INTERNATIONAL STANDARD

ISO/IEC 15938-1

First edition
2002-07-01

AMENDMENT 1
2005-02-15

Information technology — Multimedia content description interface —

Part 1: **Systems**

AMENDMENT 1: Systems extensions

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

Partie 1: Systèmes

AMENDEMENT 1: Extensions des systèmes

Reference number
ISO/IEC 15938-1:2002/Amd.1:2005(E)

© ISO/IEC 2005

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

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 2005

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.org
Web www.iso.org

Published in Switzerland

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

© ISO/IEC 2005 – All rights reserved

Not for Resale

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.

Amendment 1 to ISO/IEC 15938-1:2002 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 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

This document specifies the first Amendment to the Systems (MDS) part of ISO/IEC 15938.

Note: this document preserves the sectioning of ISO/IEC 15938-1 Systems. The text and figures given below are additions and/or modifications to those corresponding sections in ISO/IEC 15938-1. All figures and tables shall be renumbered due to the addition of several figures and tables.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Multimedia content description interface —

Part 1: Systems

AMENDMENT 1: Systems extensions

Add the following reference in Clause 2:

- *RFC 1950, ZLIB Compressed Data Format Specification version 3.3.*

Add the following definition at the end of subclause 3.1.2.2.5:

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.

Add the following definitions to subclause 3.2 (keep alphabetical order):

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

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

schema update unit
Information in an access unit, conveying a schema or a portion thereof. Schema update units provide the means to modify the current decoder schema knowledge.

description fragment reference
A reference to a description fragment.

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

fragment reference
short term for description fragment reference.

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

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

fragment reference format
An encoding format of fragment references.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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

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.

optimised decoder
A decoder associated to a type and dedicated to certain encoding methods better suited than the generic ones.

type codec
Synonym to optimised decoder.

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 description stream lifetime.

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 description stream lifetime.

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.

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

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

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 9) 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.

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

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

skippable subtree
A subtree of an XML document that the decoder is permitted not to decode.

optimised decoder mapping
An association between a type and a set of optimised decoders.

Renumber all definitions in subclause 3.2.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Add the following abbreviations to the table in subclause 4.1:

MSB	Most Significant Bit
SU	Schema Update
SUU	Schema Update Unit

Add the following mnemonic to subclause 4.3:

Name	*Definition*
vlurmsbf5	Variable length code unsigned rational number, most significant bit first. The first n bits (Ext) which are '1' except of the nth bit which is '0', indicate that the rational number R in the interval 0≤R<1 is encoded by n times 4 bits. The ith bit of the n times 4 bits representing the rational number corresponds to a value of 2^-i. Thus the (n+1)st bit of the vlurmsbf5 code word (which corresponds to the MSB of the rational number) represents a value of ½, the (n+2)nd bit of the vlurmsbf5 code word represents a value of ¼., and so forth. An example for this type is shown in Figure AMD1-1. Note - Comparing two rational numbers A and B represented by a vlurmsbf5 code word can be done by comparing bit by bit the rational numbers starting from their respective MSBs. Then the rational number A is bigger if there is a '1' bit at a position at which there is a '0' for B. A is also bigger if there is a '1' bit at a position which is not present for B and when A is longer than B.

In subclause 4.3, add the following figure after Figure 2:

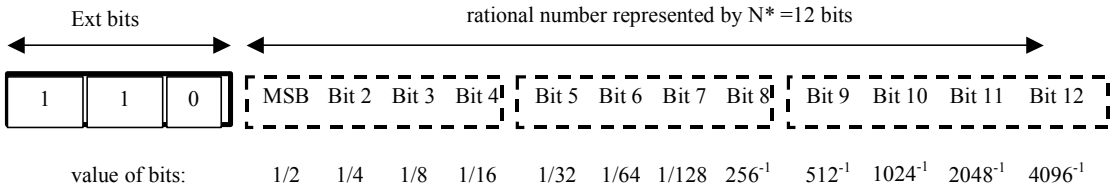


Figure AMD1-1 - Informative example for the vlurmsbf5 data type

Remove the '(informative)' in the title of subclause 5.2.3.

In subclause 5.2.3, replace this paragraph:

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.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

by

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 schema components (XML elements, types and attributes). BiM defines two methods to address schema components.

The first method allows the decoder to resolve a schema, possibly including schema components originating from several namespaces, at initialization phase. This set of schema components form the *initial schema.* In this schema, all type and substitution codes are merged together no matter the namespace they belong to. This results in shorter codes in the binary description stream. The initial schema can't be updated and is considered fixed for the binary description stream lifetime. It contains by default and at minimal the type codes of the xml schema types: anyType, anySimpleType, and all xml schema simple types. In this specification, anySimpleType is considered as a subtype of anyType.

The second method allows the decoder both to resolve a schema at initialization phase (the *initial schema*) and to receive updated schema information called *additional schemas*. Additional schemas differ from the initial schema as the codes of their schema components defined in different namespaces are defined in different code spaces. This results in larger code size but has the required flexibility for late updating. For full flexibility it is also possible to receive exclusively additional schemas and thus to operate without initial schema.

Both initial schemas and additional schemas are part of a unique table in which each entry identifies a specific schema. The first entries identify schemas that are part of the initial schema. The following ones identify additional schemas.

To further improve compression, BiM allows the association of specific codecs to specific data types instead of using the generic mechanisms defined in Clause 8. These encoding schemes can be optimised with the full knowledge of the properties of that data type.

In subclause 5.2.3, replace this paragraph:

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, can be acquired on application demand, or appear in a different part of the tree.

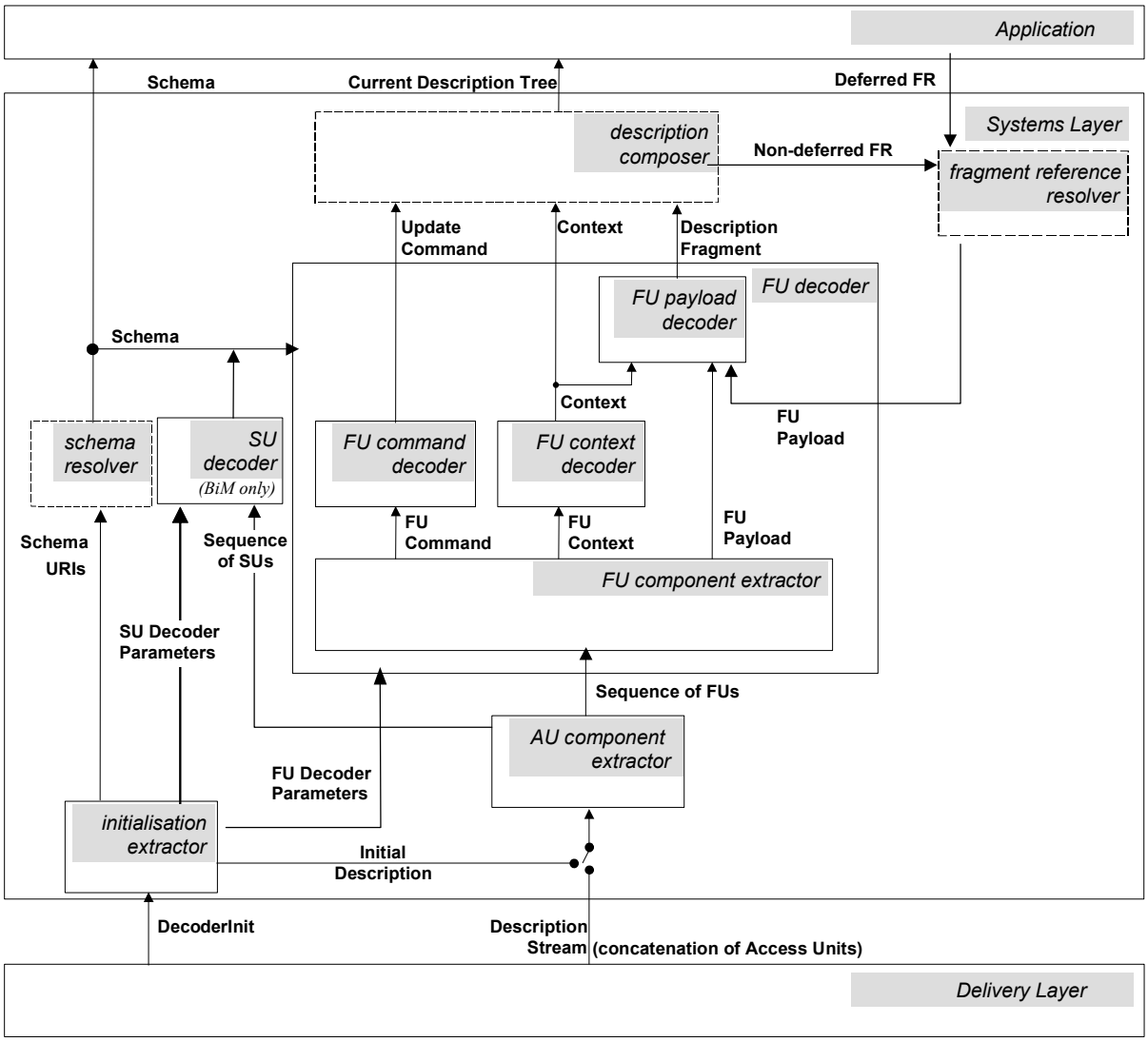
by

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, can be acquired on application demand, or appear in a different part of the tree.

Note - The schema update capabilities provided by the BiM framework defined in this specification aims at upgrading BiM decoder. It is not a mean to transmit an XML schema as is. To do so, one should use the W3C schema of schema to encode its schema.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 5.3, replace the "Figure 4 - Terminal Architecture" by the following figure:



In subclause 5.3, second paragraph, replace:

…(FU Decoder Parameters, in Figure …

by

…(FU Decoder Parameters and SU Decoder Parameters, in Figure …

In subclause 5.4, add the following text after the first paragraph:

In the case of BiM, an access unit is composed of any number of schema update units followed by any number of fragment update units which are extracted by the access unit component extractor.

A schema update unit carries parts of an additional schema and is composed of

— a namespace identifier,

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— a set of code tables to represent global elements, global types and global attributes,

— a binary encoded schema carrying the schema components definitions.

The full schema is not always necessary for the decoding of a particular binary description stream. To avoid unnecessary transmission, a schema update unit may contain only the definitions that are required for the decoding of the bitstream. In this case the code tables can also be sent partially.

Some further constraints are applied to the acquisition of schema update units, notably to ensure that a decoder will not break in case of a missed schema update unit. A specific schema update unit, the so-called first schema update unit, contains initialization information and shall be acquired by the decoder before any use of a received definition. The decoder behavior in case of such missed schema update units is not normative. A transmitted schema definition shall not change during binary description stream lifetime and there shall not be two schema identifiers associated to the same namespace. Finally, all the optimised decoders associated to existing types are immediately applied to all types they derive from in accordance to the rules defined for the optimized decoders in Clause 9.

Once received by the decoder, a schema update unit immediately updates schema information managed by the decoder. It becomes available for the fragment update unit carried in the same access unit as well as future access units.

In subclause 5.4, replace this paragraph:

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:

by

In case of TeM, an access unit is composed of any number of fragment update units.

In both TeM and BiM, fragment update units are extracted in sequence by the fragment update component extractor. Each fragment update unit consists of:

In subclause 5.4, replace the following paragraph:

— a fragment update payload conveying the coded description fragment to be added or replaced.

by

— a fragment update payload conveying either the coded description fragment (extracted out of the original description) to be added or replaced, or a reference to it.

In subclause 5.4, replace the following text:

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.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

by

The corresponding update command and context are processed by the non-normative description composer, which either places the description fragment extracted out of the original description or a reference to it received from the fragment update payload decoder at the appropriate node of the current description tree, or sends a reconstruction event containing this information to the application. If the payload consists of a fragment reference, depending on its nature, the referenced fragment is either immediately acquired (non-deferred fragment reference) or its acquisition is left to the application (deferred fragment references). In case of a deferred fragment reference, a fragment reference marker is available to the application to help further acquisition. This marker consists of the fragment reference itself, the name and type of the top most element of the referenced fragment. The fragment reference marker is added to the current description tree at the location defined by the fragment update context.

