined to describe the different data types used in the coded bitstream.

Name	Definition
bslbf	Bit string, left bit first, where "left" is the order in which bit strings are written in this part of ISO/IEC 15938. Bit strings are generally written as a string of 1s and 0s within single quote marks, e.g. '1000 0001'. Blanks within a bit string are for ease of reading and have no significance. For convenience large strings are occasionally written in hexadecimal, in this case conversion to a binary in the conventional manner will yield the value of the bit string. Thus the left most hexadecimal digit is first and in each hexadecimal digit the most significant of the four bits is first.

uimbsf	Unsigned integer, most significant bit first.
vlclbf	Variable length code, left bit first, where "left" refers to the order in which the VLC codes are written. The byte order of multibyte words is most significant byte first.
vluimbsf8	Variable length code unsigned integer, most significant bit first. The size of vluimbsf8 is a multiple of one byte. The first bit (Ext) of each byte specifies if set to 1 that another byte is present for this vluimbsf8 code word. The unsigned integer is encoded by the concatenation of the seven least significant bits of each byte belonging to this vluimbsf8 code word An example for this type is shown in Figure 1.
vluimbsf5	Variable length code unsigned integer, most significant bit first. The first n bits (Ext) which are 1 except of the n-th bit which is 0, indicate that the integer is encoded by n times 4 bits. An example for this type is shown in Figure 2.

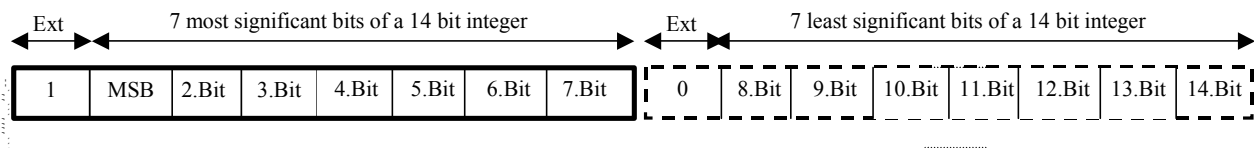


Figure 1 — Informative example for the vluimbsf8 data type

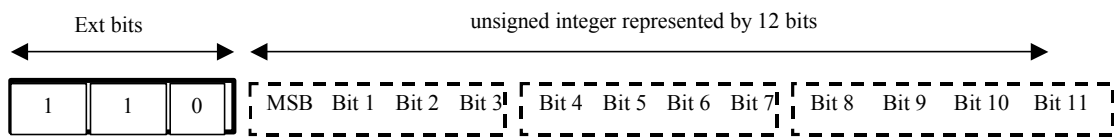


Figure 2 — Informative example for the vluimbsf5 data type

5 System architecture

5.1 Terminal architecture

ISO/IEC 15938 provides the means to represent coded multimedia content descriptions. The entity that makes use of such coded representations of the multimedia content description is generically referred to as the “ISO/IEC 15938 terminal” or just “terminal” in short. This terminal may correspond to a standalone application or be part of an application system.

This and the following three subclauses provide the description of an ISO/IEC 15938 terminal, its components, and their operation. The architecture of such a terminal is depicted in Figure 3. The following subclauses introduce the tools specified in this part of the specification.

In Figure 3, there are three main layers outlined: the application, the normative systems layer, and the delivery layer. ISO/IEC 15938-1 is not concerned with any storage and/or transmission media (whose behaviours and characteristics are abstracted by the delivery layer) or the way the application processes the current description. This specification does make specific assumptions about the delivery layer, and those assumptions are outlined in subclause 5.5.4. The systems layer, which is the subject of this part of ISO/IEC 15938, defines a decoder whose architecture is described here to provide an overview and to establish common terms of reference. A compliant decoder need not implement the constituent parts as visualised in Figure 3, but shall implement the normative decoding process specified in Clauses 6 through 8.

5.2 General characteristics of the decoder

5.2.1 General characteristics of description streams

An ISO/IEC 15938 terminal consumes description streams and outputs a – potentially dynamic – representation of the description called the current description tree. Description streams shall consist of a sequence of one or more individually accessible portions of data named access units. An Access Unit (AU) is the smallest data entity to which “terminal-oriented” (as opposed to “described-media oriented”) timing information can be attributed. This timing information is called the “composition” time, meaning the point in time when the resulting current description tree corresponding to a specific access unit becomes known to the application. The timing information shall be carried by the delivery layer (see subclause 5.5.4). The current description tree shall be schema-valid after processing each access unit.

A description consisting of textual access units is termed a textual description stream and is processed by a textual decoder (see subclause 5.2.2 and clause 6). A description stream consisting of binary access units is termed a binary description stream and is processed by a binary decoder (see subclause 5.2.3 and Clauses 7 and 8). A mixture of both formats in a single stream is not permitted. The choice of either binary or textual format for the description stream is application dependent. Any valid ISO/IEC 15938 description, with the exception of those listed in 5.6.4, may be conveyed in either format.

5.2.2 Principles of the textual decoder (informative)

The ISO/IEC 15938-1 method for textual encoding, called TeM, enables the dynamic and/or progressive transmission of descriptions using only text. The original description, in the form of an XML document, is partitioned into fragments (see 5.5.1) that are wrapped in further XML code so that these resulting AUs can be individually transported (e.g. streamed or sent progressively). The decoding process for these AUs does not require any schema knowledge. The resulting current description tree may be byte-equivalent to the original description if desired by the encoder, but it may also exhibit dynamic characteristics such that certain parts of the description are present at the decoder only at chosen times, are never present at all, or appear in a different part of the tree.

5.2.3 Principles of the binary decoder (informative)

Using the ISO/IEC 15938-1 generic method for binary encoding, called BiM, a description (nominally in a textual XML form) can be compressed, partitioned, streamed, and reconstructed at terminal side. The reconstructed XML description will not be byte-equivalent to the original description. Namely, the binary encoding method does not

preserve processing instructions, attribute order, comments, or non-significant whitespace. However, the encoding process ensures that XML element order is preserved.

The BiM, in order to gain its compression efficiency, relies on a schema analysis phase. During this phase, internal tables are computed to associate binary code to XML elements, types and attributes. This principle mandates the full knowledge of the same schema by the decoder and the encoder for maximum interoperability.

As with the textual decoder, the resulting current description tree may be topologically equivalent to the original description if desired by the encoder, but it may also exhibit dynamic characteristics such that certain parts of the description are present at the decoder only at chosen times, are never present at all, or appear in a different part of the tree.

5.3 Sequence of events during decoder initialisation

The decoder set-up is signalled by the initialisation extractor receiving a textual or binary `DecoderInit` (specified in 6.2 and 7.2). The signalling of the use of either binary or textual encoding is outside the scope of this specification. However, if the `DecoderInit` is binary, then the following description stream shall consist of binary access units. Similarly, if the `DecoderInit` is textual, then the following description stream shall consist of textual access units. The `DecoderInit` shall be received by the systems layer from the delivery layer. The `DecoderInit` will typically be conveyed by a separate delivery channel compared to the description stream, which is also received from the delivery layer. The component parts of the description stream are discussed in subclause 5.4.