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.

Change the title of subclause 5.5.2 by:

Deferred nodes, fragment references and their use

In subclause 5.5.2, add the following sentence at the end of the first paragraph:

Some deferred nodes are marked with a fragment reference marker that specifies where the fragment can be acquired. It is then left to the application to decide when to acquire it.

In subclause 5.5.2, replace the following sentence:

… The deferred nodes may then be replaced in any subsequent access unit without changing the tree topology maintained internally in the decoder. …

by

… The deferred nodes may then be replaced in any subsequent access unit or on application demand without changing the tree topology maintained internally in the decoder. …

In subclause 5.5.3, remove "ISO/IEC 15938" of the second sentence of the first paragraph.

In subclause 5.5.3, add the following note at the end of the subclause:

Note – Forward compatibility can also be used to generate bitstreams that can be decoded even in case of a schema update unit has not been received (for example because an error occurred) or because the decoder is not able to accept schema update units.

In subclause 5.6.3, replace the following text:

In the TeM, the commands are AddNode, ReplaceNode, and DeleteNode. The AddNode is effectively an "append" command, adding an element of the target node. Insertion between two already-received,

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

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.

by

In the TeM, the commands are AddNode, ReplaceNode, and DeleteNode. The AddNode is by default an “append” command, adding an element after the last child of the target node. The position of the added element can also be explicitely defined allowing an element for instance to be inserted before the first element or between two other elements. By performing a DeleteNode on a node on the current description tree, the addressable indices of its siblings change appropriately.

In subclause 5.6.3, replace the following text:

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 . 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.

by

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. The insertion point for the node data is indicated using so called position codes. BiM supports 2 mechanisms for representing these position codes: integer or rational numbers. The first method (integer numbers) requires that "holes" must be left to enable the insertion of new node data. Thus documents to be sent must be known in advance or some contingency holes must be allocated. 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. The “hole” in the numbering is not visible in the current description tree generated by the description composer. However, it is not always possible to know in advance the number of nodes to be added and where within the description tree they are to be added. The second method (rational numbers) removes this limitation by enabling the insertion of new node data at any location within the current binary description tree. A single method shall be used for representing position codes within a given stream. This is signalled within the decoderInit.

In subclause 5.6.4, replace the following text:

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.

by

Wildcards (defined in ISO/IEC 15938-2) are supported by the BiM, only in the case the schema of the elements or attributes validated by the wildcard is known by the decoder (i.e. it is in the initial schema or in one of the additional schemas acquired by the decoder).

The BiM encodes only the canonical form of XML documents. Therefore comments as well as namespace prefixes are lost in the encoding process.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Add the following subclause (subclause 5.8):

5.8 Decoding of Fragment References

5.8.1 Decoding of Non-Deferred Fragment References

The result of decoding a non-deferred fragment reference shall be, passed to a mechanism (fragment reference resolver) which returns fragment update payload data to the FU payload decoder. This fragment update payload data may be in one of two forms:

— a TeM fragment update payload containing the description fragment data in case of a TeM bitstream;

— a BiM fragment update payload containing the description fragment data in case of a BiM bitstream.

Note - Examples of possible fragment resolver are:
- An HTTP communication session to a WEB server
- A DSM-CC Object carousel

5.8.2 Decoding of Deferred Fragment References

The result of decoding a deferred fragment reference shall be a fragment reference marker which consists of a fragment reference, the name and type of its top most element.

Note - This fragment reference marker is signalled to the application and can be used to acquire the fragment through the fragment reference resolver at any instant of the description stream.

In subclause 6.1, replace the following text:

```
<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>
```

by

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"

       xmlns:mpeg7s="urn:mpeg:mpeg7:systems:amd1:2004"
       targetNamespace="urn:mpeg:mpeg7:systems:amd1:2004"
       elementFormDefault="qualified"
       attributeFormDefault="unqualified">

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

</schema>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 6.5.2, replace the add command definition by:

addNode	Adds the node conveyed within the `FUPayload` to the context node or the parent node of the context node according to the value of the position attribute of the FUContext. If the position is set to the value “lastChild”, the node conveyed within the FUPayload shall be added to the context node as the last child of the context node, or if the position attribute is set to the value “prevSibling”, the node conveyed within the FUPayload shall be added to the parent node of the context node as the previous sibling of the context node.

In subclause 6.6.1, rename the “`FUContextType`” in the DDL definition to “`FragmentUpdateContextTypeBase`”.

In subclause 6.6.1, add the following DDL definition:

```
<complexType name="FragmentUpdateContextType">
   <simpleContent>
      <extension base="mpeg7s:FragmentUpdateContextTypeBase">
         <attribute name="position" default="lastChild">
            <simpleType>
               <restriction base="string">
                  <enumeration value="lastChild"/>
                  <enumeration value="prevSibling"/>
               </restriction>
            </simpleType>
         </attribute>
      </extension>
   </simpleContent>
</complexType>
```

In subclause 6.6.2, add the following semantics:

position	This attribute shall be only present when the FUCommand takes the value “addNode”, and indicates the position of the node added by the “addNode” command against the context node specified by the navigation path. This attribute can have the following values: — “lastChild” : the last child of the context node. — “prevSibling” : the previous sibling of the context node.

In subclause 6.7.1, replace the following text:

```
<complexType name="FragmentUpdatePayloadType">
     <sequence>
       <any processContents="skip" minOccurs="0"/>
     </sequence>

     <attribute name="hasDeferredNodes" type="boolean"
                use="required" default="false"/>
     <anyAttribute namespace="##other" processContents="skip" use="optional"/>
</complexType>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

by

```
<complexType name="FragmentUpdatePayloadType">
     <sequence>
        <any namespace="##other" processContents="skip" minOccurs="0"/>
        <element name="FragmentReference" type="mpeg7s:FragmentReferenceType"
                   minOccurs="0" />
     </sequence>

     <attribute name="hasDeferredNodes" type="boolean"
                 use="required" default="false"/>
     <anyAttribute namespace="##other" processContents="skip" use="optional"/>
</complexType>
```

In subclause 6.7.2, add the following semantics:

FragmentReference	Defines that the current fragment update payload carries a fragment reference instead of a complete fragment. This element shall not be present if fragment payload actually contains a fragment. The element name and type of the top most element of the fragment being referenced shall be carried before the fragment reference itself.

Add the following subclause (subclause 6.8):

6.8 Textual Fragment Reference

6.8.1 Syntax

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

<complexType name="FragmentReferenceType" abstract="true">
   <attribute name="isDeferred" type="boolean" use="optional" default="false"/>
</complexType>

<complexType name="URIFragmentReferenceType" >
   <complexContent>
      <extension base="mpeg7s:FragmentReferenceType">
         <attribute name="href" type="anyURI" use="required" />
      </extension>
   <complexContent>
</complexType>
```

6.8.2 Semantics

The `FragmentReferenceType` is an abstract complex type which serves as a base type for specific implementation of a fragment reference. The `URIFragmentReferenceType` is a concrete complexType which defines fragment references as a URI.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	*Definition*
isDeferred	Defines the deferred nature of the fragment reference: — If the `isDeferred` attribute has a value of true, the fragment reference is deferred and shall be resolved as defined in subclause 5.8.2. — If the `isDeferred` attribute has value of false, the fragment reference is non-deferred and shall be resolved as defined in subclause 5.8.1.
href	Defines the URI of a fragment refererence of type `URIFragmentReferenceType`.

6.8.3 Examples

In the following, example of the instances of the FUPayload datatype using the fragment reference is shown:

```
<FUPayload>
   <mpeg7:VisualDescriptor xsi:type="mpeg7:ScalableColorType"/>
   <FragmentReference xsi:type="mpeg7s:URIFragmentReferenceType"
      isDeferred="true"
      href="http://aaa.bbb/ccc.xml"/>
</FUPayload>
```

In subclause 7.1, add the following text and figure at the end of the subclause:

Identifying schema components in the BiM framework

As described in Clause 5, BiM relies upon schema knowledge. In this specification, schema components (elements, types and attributes) are identified by both a *schema identifier* and a *component identifier*.

The decoder manages both a unique initial schema and several additional schemas. From the decoder point of view, both initial schemas and additional schemas are indentified through a unique table in which each entry identifies a specific schema: the first '`NumberOfSchemas`' entries identify schemas that are part of the initial schema. The following ones identify additional schemas (starting at the '`NumberOfSchemas`' entry and ending at the '`NumberOfSchemas + NumberOfAdditionalSchemas - 1`').

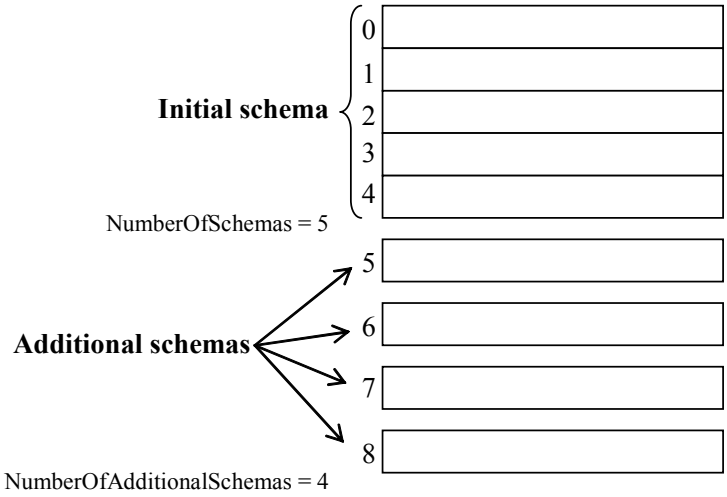


Figure AMD1-2 - Addressing the initial schema and the additional schemas

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The schema component codes (type codes, element codes or attribute codes) are accessible through all these schemas. However codes are constructed differently depending on which schema they are defined. The initial schema aggregates all schema components possibly coming from different namespaces in a single code space. On the contrary, additional schemas contains only schema components which are defined in their namespace.

In subclause 7.2.1, replace the following text:

An optimised type codec specifies an encoding for a data type not using the generic mechanism specified in Clause 8 but optimised with full knowledge of the properties of that data type. Some optimised type codecs are specified in Part 3 of this specification.

by

An optimised type codec specifies an encoding for a data type not using the generic mechanism specified in Clause 8 but optimised with full knowledge of the properties of that data type. There are two kinds of optimised type codec (or optimised decoders). A fixed optimised decoder associates a specific encoding scheme to a type of the schema (complex as well as simple) and this association is fixed for the entire stream. An advanced optimised decoder associates a specific encoding scheme to any simple type and this association can be changed during the transmission of the bitstream. Moreover, several advanced optimised decoders can be associated to a single type and can accept parameters. Some fixed optimised type codecs are specified in Part 3 of the specification. Some advanced optimised decoders are defined in Clause 9.

In subclause 7.2.1, add the following text at the end of the subclause:

Several other coding modes are initialised in the `DecoderInit` related to the features used by the binary description stream: the insertion of elements, the transmission of schema information and references to fragments.

Transmission of additional schema is specified for two different use cases: The retrieval of schema information in binary format from a location indicated by a URI, the transmission of schema information in a binary description stream jointly or not with the transmission of a description. In the latter case there is a requirement that all schema information needed for the decoding of a fragment of the transmitted description must have been received before such fragment arrives.

Replace subclause 7.2.2 content by:

DecoderInit () {	**Number of bits**	**Mnemonic**
SystemsProfileLevelIndication	8+	vluimsbf8
UnitSizeCode	3	bslbf
NoAdvancedFeatures	1	bslbf
ReservedBits	4	bslbf
If (! NoAdvancedFeatures) {		
AdvancedFeatureFlags_Length	8+	vluimsbf8
/** FeatureFlags **/		
InsertFlag	1	bslbf
AdvancedOptimisedDecodersFlag	1	bslbf
AdditionalSchemaFlag	1	bslbf

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AdditionalSchemaUpdatesOnlyFlag	1	bslbf
FragmentReferenceFlag	1	bslbf
MPCOnlyFlag	1	bslbf
HierarchyBasedSubstitutionCodingFlag	1	bslbf
ReservedBitsZero	FeatureFlags_Length*8-6	bslbf
/** FeatureFlags end **/		
}		
*/** Start FUUConfig **/*		
If(! AdditionalSchemaUpdatesOnlyFlag) {		
NumberOfSchemas	8+	vluimsbf8
for (k=0; k< NumberOfSchemas; k++) {		
SchemaURI_Length[k]	8+	vluimsbf8
SchemaURI[k]	8* SchemaURI_Length[k]	bslbf
LocationHint_Length[k]	8+	vluimsbf8
LocationHint[k]	8* LocationHint_Length[k]	bslbf
NumberOfTypeCodecs[k]	8+	vluimsbf8
for (i=0; i< NumberOfTypeCodecs[k]; i++) {		
TypeCodecURI_Length[k][i]	8+	vluimsbf8
TypeCodecURI[k][i]	8* TypeCodecURI_Length[k][i]	bslbf
NumberOfTypes[k][i]	8+	vluimsbf8
for (j=0; j< NumberOfTypes[k][i]; j++) {		
TypeIdentificationCode[k][i][j]	8+	vluimsbf8
}		
}		
}		
*/** FUUConfig - Advanced optimised decoder framework **/*		
If (AdvancedOptimisedDecodersFlag) {		
NumOfAdvancedOptimisedDecoderTypes	8+	vluimsbf8
for (i=0; i< NumOfAdvancedOptimisedDecoderTypes; i++) {		
AdvancedOptimisedDecoderTypeURI_Length[i]	8+	vluimsbf8
AdvancedOptimisedDecoderTypeURI[i]	8* AdvancedOptimisedDecoderTypeURI_Length[i]	bslbf
}		
AdvancedOptimisedDecodersConfig ()		
}		

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

*/** FUUConfig - Fragment reference framework **/*		
If (FragmentReferenceFlag) {		
NumOfSupportedFragmentReferenceFormat	8	uimsbf
for (i=0;i< NumOfSupportedFragmentReferenceFormat;i++) {		
SupportedFragmentReferenceFormat[i]	8	blsbf
}		
}		
}		
/** *end FUUConfig **/*		
If (AdditionalSchemaFlag) {		
AdditionalSchemaConfig ()		
}		
*/** Initial description **/*		
If (!AdditionalSchemaUpdateOnlyFlag) {		
InitialDescription_Length	8+	vluimsbf8
InitialDescription()		
}		
}		

In subclause 7.2.3, add the following semantics after the sematics of the `UnitSizeCode` *syntax element:*

NoAdvancedFeatures	Signals that none of the following advanced features is used in the binary stream: — dynamic insertions of child elements in the binary current description tree; — advanced optimised decoders; — schema transmission; — fragment references; — position codes only based on MPC.
AdvancedFeatureFlags_Length	Defines the number of bytes used for the indication of the advanced features. Note – This length provides a simple framework for future extensions.
InsertFlag	Signals that the insertion of child elements in the binary description tree at specific positions is performed by the use of rationale position code as described in subclause 7.6.5.5.
AdvancedOptimisedDecodersFlag	Signals that advanced optimised decoders are supported as described in Clause 9.
AdditionalSchemaFlag	Signals that additional schemas are supported.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

AdditionalSchemaUpdatesOnlyFlag	Signals that the description stream contains only additional schema updates i.e. no fragment update units. The `AdditionalSchemaFlag` shall be set to true when this flag is set to true.
FragmentReferenceFlag	Signals that fragment references are supported.
MPCOnlyFlag	Signals that position codes in the fragment update context are encoded in MPC mode only.
HierarchyBasedSubstitutionCodingFlag	Signals that element substitution codes are computed taking into account their substitution hierachy. If additional schemas are supported (i.e. `AdditionalSchemaFlag`==true) this flag shall be set to true.

In subclause 7.2.3, replace the "NumberOfSchemas" semantics by:

NumberOfSchemas	Indicates the number of schemas on which the description stream is based. These schemas compose the initial schema. A zero-value is forbidden.

In subclause 7.2.3, add the following semantics after the sematics of the `TypeIdentificationCode` syntax element

NumOfAdvancedOptimisedDecoderTypes	Defines the number of advanced optimised decoder types that are necessary to properly decode the binary description stream.
AdvancedOptimisedDecoderTypeURI_Length[i]	Indicates the size in bytes of the `AdvancedOptimisedDecoderTypeURI[i]` syntax element.
AdvancedOptimisedDecoderTypeURI[i]	Defines the UTF-8 representation of the URI referencing the advanced optimised decoder type with index i.
AdvancedOptimisedDecodersConfig()	See subclause 9.2.
NumOfSupportedFragmentReferenceFormat	Specifies the number of fragment reference format that shall be supported by the decoder.
SupportedFragmentReferenceFormat[i]	Specifies the ith fragment reference format, according to Table AMD1-1, that shall be supported by the decoder. The `SupportedFragmentReferenceFormat[0]` indicates the default fragment reference format.
AdditionalSchemaConfig()	See subclause 7.2.4.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.2.3, add the following table:

Table AMD1-1 - Fragment Reference Formats

Fragment Reference Type	***Fragment Reference Format***	***Description***
0		ISO reserved
1	URIFragmentReference	This fragment reference format should be used where the reference can be expressed as a URI.
2 - 224		ISO reserved
225 – 255		Private Use

Add the following subclause (subclause 7.2.4):

7.2.4 Syntax of AdditionalSchemaConfig

AdditionalSchemaConfig () {	**Number of bits**	**Mnemonic**
NumberOfAdditionalSchemas	8+	vluimsbf8
NumberOfKnownAdditionalSchemas	8+	vluimsbf8
for (int t=0;t<NumberOfKnownAdditionalSchemas;t++){		
KnownAdditionalSchemaID	8+	
AdditionalSchemaURI_Length[KnownAdditionalSchemaID]	8+	vluimsbf8
AdditionalSchemaURI[KnownAdditionalSchemaID]	8* AdditionalSchemaURI_Length [KnownAdditionalSchemaID]	bslbf
BinaryLocationHint_Length[KnownAdditionalSchemaID]	8+	vluimsbf8
BinaryLocationHint[KnownAdditionalSchemaID]	8*BinaryLocationHint_Length[KnownAdditionalSchemaID]	bslbf
NumberOfTypeCodecs[KnownAdditionalSchemaID]	8+	vluimsbf8
for (i=0; i< NumberOfTypeCodecs[KnownAdditionalSchemaID]; i++) {		
TypeCodecURI_Length[KnownAdditionalSchemaID][i]	8+	vluimsbf8
TypeCodecURI[KnownAdditionalSchemaID][i]	8* TypeCodecURI_Length[KnownAdditionalSchemaID][i]	bslbf
NumberOfTypes[KnownAdditionalSchemaID][i]	8+	vluimsbf8
for (j=0; j< NumberOfTypes[KnownAdditionalSchemaID][i]; j++) {		
TypeIdentificationCode[KnownAdditionalSchemaID][i][j]	8+	vluimsbf8
}		
}		

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
SchemaEncodingMethod	8	blsbf
ExternallyCastableTypeTable(InitialSchema)		
ExternallySubstitutableElementTable(InitialSchema)		
ReservedBitsZero	7	blsbf
}		

Add the following subclause (subclause 7.2.5):

7.2.5 Semantics of AdditionalSchemaConfig

Name	*Definition*
NumberOfAdditionalSchemas	Indicates the number of schemas that can be transmitted and that are not declared in the list of `schemaURI`. If additional schemas are not supported, this value is set to zero.
NumberOfKnownAdditionalSchemas	Indicates the number of additional schemas that are known to be updated in the bitstream.
KnownAdditionalSchemaID	Identifies a schema known to be updated in the bistream. This identifier shall only address an additional schema i.e. its value shall be superior to '`NumberOfSchemas-1`'
AdditionalSchemaURI_Length[KnownAdditionalSchemaID]	Indicates the size in bytes of the `AdditionalSchemaURI[KnownAdditionalSchemaID]` length. A value of zero is forbidden.
AdditionalSchemaURI[KnownAdditionalSchemaID]	Indicates the UTF-8 representation of the URI of the additional schema identified by `KnownAdditionalSchemaID`. Note – This field allows to identify some of the additional schemas that are expected to be updated. This information allows one decoder not to monitor the schema updates for which it already knows the schema.
BinaryLocationHint_Length[KnownAdditionalSchemaID]	Indicates the size in bytes of the `BinaryLocationHint_Length[KnownAdditionalSchemaID]`. A value of zero indicates that for the schema that is referenced by the index `KnownAdditionalSchemaID` there is no binary encoded schema available.
BinaryLocationHint[KnownAdditionalSchemaID]	This is the UTF-8 representation of the URI to unambiguously reference the location of the binary encoded schema that is referenced by the index `KnownAdditionalSchemaID`. The schema can be fetched by the schema resolver and is then received as a description stream composed only of schema update units i.e. for which the `SchemaOnlyFlag` is set to true.
NumberOfTypeCodecs[KnownAdditionalSchemaID]	see `NumberOfTypeCodecs[k]` in 7.2.3.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TypeCodecURI_Length[KnownAdditionalSchemaID]	see `TypeCodecURI_Length[k]` in 7.2.3.
TypeCodecURI[KnownAdditionalSchemaID][i]	see `TypeCodecURI[k][i]` in 7.2.3.
NumberOfTypes[KnownAdditionalSchemaID][i]	see `NumberOfTypes[k][i]` in 7.2.3.
TypeIdentificationCode[KnownAdditionalSchemaID][i][j]	see `TypeIdentificationCode[k][i][j]` in 7.2.3.
SchemaEncodingMethod	Indicates the encoding method of the schema update units.
ExternalCastableTypeTable	Defines the types of the initial schema that are externally castable as defined in subclause 7.7.5.4 and 7.7.5.5.
ExternalSubstitutableElementTable	Defines the elements of the initial schema that are substitutable as defined in subclause 7.7.6.4 and 7.7.6.5.

Table AMD1-2 - Schema encoding method

SchemaEncodingMethod	***definition***
0	ISO reserved
1	BiM encoded schema as described in subclause 7.7.8
2-224	ISO reserved
225-255	Private use

In subclause 7.3.2, replace the `AccessUnit` *syntax by:*

AccessUnit () {	**Number of bits**	**Mnemonic**
If (AdditionalSchemaFlag) {		
NumberOfSUU	8+	vluimsbf8
for (i=0; i< NumberOfSUU ; i++) {		
SchemaUpdateUnit()		
}		
}		
If(! AdditionalSchemaUpdateOnlyFlag) {		
NumberOfFUU	8+	vluimsbf8
for (i=0; i< NumberOfFUU ; i++) {		
FragmentUpdateUnit()		
}		
}		
}		

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.3.3, replace the semantics table by:

Name	*Definition*
NumberOfSUU	Indicates the number of schema update units in this access unit. Value ‘0’ signifies that no schema update unit is carried.
NumberOfFUU	Indicates the number of fragment update units in this access unit. Value '0' signifies that no fragment update unit is carried.