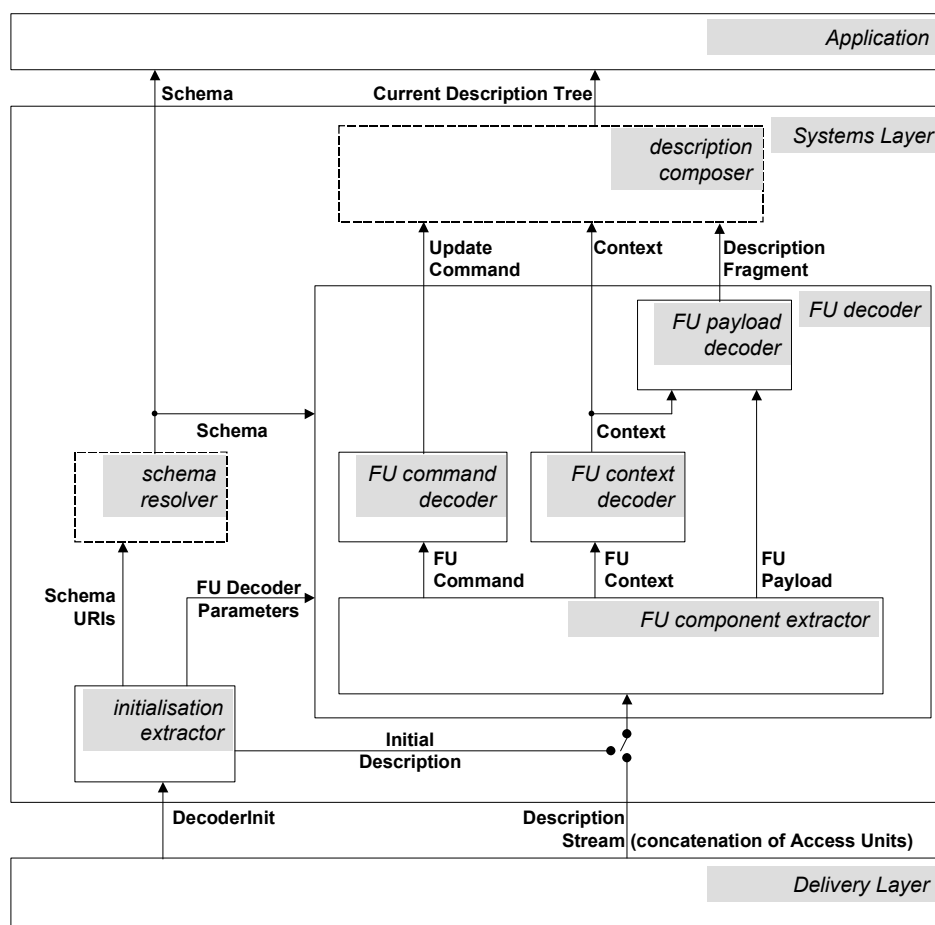


Figure 3 — Terminal Architecture

Dashed boxes in the systems layer are non-normative. FU is an abbreviation for Fragment Update.

The `DecoderInit` contains a list of URIs that identifies schemas, miscellaneous parameters to configure the decoder (FU Decoder Parameters, in Figure 3), and an initial description. There shall be only one `DecoderInit` per description stream. The list of URIs (Schema URIs, in Figure 3) is passed to a schema resolver that associates the URIs with schemas to be passed into the fragment update decoder (see subclauses 6.2 and 7.2). The schema resolver is non-normative and may, for example, retrieve schema documents from a network or refer to pre-stored schemas. The resulting schemas are used by the binary decoder specified in Clauses 7 and 8 and by any textual DDL parser that may be used for schema validation. If a given Schema URI is unknown to the schema resolver, the corresponding data types in a description stream shall be ignored (i.e., “skipped” and not processed).

The initial description has the same general syntax and semantics as an access unit, but with restrictions, as described in subclauses 6.2 and 7.2. The initial description initialises the current description tree without conveying it to the application. The current description tree is then updated by the access units that comprise the description stream. The initial description may be empty, since a schema-valid current description tree for consumption by the application need only be generated after the first access unit is decoded.

5.4 Decoder behaviour

The description stream shall be processed only after the decoder is initialised. The behaviour of the decoder when access units are received before the decoder is initialised is non-normative. Specifically, there is no requirement to buffer such “early AUs.”

An access unit is composed of any number of fragment update units, each of which is extracted in sequence by the fragment update component extractor. Each fragment update unit consists of:

- a fragment update command that specifies the type of update to be executed (i.e., add, replace or delete content or a node, or reset the current description tree);
- a fragment update context that identifies the data type in a given schema document, and points to the location in the current description tree where the fragment update command applies; and
- a fragment update payload conveying the coded description fragment to be added or replaced.

A fragment update extractor splits the fragment update units from the access units and emits the above component parts to the rest of the decoder. The fragment update command decoder generally consists of a simple table lookup for the update command to be passed on to the description composer. The decoded fragment update context information (‘context’ in Figure 3) is passed along to both the description composer and the fragment update payload decoder. The fragment update payload decoder embodies the BiM Payload decoder (Clause 8) or, in the case of the TeM, a DDL parser, which decodes a fragment update payload (aided by context information) to yield a description fragment (see Figure 3).