In subclause 7.4.2, insert the following syntax elements, between the `FUU_Length` *and the* `FragmentUpdateCommand` *syntax elements:*

If (AdvancedOptimisedDecodersFlag){		
OptimisedDecoderReparameterization	2	bslbf
if (OptimisedDecoderReparameterization == '00') {		
AdvancedOptimisedDecodersConfig ()		
}		
}		

In subclause 7.4.3, insert the following semantics, between the `FUU_Length` *and the* `FragmentUpdateCommand` *semantics:*

OptimisedDecodersReparameterization	This 2-bit flag signals if the parameters of the optimised decoders shall be updated. It can take the following values: — ‘00' – the optimised decoder instance table and mappings shall be redefined; — '01' – the optimised decoder instance table and mappings shall not be redefined; — '10' – the optimised decoder instance table and mappings are reset to the default table and mappings defined in the `DecoderInit`; — '11' – reserved.
AdvancedOptimisedDecodersConfig()	See subclause 9.2.

In subclause 7.6.1, replace the following text:

There are two different TBC tables associated to each complexType: The ContextTBC table contains only references to the child elements of complexType and additionally one code word to signal the termination of the path (Path Termination Code). The ContextTBC table contains also one TBC to refer to the parent node. It allows relative navigation within the binary format description tree and move upwards to the parent node. The OperandTBC table additionally contains also the references to the attributes and either to the elements of simpleType or to a simple content, but does not contain the Path Termination Code nor the reference to the parent node. Furthermore, in the OperandTBC table one TBC is reserved for User Data Extension as defined in section 7.6.5.2. Example TBC tables are shown in Table 4 and Table 5.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

by

There are two different TBC tables associated to each complexType: The ContextTBC table contains only references to the child elements of complexType and additionally one code word to signal the termination of the path (Path Termination Code). The ContextTBC table contains also one TBC to refer to the parent node. It allows relative navigation within the binary format description tree and move upwards to the parent node. The OperandTBC table additionally contains also the references to the attributes and either to the elements of simpleType or to a simple content, but does not contain the Path Termination Code nor the reference to the parent node. Furthermore, in the OperandTBC table one TBC is reserved for User Data Extension as defined in subclause 7.6.5.2. In case of a mixed content model the OperandTBC table also contains a reference to the character data that may appear between the elements. Example TBC tables are shown in Table 4 and Table 5.

Add the following sentence before the Table "Example of a Context TBC Table":

In this example the content model of the complex type definition is not 'mixed'.

In subclause 7.6.2, replace the `SchemaID` *syntax element in the* `FragmentUpdateContext` *syntax table:*

SchemaID	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf

In subclause 7.6.2, replace the `ContextPath` *syntax table by:*

ContextPath () {	**Number of bits**	**Mnemonic**
TBC_Counter = 0		
NumberOfFragmentPayloads = 1		
do {		
if ((ContextModeCode == '001' ‖ ContextModeCode == '011') && TBC_Counter ==0) { * /* absolute addressing mode and first TBC of the context path */*		
If (AdditionalSchemaFlag) {		
SchemaIDOfSBC_Context_Selector	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
Extended_SBC_Context_Selector	ceil(log2(number_of_global elements in `SchemaIDOfSBC_Context_Selector`))	bslbf
} else {		
SBC_Context_Selector	ceil(log2(number of global elements +1))	bslbf
}		
PathTypeCode()		

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

}		
else {		
SBC_Context	ceil(log2(number of child elements of complexType + 2))	bslbf
If (SBC_Context == “SBC_any”) {		
AnyElementDecoding ()		
} else {		
SubstitutionCode()		
}		
PathTypeCode()		
}		
TBC_Counter ++		
} while ((SBC_Context_Selector != “Path Termination Code”) && (SBC_Context != “Path Termination Code”))		
if (SBC_Context_Selector == “Path Termination Code”)){		
If (AdditionalSchemaFlag) {		
SchemaIDOfSBC_Operand_Selector	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
Extended_SBC_Operand_Selector	ceil(log2(number_of_global elements in `SchemaIDOfSBC_Operand_Selector`))	bslbf
} else {		
SBC_Operand_Selector	ceil(log2(number of global elements))	bslbf
}		
PathTypeCode()		
}		
else {		
SBC_Operand	ceil(log2(number of child elements + number of attributes + has_simpleContent + 1))	bslbf
if (SBC_Operand == “SBC_anyAttribute”) {		
SingleAnyAttributeDecoding()		
}		
if (SBC_Operand == “SBC_any”) {		
AnyElementDecoding()		
}		
SubstitutionCode()		
PathTypeCode()		
}		

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TBC_Counter ++		
for (i=0; i < TBC_Counter; i++) {		
PositionCode()		
}		

if ((ContextModeCode == '011') ||
(ContextModeCode == '100')) {
/ multiple fragment update payload mode*/*		

do {		
IncrementalPositionCode	ceil(log2(NumberOfMultiOccurrenceLayer+2))	bslbf
if (IncrementalPositionCode != "Skip_Indication") {		
NumberOfFragmentPayloads++		
}		
else {		
IncrementalPositionCode */* indicating the skipped position */*	ceil (log2(`NumberOfMultiOccurrenceLayer`+2))	bslbf
}		
} while (IncrementalPositionCode != "IncrementalPositionCodeTermination")		
NumberOfFragmentPayloads-- */* there is no fragment update payload corresponding to the IncrementalPositionCodeTermination */*		
}		
}		

In subclause 7.6.3, in the first table, replace the `SchemaID` *semantics by:*

SchemaID	Identifies the schema (from the list of `schemaURI`s transmitted in the `DecoderInit` (optionally extended by a list of additional schemas) which is used as basis for the fragment update context coding. The `SchemaID` code word is built by sequentially addressing the list of `SchemaURI` contained in the `DecoderInit` (optionally followed by the additional schemas). The length of this field is determined by: "ceil(log2(`NumberOfSchemas`))" or "ceil(log2(`NumberOfSchemas + NumberOfAdditionalSchemas`))" depending on the presence of additional schemas. The value of this code word is the same as the variable "k" in the definition of the `SchemaURI[k]` syntax element as specified in 7.2.3 optionally extended to additional schemas. The `SchemaID` syntax element is also used for the decoding of the fragment update payload as described in subclause 8.4.4. If the `ContextModeCode` selects a relative addressing mode then the `SchemaID` shall have the same value as in the previous fragment update unit.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.3, in the second table, add the following semantics after the `TBC_Counter` *semantic:*

SchemaIDOfSBC_Context_Selector	Identifies the schema in which the `Extended_SBC_Context_Selector` selects a declared global element.
Extended_SBC_Context_Selector	Selects one global element of the schema referenced by `SchemaIDOfSBC_Context_Selector` using the ContextTBC table as specified in 7.6.5.2.3.

In subclause 7.6.3, in the second table, add the following semantic after the `SBC_Context` *semantic:*

AnyElementDecoding()	See 8.5.2.4.5.2 and 8.5.2.4.5.3.

In subclause 7.6.3, in the second table, add the following semantics after the `SubstitutionCode` *semantic:*

SchemaIDOfSBC_Operand_Selector	Identifies the schema in which the `Extended_SBC_Operand_Selector` selects a declared global element.
Extended_SBC_Operand_Selector	Selects one global element of the schema referenced by `SchemaIDOfSBC_Operand_Selector` using the ContextTBC table as specified in 7.6.5.2.3.

In subclause 7.6.3, in the second table, add the following semantic after the `SBC_Operand` *semantic:*

SingleAnyAttributeDecoding()	See 8.5.3.3

In subclause 7.6.5.2.2, add the following bullet after the 6th bullet (i.e. "In the table for OperandTBCs the all-zero SBC…"):

— In the table for OperandTBCs the all-zero-and-one SBC (ex. 00001) is assigned to the character data in the mixed content of a datatype if the datatype has a mixed content model.

In subclause 7.6.5.2.2, replace the following text:

— All other Schema Branch Codes are assigned to the children nodes of the complexType. The children are defined as the attributes of the complex type as well as, either the contained elements or a dedicated child representing a simple content. If there are two or more element declarations with the same name in the complexType definition then each shall be assigned a different SBC.

by

— All other Schema Branch Codes are assigned to the children nodes of the complexType. The children are defined as the attributes of the complex type as well as, either the contained elements or a dedicated child representing a simple content. If there are two or more element declarations with the same name in the complexType definition then each shall be assigned a different SBC. If there is an "any" element declared in the complex type then a SBC is also assigned to this element and this SBC is called

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

"SBC_any". If there is an "anyAttribute" declaration in the complex type then a SBC is also assigned to it and this SBC is called "SBC_anyAttribute".

In subclause 7.6.5.2.2, replace the following text:

— The SBCs for child elements and simple content are assigned first, the SBCs for attributes are assigned last. The attributes are ordered lexicographically for the assignment of the Schema Branch Codes.

by

— The SBCs for child elements and simple content are assigned first, the SBCs for attributes are assigned last. The attributes are ordered lexicographically for the assignment of the Schema Branch Codes. The lexicographical ordering for an "any" element and for an "anyAttribute" is done with respect to their signature as defined in subclause 8.5.2.2.4

Replace the entire content of subclause 7.6.5.2.3 by:

For the special case of the selector node the following rules apply:

If the `AdditionalSchemaFlag` in the binary `DecoderInit` equals '0' then

— The length in bits of these SBCs is determined by the number of global elements declared in the schema referred by the `SchemaID` as follows:

  — SBC_Context_Selector: ceil(log2(number of global elements + 1)).

  — SBC_Operand_Selector: ceil(log2(number of global elements)).

— The SBCs are assigned sequentially to the global elements defined in the schema referred by the `SchemaID`. Before the assignment:

  — in case the `HierarchyBasedSubstitutionCodingFlag` is set to false, a lexicographical ordering of all global elements is performed.

  — in case the `HierarchyBasedSubstitutionCodingFlag` is set to true, a depth first ordering is performed with respect to the hierarchy of element substitutions which forms one or several trees as shown in Figure AMD1-3. For elements which are siblings within the element substitution hierarchy or roots of a substitution hierarchy a lexicographical ordering is performed based on their expanded name as defined in 8.2.

— No codes are assigned for a reference to the parent node nor for the User Data Extension Code. The Path Termination Code, however, is present in the ContextTBC table.

If the `AdditionalSchemaFlag` in the binary `DecoderInit` equals '1' then

— The length in bits of the `Extended_SBC_Context_Selector` respectively the `Extended_SBC_Operand_Selector` is determined by the number of global elements declared in the schema referred by the `SchemaIDOfSBC_Context_Selector` respectively `SchemaIDOfSBC_Operand_Selector` as follows:

  — Extended_SBC_Context_Selector: ceil(log2(number of global elements + 1)).

  — Extended_SBC_Operand_Selector: ceil(log2(number of global elements)).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— The SBCs are assigned sequentially to the global elements defined in the schema referred by the `SchemaIDOfSBC_Context_Selector` respectively `SchemaIDOfSBC_Operand_Selector`. No SBCs are assigned to elements imported from other namespaces into this schema. Before the assignment:

  — in case the `HierarchyBasedSubstitutionCodingFlag` is set to false, a lexicographical ordering of all global elements is performed.

  — in case the `HierarchyBasedSubstitutionCodingFlag` is set to true, a depth first ordering is performed with respect to the hierarchy of element substitutions which forms one or several trees as shown in Figure AMD1-3. For elements which are siblings within the element substitution hierarchy or roots of a substitution hierarchy a lexicographical ordering is performed based on their expanded name as defined in 8.2.

— No codes are assigned for a reference to the parent node nor for the User Data Extension Code. The Path Termination Code, however, is present in the ContextTBC table.

In subclause 7.6.5.3.1, add the following sentence at the end of the subclause:

The `GlobalSubstitutionSelect` is used when the substitute element is defined in a other schema than the expected element. In that case, the `GlobalSubstitutionSelect` selects the substitute element from the set of all elements defined in the schema referenced by the `SchemaID`.

In subclause 7.6.5.3.2, replace the `SubstitutionCode` syntax table by:

SubstitutionCode () {	**Number of bits**	**Mnemonic**
if (substitution_possible == 1 || external_element_substitution_possible == 1 || all_element_externally_substitutable == 1) {		
SubstitutionFlag	1	bslbf
if (SubstitutionFlag == 1) {		
if (external_element_substitution_possible == 1 || all_element_externally_substitutable == 1) {		
SchemaSwitching	1	bslbf
if (SchemaSwitching) {		
SchemaID	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
GlobalSubstitutionSelect	ceil(log2 (number_of_global_elements_in_schema_referred_by_SchemaID))	bslbf
} else {		
SubstitutionSelect	ceil(log2(number_of_possible_substitutes))	bslbf
}		

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

} else {		
SubstitutionSelect	ceil(log2(number_of_possible_substitutes))	bslbf
}		
}		
}		
}		

In subclause 7.6.5.3.3, add the following semantics after the `substitution_possible` *semantic:*

external_element_substitution_possible	This internal flag indicates whether the element can be subject to a substitution occurring in an other schema than the one in which the element is defined. This flag is set by the `ExternallySubstitutableType` table defined in subclause 7.7.6.4 and 7.7.6.5.
all_element_externally_substitutable	This internal flag indicates whether every element defined in the schema of the expected element can be subject to a substitution occurring in an other schema. This flag is set by the `ExternallySubstitutableType` table defined in subclause 7.7.6.4 and 7.7.6.5.

In subclause 7.6.5.3.3 add the following semantics after the `SubstitutionFlag` *semantic:*

SchemaSwitching	Indicates whether the element substitution occurs in an other schema than the schema where the expected element is defined.
SchemaID	Identifies the schema in which the substitute element is defined.
GlobalSubstitutionSelect	This code identifies the substitute element in the schema of index '`SchemaID`' in the `SchemaURI[k]` table. When the `HierarchyBasedSubstitutionCondingFlag` is set to false or when it is not defined in the `DecoderInit`, the code referring to the elements are assigned sequentially starting from zero after lexicographical ordering of all global elements using their expanded names as defined in subclause 8.2. When the `HierarchyBasedSubstitutionCodingFlag` is set to true, the `SubstitutionSelect` codes are assigned in a depth-first manner with respect to the hierarchy of element substitutions which forms on or several trees as shown in an example in Figure AMD1-3. For elements which are siblings within the element substitution hierarchy or elements which are roots of a substitution hierarchy the code words are assigned in a lexicographical order based on their expanded names. The order of elements that have a head of substitution in an other schema than the one identified by `SchemaID` are defined in the same relative order than if they were in the *initial schema*. The length of this field is determined by "ceil(log2(number_of_global elements in the schema identified by `SchemaID`))" in both cases. Note – If schema identified by `SchemaID` is an additional schema, the substitution select codes are computed on the set of all elements defined in the namespace identified by the `SchemaID` entry in the `SchemaURI` table of the `DecoderInit`.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.5.3.3, replace the semantic of `SubstitutionSelect` *by:*

SubstitutionSelect	This code is used as address within a substitution group where each element defined in the schema of the expected element is assigned a `SubstitutionSelect` code. When the `HierarchyBasedSubstitutionCondingFlag` is set to false or when it is not defined in the `DecoderInit`, the `SubstitutionSelect` codes referring to the elements are assigned sequentially starting from zero after lexicographical ordering of the element using their expanded names as defined in subclause 8.2. The length of this field is determined by "ceil(log2(number_of_possible_substitutes in the schema of the expected element))". In case the `HierarchyBasedSubstitutionCondingFlag` is true, the `SubstitutionSelect` codes are assigned in a depth-first manner with respect to the hierarchy of element substitutions which forms one or several trees as shown in an example in Figure AMD1-3. For elements which are siblings within the element substitution hierarchy or for elements which are roots of a substitution hierarchy the code words are assigned in a lexicographical order based on their expanded names. The element substitution code identifies the substitute element which is used in the encoded document. The length of the code word for the element substitution code is equal to "ceil(log2(number of possible_substitute in the schema of the expected element))". Note – If the schema identified by `SchemaID` is an additional schema, the substitution select codes are computed on the set of all elements defined in the namespace identified by the `SchemaID` entry in the `SchemaURI` table of the `DecoderInit`.

In subclause 7.6.5.3.3, add the following figure after the semantics table:

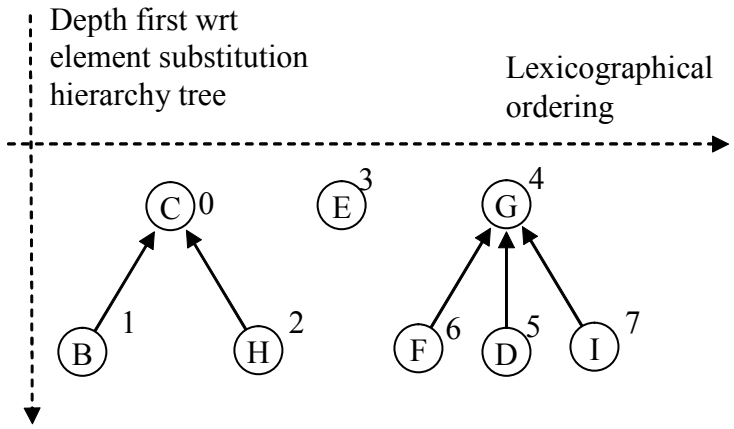


Figure AMD1-3 - Example for the Element Substitution Identification Code assignment for some elements in the hierarchy based coding mode

In subclause 7.6.5.4.1, add the following sentence at the end of the subclause:

The `GlobalTypeIdentificationCode` is used when the effective type is defined in an other schema than the expected type. In that case, the `GlobalTypeIdentificationCode` selects the effective type from the set of all types defined in the schema referenced by the `SchemaID`.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.5.4.2, replace the `PathTypeCode` *syntax table by:*

PathTypeCode () {	**Number of bits**	**Mnemonic**
if (type_cast_possible == 1 || external_type_cast_possible == 1 || all_type_externally_castable == 1) {		
TypeCodeFlag	1	bslbf
if ((TypeCodeFlag == 1) {		
if (external_type_cast_possible == 1 || all_type_externally_castable == 1) {		
SchemaSwitching	1	bslbf
if (SchemaSwitching) {		
SchemaID	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
GlobalTypeIdentificationCode	ceil(log2 (number_of_global_types_in_schema_referred_by_SchemaID))	bslbf
} else {		
TypeIdentificationCode	ceil(log2(number of derived types))	bslbf
}		
} else {		
TypeIdentificationCode	ceil(log2(number of derived types))	bslbf
}		
}		
}		
}		

In subclause 7.6.5.4.3 add the following semantics after the `type_cast_possible` *semantic:*

external_type_cast_possible	Indicates whether the expected type can be subject to a type casting occurring in an other schema than the one in which the type is defined. This flag is set by the `ExternallyCastableType` table defined in subclause 7.7.5.4 and 7.7.5.5.
all_type_externally_castable	Indicates whether every type defined in the schema of the expected type can be subject to a type casting occurring in an other schema. This flag is set by the `ExternallyCastableType` table defined in subclause 7.7.5.4 and 7.7.5.5.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.5.4.3 add the following semantics after the `TypeCodeFlag` *semantic:*

SchemaSwitching	Indicates whether the type cast occurs in an other schema than the schema of the expected type.
SchemaID	Identifies the schema in which the derived type element is addressed.
GlobalTypeIdentificationCode	Identifies a type defined in `SchemaIDOfDerivation` by a code word. The Type Identification Code is generated for a given type (simpleType or complexType) from the set of all types (itself being not included) including abstract types defined in the schema referenced by `SchemaID`. The Type Identification Codes are assigned in a depth-first manner with respect to the hierarchy of types which forms a tree as shown in an example in Figure 9. For types which are siblings within the type hierarchy the code words are assigned in a lexicographical order based on their expanded names. The Type Identification Code identifies the derived type which is used for the type cast. The length of the code word for the Type Identification Code is equal to “ceil(log2(number of types in the schema))”. The order of types that have a super type in an other schema than the one identified by `SchemaID` are defined in the same relative order than if they were in the initial schema. Note – If schema identified by `SchemaID` is an additional schema, the type codes are computed on the set of all types defined in the namespace identified by the `SchemaID` entry in the `SchemaURI` table of the `DecoderInit`.

In subclause 7.6.5.5.1, replace the following text:

Within a TBC a Position Code shall uniquely identify the position of a node among its sibling nodes in the binary description tree. It is present only if multiple occurrences are possible for the element referenced by the SBC or for any model group declared in the corresponding complexType definition. Position Codes are distinguished in Multiple element Position Codes (MPC) and Single element Position Codes (SPC) for efficiency reasons. The presence of the Position Code and the decision whether SPC or MPC are used is determined by the complexType definition.

by

Within a TBC a Position Code shall uniquely identify the position of a node among its sibling nodes in the binary description tree. Position Codes are distinguished in Multiple element Position Codes (MPC) and Single element Position Codes (SPC) for efficiency reasons. If the `MPCOnlyFlag` in the `DecoderInit` is set to true, MPC are always used. the `MPCOnlyFlag` in the `DecoderInit` is set to false, the presence of the Position Code and the decision whether SPC or MPC are used is determined by the complexType definition. In the second case, a position code is present only if multiple occurrences are possible for the element referenced by the SBC or for any model group declared in the corresponding complexType definition. Also the OperandTBC of character content in a mixed content model contains a position code.

In subclause 7.6.5.5.1, add the following text and figures after the note:

If the `InsertFlag` in the Binary `DecoderInit` is set to true then the Position Codes represent rational numbers (Rational Position Codes). Otherwise Position Codes represent integer numbers. In both cases the child elements are sorted in increasing order of these values.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Rational Position Codes are used to allow the insertion of child elements at any specific possible position in the binary description tree. Position Codes representing rational numbers are specified by the following rules:

— Rational Position Codes represent rational numbers in the interval]0<n<1[.

— Rational Position Codes are encoded in the *vlurmsbf5* format.

In Figure AMD1-4 an example of a binary description tree of the element A including an assignment of position codes to child elements B is given. The Position Codes representing rational numbers specify the order in which the child elements B reside in the binary description tree.

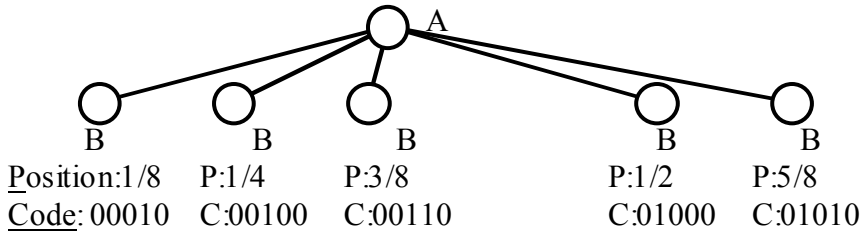


Figure AMD1-4 - Assignment of Position Codes to a set of child elements

When a new element B is inserted at any position then a new Position Code representing rational numbers is used so that the correct ordering in the binary description tree is unambiguously specified (see Figure AMD1-5).

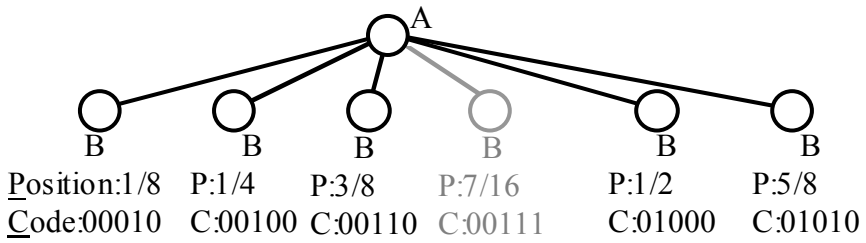


Figure AMD1-5 - Position Code of an inserted child element B (grey node)

In subclause 7.6.5.5.2, replace the following text:

A SPC is used, if a Position Code is present according to 7.6.5.5.1and if the corresponding complexType does not contain model groups with maxOccurs > 1. The SPC is only present if the SBC addresses an element with maxOccurs > 1. The SPC indicates the position of the node among the nodes addressed by the same SBC.

by

A SPC is used, if a Position Code is present according to 7.6.5.5.1, if the MPCOnlyFlag is set to false, and if the corresponding complexType does not contain model groups with maxOccurs > 1. The SPC is only present if the SBC addresses an element with maxOccurs > 1. The SPC indicates the position of the node among the nodes addressed by the same SBC.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.5.5.2, replace the following text:

The position is represented as integer value. The length in bits of the SPC is the equal to "ceil(log2(maxOccurs of the element addressed by the SBC))". If this length exceeds 4 bits then vluimsbf5 is used for coding the SPC.

by

If, according to 7.6.5.5.1, the position is represented as rational number, the value is encoded as vlurmsbf5 .The value 0 shall be omitted.