The corresponding update command and context are processed by the non-normative description composer, which either places the description fragment received from the fragment update payload decoder at the appropriate node of the current description tree at composition time, or sends a reconstruction event containing this information to the application. The actual reconstruction of the current description tree by the description composer is implementation-specific, i.e., the application may direct the description composer to prune or ignore unwanted elements as desired. There is no requirement on the format of this current description tree, e.g. it may remain a binary representation.

5.5 Issues in encoding descriptions

5.5.1 Fragmenting descriptions

A description stream serves to convey a multimedia content description, as available from a (non-normative) sender or encoder, to the receiving terminal, possibly by incremental transmission in multiple access units. Any number of decompositions of the source description may be possible and it is out of scope of this specification to define such decompositions. Figure 4 illustrates an example of a description, consisting of a number of nodes, that is broken into two description fragments.

If multiple description fragments corresponding to a specific node of the description are sent (e.g., a node is replaced) then the previous data within the nodes of the description represented by that description fragment become unavailable to the terminal. Replacing a single node of the description shall effectively overwrite all children of that node.

NOTE If an application wishes to retain such updated node information, it may do so. However, access to such outdated portions of the description is outside the scope of this specification.

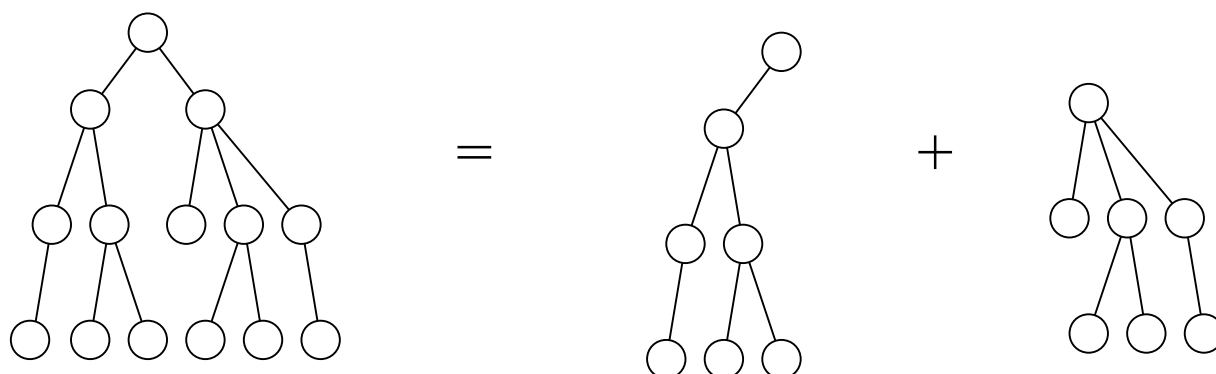


Figure 4 — Decomposition of a description into two description fragments

5.5.2 Deferred nodes and their use

With both the TeM and the BiM, there exists the possibility for the encoder to indicate that a node in the current description tree is “Deferred.” A deferred node shall not contain content, but shall have a type associated with it. A deferred node is addressable on the current description tree (there is a fragment update context that unambiguously points to it), but it shall not be passed on to any further processing steps, such as a parser or an application. In other words, a deferred node is a placeholder that is rendered “invisible” to subsequent processing steps.

The typical use of deferred nodes by the encoder is to establish a desired tree topology without sending all nodes of the tree. Nodes to be sent later are marked as “deferred” and are therefore hidden from a parser. Hence, the current description tree minus any deferred nodes must be schema-valid at the end of each access unit. The deferred nodes may then be replaced in any subsequent access unit without changing the tree topology maintained internally in the decoder. However, there is no guarantee that a deferred node will ever be filled by a subsequent fragment update unit within the description stream.

5.5.3 Managing schema version compatibility with ISO/IEC 15938-1

It is very conceivable that a given schema will be updated during its lifetime. Therefore, ISO/IEC 15938-1 provides, with some constraints, interoperability between different versions of ISO/IEC 15938 schema definitions, without the full knowledge of all schema versions being required.