If, according to 7.6.5.5.1, the position is represented as integer value. The length in bits of the SPC is the equal to "ceil(log2(maxOccurs of the element addressed by the SBC))". If this length exceeds 4 bits then vluimsbf5 is used for coding the SPC.

In subclause 7.6.5.5.2, add the following text at the end of the subclause:

In the case of a complex type for which SPCs are used and that has mixed content the OperandTBC assigned to the character data in the mixed content also has a SPC. The position code of this OperandTBC is encoded assuming a maxOccurrence=MPA+1 since before and after each instantiated element character data can be present. The value MPA is specified in subclause 7.6.5.5.3.

Add the following text at the beginning of subclause 7.6.5.5.3:

If according to 7.6.5.5.1 the position is represented as rational number then the value is encoded as vlurmsbf5 and the value 0 shall be omitted.

If according to 7.6.5.5.1 the position is represented as integer number then the length in bits of the MPC is determined by the following method, which uses the 'max occurs' property of the effective content particles of the type definition.

Add the following text after the first paragraph:

In the case of a complex type that has mixed content model the OperandTBC assigned to the character data also uses a MPC.

Add the following text after the "For an element declaration particle" bullet:

In the case of a mixed content model the MPA_{mixed}=2MPA+1 since before and after each instantiated element character data can be present.

Replace subclause 7.6.5.5.4 by the following text and figure:

If an instantiated element was conveyed as part of a fragment update payload then the corresponding node has not been explicitly assigned a position in the binary format description tree. In this case, the following implicit positions are assigned to each added node for which a position code is expected in the TBC addressing this node:

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— If Position Codes represent integer numbers:

  — in the case a MPC is expected: a position is assigned incrementally (starting from zero) to the added elements.

  — in the case a SPC is expected: a position is assigned incrementally (starting from zero) to the added elements corresponding to the same SBC.

— If Position Codes represent rational numbers: to Z consecutive elements the positions represented by rational numbers are assigned by the following steps. In the case a MPC is expected: Z is the number of all elements which have the same parent node. In the case a SPC is expected: Z is the number of all elements which have the same parent node and which correspond to the same SBC.

  — In this steps, P(i) denotes the i-th assigned position.

    Step1: Determine Z.

    Calculate N=2^{ceil(log2(Z+1))},

    Set i=0,P=0.

    Step2: while(i≤(3Z-N+3)/2 | mod_2(i)==0) {P(i)=P+=1/(N); i++;}.

Step3: while (i<Z) {P(i)=P+=2/N; i++;}. End.In the implicit assignment of rational position codes, the step 2 performs an ascending-oriented assignment and the step 3 performs an balance-oriented assignment. The ascending-oriented assignment is efficient in case of appending subsequent fragments context paths, whereas the balance-oriented assignment is efficient in case of inserting/replacing subsequent fragments context paths. The condition of step 2 controls the ratio between such ascending and balance-oriented assignment. Figure AMD1-6 shows an example of the implicit assignment of positions represented by rational number.

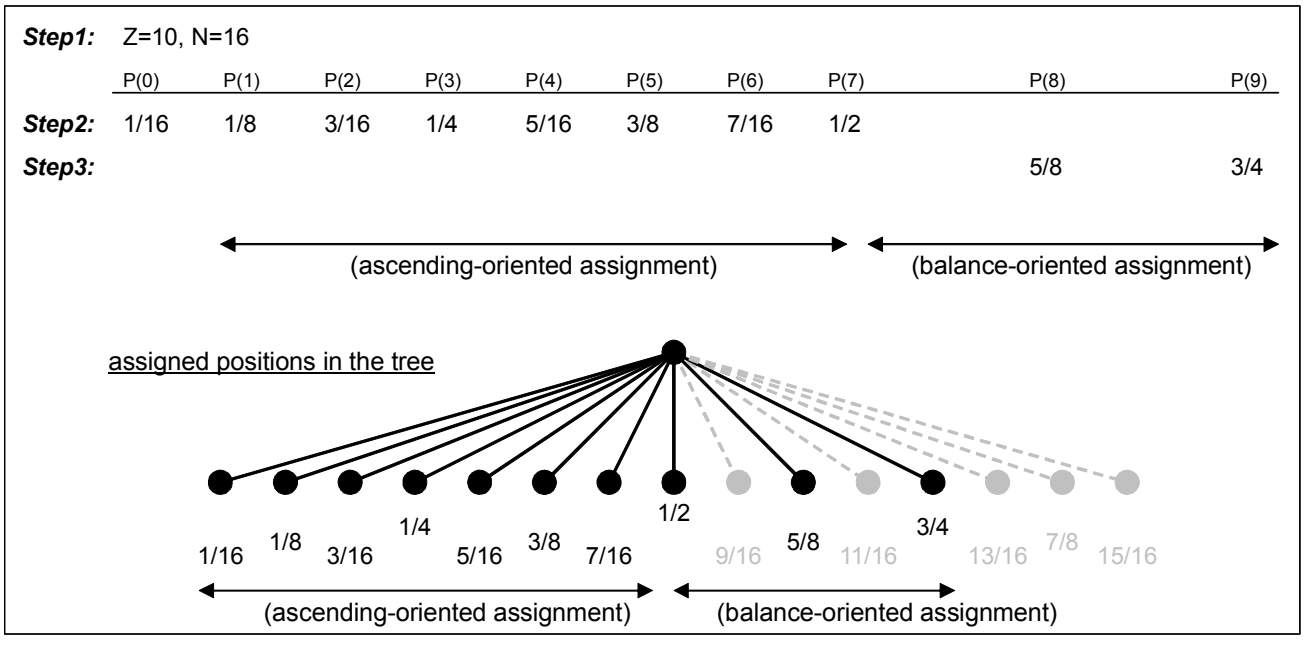


Figure AMD1-6 - an example on implicit assignment of positions represented by rational numbers

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 7.6.5.6, replace the first paragraph by:

A fragment update unit can contain multiple fragment update payloads of the same type if the context paths of those fragment update payloads are identical except for their position codes. If position codes represent integer numbers, the position codes for the first fragment update payload are coded in the same way as in the case of a single payload, while the position codes for the other fragment update payloads within this fragment update unit are indicated in the context path by “Incremental Position Codes” as shown in Figure 10.

In subclause 7.6.5.6, add the following paragraph before Figure 10:

In the case of rational position codes, the structure of the context path is the same as the case of integer position codes. Only the Position Codes and the Incremental Position codes differ. For the Context Path rational position codes are used. Also for the incremental position codes rational position increments are specified. The order of rational position increments are predefined (see Figure AMD1-7).

In subclause 7.6.5.6, replace the following text:

The set of multiple-occurrence nodes is indexed beginning from the starting node of the context path. An Incremental Position Code indicates the index of the multiple-occurrence node in the context path for which the position code shall be incremented by 1. The position code for all multiple-occurrence nodes with a higher index is set to “0”.

by

The set of multiple-occurrence nodes is indexed beginning from the starting node of the context path. An Incremental Position Code indicates the index of the multiple-occurrence node in the context path for which the position code shall be incremented. The position code for all multiple-occurrence nodes with a higher index is set to an initial value.

If Position Codes represent integer numbers, the position code for the multiple-occurrence node indicated by the Incremental Position Code shall be incremented according to the ascending order, i.e. it shall be incremented by “1”, and the initial value that the position code for all multiple-occurrence nodes with a higher index is set to is “0”.

If Position Codes represent rational numbers, the position codes shall be sorted in the following increment order of rational numbers before they are encoded:

1/2 → 1/4 → 3/4 → 1/8 → 3/8 → 5/8 → 7/8 → 1/16 → …

where, the i-th (i=0,1,2,…) rational number r[i] of this order is expressed by;

r[i] = (2(i+1) +1 - 2^j) / 2^j, where j = 1+int(log2(i+1)).

This order is defined first based on the resolution of the rational numbers which is the order of dividing the area (0,1) into halves repeatedly (Figure AMD1-7 (1)), i.e. the value of denominator. After that the ascending order is applied to the numbers in the same resolution (Figure AMD1-7 (2)).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

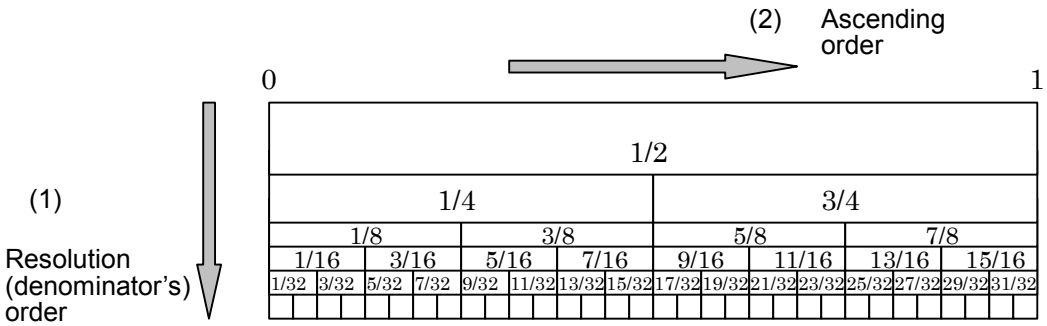


Figure AMD1-7 - Order of rational numbers

The position code for the multiple-occurrence node indicated by the Incremental Position Code shall be incremented according to the order of rational numbers. The initial value of the order is “1/2”.

The order of fragment update payloads in a fragment update unit is kept in the ascending order of their rational position code values. After decoding the position codes of rational numbers, the decoded position codes shall be re-sorted into the ascending order of their rational code values and be assigned to the multiple payload in this order.

In subclause 7.6.5.6, replace the following text:

An example for the multiple fragment update payload mode is given below:

by

An example for the multiple fragment update payload mode with integer position codes is given below:

Add the following text and figure at the end of subclause 7.6.5.6:

Figure AMD1-8 shows an example of the encoding/decoding processes for the multiple payload mode with rational position codes. If Position Codes represent rational numbers, the position codes are sorted in the order of rational numbers before they are encoded (Figure AMD1-8 (1)). Once the positions are sorted, incremental position coding is applied to the rational position codes with the increment order of rational numbers and the initial value “1/2”. After the position codes are decoded, they are re-sorted in the ascending order (Figure AMD1-8 (2)).