Two different forms of compatibility between different versions of schema are distinguished. In both cases, it is assumed that the updated version of a schema imports the previous version of that schema. Backward compatibility means that a decoder aware of an updated version of a schema is able to decode a description conformant to a previous version of that schema. Forward compatibility means that a decoder only aware of a previous version of a schema is able to partially decode a description conformant to an updated version of that schema.

With both the textual and binary format, backward compatibility is provided by the unique reference of the used schema in the `DecoderInit` using its Schema URI as its namespace identifier.

When using the binary format, forward compatibility is ensured by a specific syntax defined in Clause 7 and 8. Its main principle is to use the namespace of the schema, i.e., the Schema URI, as a unique version identifier. The binary format allows one to keep parts of a description related to different schema in separate chunks of the binary description stream, so that parts related to unknown schema may be skipped by the decoder. In order for this

approach to work, an updated schema should not be defined using the ISO/IEC 15938-2 “redefine” construct but should be defined in a new namespace. The Decoder Initialisation identifies schema versions with which compatibility is preserved by listing their Schema URIs. A decoder that knows at least one of the Schema URIs will be able to decode at least part of the binary description stream.

5.5.4 Reference consistency (informative)

The standard itself cannot guarantee reference (link) consistency in all cases. In particular, XPath-style references cannot be guaranteed to point to the correct node, especially when the topology of the tree changes in a dynamic or progressive transmission environment. With ID/IDRef, the system itself cannot guarantee that the ID element will be present, but during the validation phase, all such links are checked, and thus their presence falls under the directive that the current description tree must always be schema-valid. URI and HREF links are typically to external documents, and should be understood not to be under control by the referrer (and therefore not guaranteed).

5.6 Differences between the TeM and BiM

5.6.1 Introduction

BiM and TeM are two similar methods to fragment and convey descriptions as a description stream. While both methods allow one to convey arbitrary descriptions conformant to Parts 3 – 5 of this specification, structural differences in the TeM- and BiM-encoded representation of the description as well as in the decoding process exist.

5.6.2 Use of schema knowledge

The TeM does not require schema knowledge to reconstitute descriptions; hence, the context information identifying the operand node on which the fragment update command is applied is generated with reference to the current description tree as available to the decoder before processing the current fragment update. The TeM operates on an instantiation-based model: one begins with a blank slate (a single selector node) and adds instantiated nodes as they are presented to the terminal. Schema knowledge is, of course, necessary for schema validation to be performed.

The BiM relies upon schema knowledge, i.e., the FU decoder implicitly knows about the existence and position of all potential elements as defined by the schema, no matter whether the corresponding elements have actually been received in the instantiated description. This shared knowledge between encoder and decoder improves compression of the context information and makes the context information independent from the current description tree as available to the decoder. The BiM operates on a schema-based model: all possibilities defined by the schema can be unambiguously addressed using the context information, and as a payload is added, the instantiation of the addressed node is noted. The current description tree is built by the set of all of the instantiated nodes. One non-obvious consequence of this BiM model is that numbering in the internal binary decoder model is “sticky”: once an element is instantiated and thus assigned an address in the internal binary decoder model by its context, the address is unaffected by operations on any other nodes.

5.6.3 Update command semantics

The commands in TeM and BiM are named differently to reflect the fact that the commands operate on different models and have different semantics. The TeM commands have the suffix “node” because the TeM operates (nearly) directly on the current description tree, and thus the removal of a node completely removes it from the tree. The BiM commands have the suffix “content” because the addressing on the current description tree is by indirection, through an internal binary decoder model. Removal of an address, from the point of view of the application, removes the node and its content (sub-elements and attributes) from the current description tree, however the addressed node is still present in the internal decoder model (binary format description tree).

In the TeM, the commands are AddNode, ReplaceNode, and DeleteNode. The AddNode is effectively an “append” command, adding an element among the existing children of the target node. Insertion between two already-received, consecutive children of a node is not possible. One must replace a previously deferred node. By performing a DeleteNode on a node on the current description tree, the addressable indices of its siblings change appropriately.