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

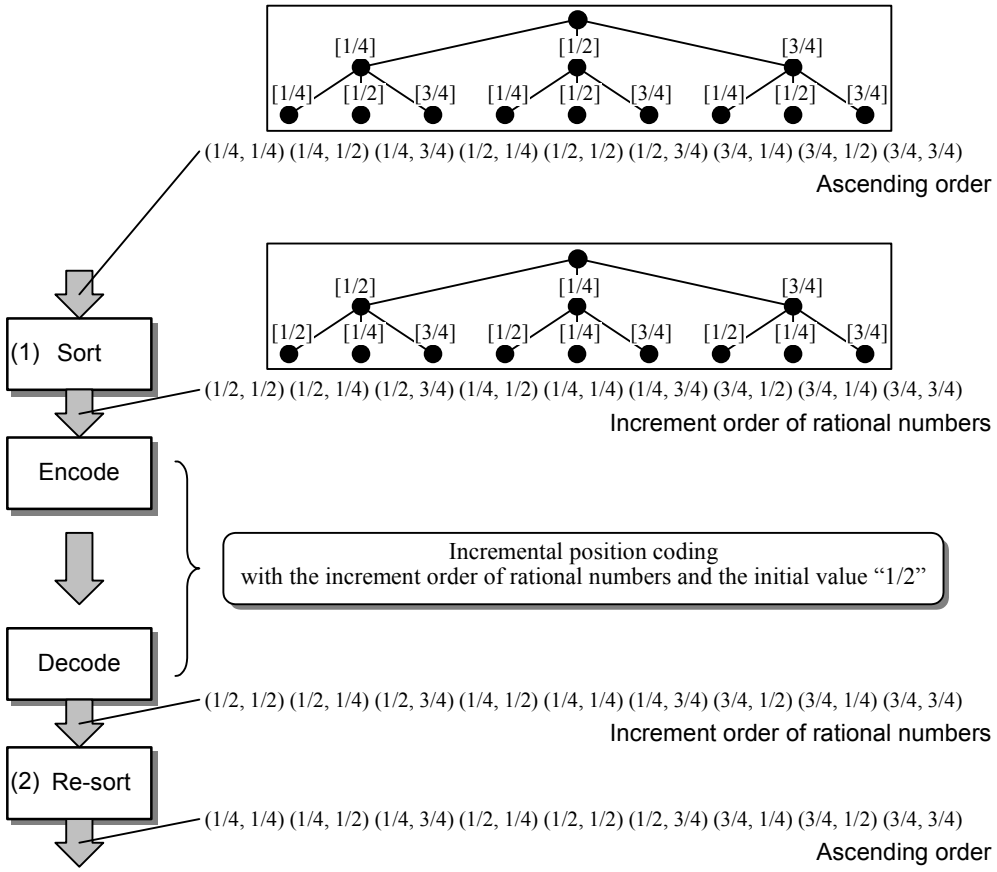


Figure AMD1-8 - Encoding/Decoding processes for the multiple payload mode with rational position codes

Add the following subclause (subclause 7.7):

7.7 Binary Schema Update Unit

7.7.1 Overview

In addition to the initial schema and known additional schemas, the decoder accept unknown additional schemas. These *unknown additional schemas* are subject to updates as described in this subclause. Uknown additional schema updates are carried in an access unit by a *schema update unit*.

A schema update unit is composed of a namespace identifier, a set of code tables to represent global elements, global types and global attributes, followed by a binary encoded schema carrying the schema components definitions. This binary encoded schema is encoded using a specific profile of BiM specified in subclause 7.7.8 using a simple XML schema for schema encoding has been defined for this purpose in this specification.

Note - The binary encoded schema only contains information needed by a BiM decoder to properly decode the bitstream. For instance the binary encoded schema does not carry key, unique elements or block. The schema update feature should not be used to carry XML schemas.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

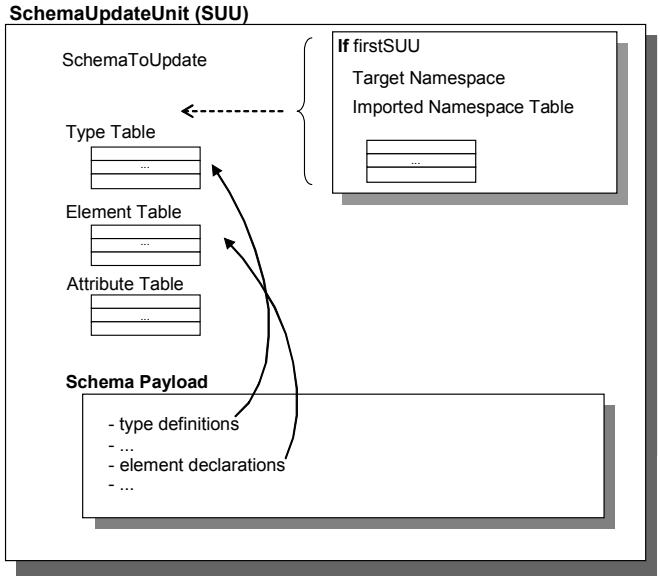


Figure AMD1-9 - A schema update unit

Some constraints are applied to the acquisition of schema update units. A specific schema update unit, the so-called first schema update unit, contains initialization information and shall be acquired by the decoder before any description conformant to the transmitted additional schema is decoded. The decoder behavior in case of a first schema update units is missed is not normative. A schema definition already transmitted shall not change during binary description stream lifetime and there shall not be two schema identifiers associated to the same namespace.

The full schema is not always necessary for the decoding of a particular binary description stream. To avoid unnecessary transmission, a schema update unit may contain only the definitions that are required for the decoding of the description stream.

Once received by the decoder, a schema update unit immediately updates schema information managed by the decoder. All the optimised decoders associated to existing types are immediately applied to all types they derive from in accordance to the rules defined for the optimized decoders in Clause 9.

7.7.2 Syntax

SUU () {	**Number of bits**	**Mnemonic**
SchemaToUpdate	ceil(log2(NumberOfAdditionalSchemas))	uimsbf
FirstSUU	1	blsbf
If (FirstSUU) {		
NamespaceURI_Length	8+	vluimsbf8
NamespaceURI_String	8*NamespaceURILength	blsbf
ImportedNamespaceTable()		
}		
SchemaTypeTable(SchemaToUpdate)		
SchemaElementTable(SchemaToUpdate)		
SchemaAttributeTable(SchemaToUpdate)		
BinaryEncodedSchema(SchemaToUpdate)		
}		

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.3 Semantics

Name	*Definition*
SchemaToUpdate	Specifies the index in the table of schema which is updated by this fragment update unit.
FirstSUU	This flag is set to true if the SUU is a FirstSUU.
NamespaceURI_Length	Signals the length in bytes of the NamespaceURI_String.
NamespaceURI_String	UTF-8 representation of the namespace URI on which the SUU applies.
ImportedNamespaceTable	This table conveys the namespace referenced in the binary encoded schema as specified in subclause 7.7.4.
SchemaTypeTable	This table conveys the type code tables as specified in subclause 7.7.5.
SchemaElementTable	This table conveys the global elements and their possible substitutions as specified in subclause 7.7.6.
SchemaAttributeTable	This table conveys the global attributes as specified in subclause 7.7.7.
BinaryEncodedSchema	This conveys the binary encoded schema definitions as specified in subclause 7.7.8.

7.7.4 Imported NamespaceTable

7.7.4.1 Overview

This table conveys the table of namespaces that are referenced in the binary encoded schema.

7.7.4.2 Syntax

ImportedNamespaceTable(Schema){	**Number of bits**	**Mnemonic**
NumberOfImportedNamespaces	8+	vluimsbf8
for (i=0; i < NumberOfNamespaces; i++) {		
ImportedNamespace_Length[i]	8+	vluimsbf8
ImportedNamespace[i]	8* MappedNamespace_Length[i]	blsbf
}		
}		

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.4.3 Semantics

Name	*Definition*
NumberOfImportedNamespaces	Indicates the number of namespaces that can be referred by a schema component definition in the binary encoded schema.
ImportedNamespace_Length[i]	Indicates the size in bytes of the `ImportedNamespace[k]`. A value of zero is forbidden.
ImportedNamespace[i]	This is the UTF-8 representation of the namespace.

7.7.5 Schema Type Table

7.7.5.1 Overview

This table conveys the table of global types defined in the namespace on which the SUU applies.

7.7.5.2 Schema Type Table Syntax

SchemaTypeTable (Schema){	**Number of bits**	**Mnemonic**
if (FirstSUU) {		
NumberOfGlobalTypes	8+	vluimsbf8
ExternallyCastableTypeTable(Schema)		
}		
PartialTransmission	1	blsbf
if (PartialTransmission){		
NumberOfTransmittedTypes	5+	vluimbsf5
for (i=0; i < NumberOfTransmittedTypes; i++) {		
TransmittedType	5+	vluimsbf5
NumOfSubtypes[TransmittedType]	5+	vluimsbf5
} else {		
for (i=0; i < NumberOfGlobalTypes; i++) {		
NumOfSubtypes[i]	5+	vluimsbf5
}		
}		

7.7.5.3 Schema Type Table Semantics

The types are defined in the order of their type codes within the namespace as specified in subclause 7.6.5.4.

Name	*Definition*
NumberOfGlobalTypes	Defines the number of global types defined in the namespace.
PartialTransmission	Indicates that the transmission of the type table is partial.
NumberOfTransmittedTypes	Indicates the number of type definitions that are transmitted in the current SUU.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

TransmittedType	Indicates the TypeCode of the type to be updated.
NumOfSubtypes[TransmittedType]	Indicates the number of subtype of the type 'TransmittedType' in the namespace.
NumOfSubtypes[i]	Indicates the number of subtype of the 'ith' type of the namespace.

7.7.5.4 Externally Castable Type Table Syntax

ExternallyCastableTypeTable(Schema) {	**Number of bits**	**Mnemonic**
IsThereExternallyCastableType	1	blsbf
If (IsThereExternalCastableType) {		
all_type_externally_castable	1	blsbf
If(!all_type_externally_castable) {		
NumberOfExternallyCastableType	5+	vluimsbf5
for(i=0;i< NumberOfExternallyCastableType; i++){		
ExternallyCastableType	ceil(log2(NumberOfGlobalTypes in Schema))	blsbf
}		

7.7.5.5 Externally Castable Type Table Semantics

This table allows to specify which types can be subject to a type casting where the subtype is defined in an other namespace than the one carried in the schema update unit.

Name	*Definition*
IsThereExternallyCastableType	Signals that some types in the schema to update can be casted into types defined defined in other namespaces.
all_type_externally_castable	Signals that all types in the schema to update can be casted into types defined in other namespaces.
NumberOfExternallyCastableType	Indicates the number of types that can be casted into types defined in other namespaces.
ExternallyCastableType	Indicates the type code of a type which can be casted into types defined in other namespaces. In case this element is subject to a substitution (subclause 7.6.5.4), its `external_type_cast_possible` flag is set to '1'.

7.7.6 Schema Element Tables

7.7.6.1 Overview

This table conveys the global elements and their substitutions on which the SUU applies. They are used to efficiently encode XML Schema substitution groups.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.6.2 Schema Element Table Syntax

SchemaElementTable (Schema){	**Number of bits**	**Mnemonic**
if (FirstSUU) {		
NumberOfGlobalElements	8+	vluimsbf8
ExternallyCastableElementTable(Schema)		
}		
PartialTransmission	1	blsbf
if (PartialTransmission){		
NumberOfTransmittedElements	5+	vluimbsf5
for (i=0; i < NumberOfTransmittedElements; i++) {		
TransmittedElement	5+	vluimsbf5
NumOfSubstituteElements[TransmittedElement]	5+	vluimsbf5
} else {		
for (i=0; i < NumberOfGlobalElements; i++) {		
NumOfSubstituteElements[i]	5+	vluimsbf5
}		
}		

7.7.6.3 Schema Element Table Semantics

The elements are defined in the order of their element codes within the namespace as specified in subclause 7.6.5.3.

Note – `HierarchyBasedSubstitutionCondingFlag` is always set to true when additional schemas are supported.

Name	*Definition*
NumberOfGlobalElements	Defines the number of global elements defined in the namespace.
PartialTransmission	Indicates that the transmission of the element table is partial.
NumberOfTransmittedElements	Indicates the number of element definitions that are transmitted in the current SUU.
TransmittedElement	Indicates the `SubstitutionSelect` code of the element to be updated.
NumOfSubstituteElements [TransmittedElement]	Indicates the number of substitute elements of the TransmittedElement in the updated namespace.
NumOfSubstituteElements [i]	Indicates the number of substitute elements of the 'i^{th}' element of the namespace.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.6.4 Externally Substitutable Element Table Syntax

ExternallySubstitutableElementTable(Schema) {	**Number of bits**	**Mnemonic**
IsThereExternallySubstitutableElement	1	blsbf
If (IsThereExternallySubstitutableElement) {		
all_element_externally_substitutable	1	blsbf
If(!all_element_externally_substitutable) {		
NumberOfExternallySubstitutableElement	5+	vluimsbf5
for(i=0;i< NumberOfExternallySubstitutableElement;i++){		
ExternallySubstitutableElement	ceil(log2(Number OfGlobalElements in Schema))	blsbf
}		

7.7.6.5 Externally Substitutable Element Table Semantics

This table allows to specify which elements can be subject to an “external” element substitution i.e. a substitution in which the substitute element is defined in an other namespace.