In the BiM, the commands are AddContent, ReplaceContent and DeleteContent. The AddContent conveys the node data for a node whose path within the description tree is predetermined from the schema evaluation as described in 5.6.2. Hence, internally to the BiM decoder, the paths to (or addresses of) non-empty sibling nodes may be non-contiguous, e.g., the second and fourth occurrence of an element may be present. The “hole” in the numbering is not visible in the current description tree generated by the description composer. Hence, if the third occurrence of said element is added (using AddContent) in a subsequent access unit, it appears to any further processing steps as an “inserted” element in the current description tree, while it simply fills the existing “hole” with respect to the internal numbering of the BiM decoder. Similarly, DeleteContent does delete the node data, but does not change the context path to this node. ReplaceContent replaces node data and does not change the context path to this node either.

For both types of decoders, the “Reset” command reverts the description to the initial description in the DecoderInit.

5.6.4 Restrictions on descriptions that may be encoded

The TeM has limited capability to update mixed content models (defined in ISO/IEC 15938-2). Although it allows the replacement of the entire element, or the replacement of child elements, the mixed content itself cannot be addressed or modified.

Wildcards and mixed content models (defined in ISO/IEC 15938-2) are not supported at all by the BiM. Therefore a schema that uses these mechanisms cannot be supported by the binary format.

5.6.5 Navigation

When navigating through a TeM description, at each step the different possible path is given by the element name, an index, and, possibly, a type identifier. The concatenation of that information is expressed (in a reduced form) by XPath.

In BiM, each step down the tree hierarchy is given by a tree branch code (TBC), whose binary coding is derived from the schema. The concatenation of all TBCs constitutes the context path information.

Both mechanisms in TeM and BiM allow for absolute and relative addressing of a node, starting either from the topmost node of the description or a context node known from the previous decoding steps.

5.6.6 Multiple payloads

With the BiM, for compression efficiency, there may be multiple payloads within a single fragment update unit that implicitly operate on subsequent nodes of the same type. This feature does not exist in the TeM.

5.7 Characteristics of the delivery layer

The delivery layer is an abstraction that includes functionalities for the synchronization, framing and multiplexing of description streams with other data streams. Description streams may be delivered independently or together with the described multimedia content. No specific delivery layer is specified or mandated by ISO/IEC 15938.

Provisions for two different modes of delivery are supported by this specification:

- Synchronous delivery – each access unit shall be associated with a unique time that indicates when the description fragment conveyed within this access unit becomes available to the terminal. This point in time is termed “composition time.”
- Asynchronous delivery – the point in time when an access unit is conveyed to the terminal is not known to the producer of this description stream nor is it relevant for the usage of the reconstructed description. The composition time is understood to be “best effort,” and the order of decoding AUs, if prescribed by the producer of the description, shall be preserved. Note, however, that this in no way precludes time related information (“described time”) to be present within the multimedia content description.

A delivery layer (DL) suitable for conveying ISO/IEC 15938 description streams shall have the following properties:

- The DL shall provide a mechanism to communicate a description stream from its producer to the terminal.
- The DL shall provide a mechanism by which at least one entry point to the description stream can be identified. This may correspond to a special case of a random access point, typically at the beginning of the stream.
- For applications requiring random access to description streams, the DL shall provide a suitable random access mechanism.
- The DL shall provide delineation of the access units within the description stream, i.e., AU boundaries shall be preserved end-to-end.
- The DL shall preserve the order of access units on delivery to the terminal, if the producer of the description stream has established such an order.
- The DL shall provide either error-free access units to the terminal or an indication that an error occurred.
- The DL shall provide a means to deliver the `DecoderInit` information (see subclauses 6.2 and 7.2) to the terminal before any access unit decoding occurs and signal the coding format (textual/binary) of said information.
- The DL shall provide signalling of the association of a description stream to one or more media streams.
- In synchronous delivery mode, the DL shall provide time stamping of access units, with the time stamps corresponding to the composition time (see section on synchronous delivery earlier in this subclause) of the respective access unit.
- If an application requires access units to be of equal or restricted lengths, it shall be the responsibility of the DL to provide that functionality transparently to the systems layer.

Companion requirements exist in order to establish the link between the multimedia content description and the described content itself. These requirements, however, may apply to the delivery layer of the description stream or to the delivery layer of the described content streams, depending on the application context:

- The DL for the description stream or the described content shall provide the mapping information between the content references within the description stream and the described streams.
- The DL for the description stream or the described content shall provide the mapping information between the described time and the time of the described content.

6 Textual Format - TeM

6.1 Overview

The following subclauses specify the syntax elements and associated semantics of the textual format for ISO/IEC 15938 descriptions, abbreviated TeM. The textual `DecoderInit` (6.2), the textual `AccessUnit` (6.3), and the textual fragment update unit (6.4), with its constituent parts: the textual fragment update command (6.5), textual fragment update context (6.6) and textual fragment update payload (6.7).

The following schema wrapper shall be applied to the syntax defined in Clause 6.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:mpeg7s="urn:mpeg:mpeg7:systems:2001"
  targetNamespace="urn:mpeg:mpeg7:systems:2001"
  elementFormDefault="qualified"
  attributeFormDefault="unqualified">

  <!-- here clause 6 schema definition -->

</schema>
```

6.2 Textual DecoderInit

6.2.1 Syntax

```
<!-- ##### -->
<!-- Definition of DecoderInit -->
<!-- ##### -->

<complexType name="mpeg7s:DecoderInitType">
  <sequence>
    <element name="SchemaReference" type="mpeg7s:SchemaReferenceType"
      maxOccurs="unbounded"/>
    <element name="InitialDescription" type="mpeg7s:AccessUnitType"
      minOccurs="0"/>
  </sequence>
  <attribute name="systemsProfileLevelIndication" type="decimal"
    use="required"/>
</complexType>

<complexType name="SchemaReferenceType">
  <attribute name="name" type="anyURI" use="required"/>
  <attribute name="locationHint" type="anyURI" use="optional"/>
</complexType>
```

6.2.2 Semantics

The `DecoderInit` is used to initialise configuration parameters required for the decoding of the textual access units.

Name	Definition
SchemaReference	A list of references to the schemas used by the description. Each reference consists of a schema URI and a location hint URI. The URI identifies the namespace associated with the schema, as specified in ISO/IEC 15938-2. The

`locationHint` is a valid and up-to-date URI that is guaranteed to provide open access to the schema. It shall be present in every case except when it is precisely identical to the corresponding URI.

NOTE A terminal need not use the `locationHint` to locate the schema and may access the schema definition associated with the schema URI using other implementation-defined means. In fact, some terminals may not be capable of retrieving or parsing new schemas.

InitialDescription This optional element conveys portions of a description using the same syntax and semantics as an access unit (see 6.3). The following restrictions on `InitialDescription`, compared to a regular fragment update unit apply:

- For all fragment update units within the `InitialDescription`, `FUCommand` shall take the value "addNode".
- The `FUContext` of the first fragment update unit within the `InitialDescription` shall have the value of "/" referring to the instance document's root element.

The `FUPayload(s)` shall contain the instance data for the initial description. This instance data provides an initial state of the current description tree.

Decoding the `InitialDescription` plus any subsequent AU or AUs shall lead, after composition, to a schema-valid current description tree that may be passed to the application.

systemsProfileLevelIndication Used to indicate which Systems profile and level is being used, as defined in Table 1. This table, defined in Clause 7, provides the code associated with each defined Systems profile and level

NOTE That the system layer is not required to output a current description tree after decoding the initial description and, therefore, the decoded instance data need not result in a schema-valid description.

6.3 Textual Access Unit

6.3.1 Syntax

```
<!-- ##### -->
<!-- Definition of AccessUnitType -->
<!-- ##### -->

<complexType name="AccessUnitType">
  <sequence>
    <element name="FragmentUpdateUnit" type="mpeg7s:FragmentUpdateUnitType"
      maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

6.3.2 Semantics

An AU is composed of a sequence of fragment update unit(s). Multiple fragment update units in an access unit are ordered and shall be processed by the terminal such that the result of applying the commands is equivalent to having executed them sequentially by the description composer in the order specified within the access unit. The