Name	*Definition*
IsThereExternallySubstituableElement	Signals that some elements in the schema to update can be substituted into elements defined defined in other namespaces.
all_element_externally_substitutable	Signals that all elements in the schema to update can be substituted into elements defined in other namespaces.
NumberOfExternallySubstitutableElement	Indicates the number of elements that can be substituted into elements defined in other namespaces.
ExternallySubstitutableElement	Indicates the element code of an element which can be substituted into elements defined in other namespaces. In case this element is subject to a substitution (subclause 7.6.5.3), its `external_element_substitution_possible` flag is set to ‘1’.

7.7.7 Schema Attribute Table

7.7.7.1 Overview

This table conveys the global attributes of the updated schema.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.7.2 Syntax

SchemaAttributeTable(Schema){	**Number of bits**	**Mnemonic**
if (FirstSUU) {		
NumberOfGlobalAttributes	8+	vluimsbf8
}		
PartialTransmission	1	blsbf
if (PartialTransmission){		
NumberOfTransmittedAttributes	5+	vluimbsf5
for (i=0; i < NumberOfTransmittedAttributes; i++) {		
TransmittedAttribute	ceil(log2(NumberOfGlobal Attributes in SchemaToUpdate))	uimsbf
}		
}		

7.7.7.3 Semantics

Name	*Definition*
NumberOfGlobalAttributes	Defines the number of global attributes defined in the namespace.
PartialTransmission	Indicates that the transmission of the element table is partial.
NumberOfTransmittedAttributes	Indicates the number of global attribute definitions that are transmitted in the current SUU.
TransmittedAttribute	Indicates the code of the received attribute.

7.7.8 Binary Encoded Schema

7.7.8.1 Overview

Each schema update unit carries a set of schema components definition in its `BinaryEncodedSchema`. This set is represented by an XML file conformant to a specific schema called the schema for encoding schema components. It is carried in a BiM encoded form using the schema for encoding schema components.

Note – The schema for encoding schema components is similar in its spirit to the XML Schema for schema. It has been however dedicated to the encoding of XML in BiM and not for validation as it is the case for the XML Schema for schema. It therefore concentrates on the features that are only used by a BiM decoder for decoding purposes only.

7.7.8.2 Decoding schema components using BiM

7.7.8.2.1 Binary Encoded Schema - DecoderInit

The following specific `DecoderInit` is used by the decoder for the decoding of binary encoded schema.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

DecoderInit() {	**Value**	**Number of bits**
SystemsProfileLevelIndication	0x00	8
UnitSizeCode	000	3
NoAdvancedFeatures	0	1
ReservedBits	1111	4
AdvancedFeatureFlags_Length	0x01	8
InsertFlag	0	1
AdvancedOptimisedDecodersFlag	1	1
AdditionalSchemaFlag	0	1
AdditionalSchemaUpdatesOnlyFlag	0	1
FragmentReferenceFlag	0	1
MPCOnlyFlag	0	1
ReservedBitsZero	00	2
NumberOfSchemas	1	8+
SchemaURI_Length[0]	0x20 (i.e. 32)	8+
SchemaURI[0]	"urn:mpeg:mpeg7:schemaUpdate:2002"	8* SchemaURI_Length[0]
LocationHint_Length[0]	0x00	8
NumOfAdvancedOptimisedDecoderTypes	0x01	8+
AdvancedOptimisedDecoderTypeURI_Length[0]	0x40 (i.e. 64)	8+
AdvancedOptimisedDecoderTypeURI[0]	"urn:mpeg:mpeg7:systems:SystemsAdvancedOptimisedDecodersCS:2003:1"	8* AdvancedOptimisedDecoderTypeURI_Length[0]
AdvancedOptimisedDecodersConfig () {		
NumOfAdvancedOptimisedDecoderInstances	0x00	8
NumOfMappings	0x00	8
}		
InitialDescription_Length	0x00	8
}		
}		

7.7.8.2.2 Binary Encoded Schema - Access Unit Constraints

A schema update unit is carried in one access unit constrained by the following rules:

— The access unit shall contain only one fragment update unit ;

— The fragment update unit shall update the top most node ;

— The fragment update unit shall use a 'AddContent" command ;

— The fragment update unit shall have a context mode code set to '001' ;

— The `lengthCodingMode` code of the fragment update payload shall be set to '00' ;

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

— The `hasDeferredNodes` flag of the fragment update payload shall be set to '0' ;

— The `hasTypeCasting` flag of the fragment update payload shall be set to '1' ;

— The `hasNoFragmentReference` flag of the fragment update payload shall be set to '1'.

Moreover in the fragment update payload the following rules applies:

— The references (e.g. "base" or "type" attributes in XML Schema) to elements, types or attributes are encoded with "SchemaID" (in the local imported namespaces table) + "global code"

7.7.8.2.3 Binary Encoded Schema – Schema

The encoded schema shall respect the following constraints:

— Global types, elements and attributes are encoded in the same order than the one defined by their respective tables in the schema update units (`SchemaTypeTable`, `SchemaElementTable` and `SchemaAttributeTable`);

— Attributes in complex type definitions are sorted according to their expanded name;

— Content models shall be normalized as described in subclause 8.5.2.2.4.

7.7.8.3 Mapping schema components to the schema for encoding

7.7.8.3.1 Overview

The following subclauses specify the syntax elements and associated semantics of the schema for encoding schema updates.

The following schema wrapper shall be applied to the syntax defined in subclause 7.7.8.3.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"

   xmlns:m7s="urn:mpeg:mpeg7:systems:encodingschema:amd1:2004"
   targetNamespace="urn:mpeg:mpeg7:systems:encodingschema:amd1:2004"
     elementFormDefault="qualified"
     attributeFormDefault="unqualified">

   <!-- here clause 7.7.8.3 schema definitions -->

</schema>
```

7.7.8.3.2 Main schema element and type

7.7.8.3.2.1 Syntax

```
<xs:element name="schema" type="schemaType"/>

<xs:complexType name="schemaType">
  <xs:sequence>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
   <xs:element name="typeDefinitions" type="typeDefinitionsType" minOccurs="0"/>
   <xs:element name="anonymousTypeDefinitions"
type="anonymousTypeDefinitionsType"
                                        minOccurs="0"/>
   <xs:element name="elementDeclarations" type="elementDeclarationsType"
                                        minOccurs="0"/>
   <xs:element name="attributeDeclarations" type="attributeDeclarationsType"
     minOccurs="0"/>
  </xs:sequence>
  <xs:attribute name="elementFormDefault"   type="qualificationType"
                          use="optional" default="unqualified"/>
  <xs:attribute name="attributeFormDefault" type="qualificationType"
                          use="optional" default="unqualified"/>
</xs:complexType>

<xs:simpleType name="qualificationType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="qualified"/>
    <xs:enumeration value="unqualified"/>
  </xs:restriction>
</xs:simpleType>
```

7.7.8.3.2.2 Semantics

Name	*Definition*
schema	The root element of the schema update.
schemaType	The set of schema components updated by this schema update. Note - The namespace of this schema is encoded within the SchemaUpdateUnit and therefore not encoded here.
typeDefinitions	conveys the list of named types (or type globally defined).
anonymousTypeDefinitions	conveys the list of anonymous types (or type locally defined).
elementDeclarations	conveys the list of global elements.
attributeDeclarations	conveys the list of global attributes.
elementFormDefault	identical to the ‘elementFormDefault’ attribute defined in XML schema.
attributeFormDefault	identical to the ‘attributeFormDefault’ attribute defined in XML schema.
qualificationType	A type used to define the qualification (qualified/unqualified) of elements and attributes. This type is used by the 'form', 'elementFormDefault' and 'attributeFormDefault' attributes.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.8.3.3 Element Declaration

7.7.8.3.3.1 Syntax

```
<xs:complexType name="elementDeclarationsType">
  <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="globalElement">
      <xs:complexType>
        <xs:sequence>
          <xs:element name="elementTypeReference" type="typeReferenceType"/>
        </xs:sequence>
        <xs:attribute name="name" type="xs:string" use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="nameStringType">
  <xs:restriction base="xs:string"/>
</xs:simpleType>
```

7.7.8.3.3.2 Semantics

Name	*Definition*
elementDeclarationsType	The list of all global element declarations carried by this schema update unit. This list shall be ordered as specified in the specification (See subclause 7.7.6).
nameStringType	Defines the type of all the names used in the schema i.e. attribute, element and type names.

7.7.8.3.4 Type Declaration

7.7.8.3.4.1 Syntax

```
<xs:complexType name="typeDefinitionsType">
  <xs:choice maxOccurs="unbounded">
    <xs:element name="complexType" type="namedComplexTypeType"/>
    <xs:element name="simpleType" type="namedSimpleTypeType"/>
  </xs:choice>
</xs:complexType>

<xs:complexType name="anonymousTypeDefinitionsType">
  <xs:element name="complexType" type="anonymousComplexTypeType"
                                  maxOccurs="unbounded"/>
  <xs:element name="simpleType" type="anonymousSimpleTypeType"
    maxOccurs="unbounded"/>
</xs:complexType>
```

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.8.3.4.2 Semantics

Name	*Definition*
typeDefinitionsType	This type conveys the list of all the global types (complex and simple) carried by this schema update. The global type (or named type) are the types globally defined in an XML schema declaration.This list shall be ordered as defined in 7.7.5. The number of types contained in this list is encoded in the `SchemaTypeTable` (see 7.7.5.2)
complexType	conveys a complex type definition.
simpleType	conveys a simple type definition.
anonymousTypeDefinitionsType	The list of all the anonymous types (or locally defined). No order is required on this list. In the case of a partial transmission, all the anonymous type required for resolving the type referencing mechanism shall be present in the schema update unit (see type referencing mechanism in 7.7.8.3.7).
complexType	conveys a complex type definition.
simpleType	conveys a simple type definition.

7.7.8.3.5 Type Definition

7.7.8.3.5.1 Syntax

```
<xs:complexType name="typeType" abstract="true">
   <xs:sequence>
     <xs:element name="derivation" minOccurs="0">
       <xs:complexType>
         <xs:sequence>
             <xs:element name="baseTypeReference" type="typeReferenceType"
                                              minOccurs="1"/>
         </xs:sequence>
         <xs:attribute name="type" type="derivationType"
           use="required"/>
       </xs:complexType>
     </xs:element>
   </xs:sequence>
</xs:complexType>

<xs:simpleType name="derivationType">
   <xs:restriction base="xs:string">
     <xs:enumeration value="extension"/>
     <xs:enumeration value="restriction"/>
   </xs:restriction>
</xs:simpleType>

<xs:complexType name="complexTypeType" abstract="true">
   <xs:complexContent>
     <xs:extension base="typeType">
       <xs:sequence>
         <xs:element name="attributes" minOccurs="0">
```

--`,,```,,,,````-`-`,,`,,`,`,,`---

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

```
        <xs:complexType>
          <xs:sequence>
            <xs:choice minOccurs="0" maxOccurs="unbounded">
              <xs:element name="attribute" type="localAttributeType"/>
              <xs:element name="attributeRef" type="attributeRefType"/>
            </xs:choice>
            <xs:element name="anyAttribute" type="anyAttributeType"
              minOccurs="0"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="content" type="contentModelType"/>
    </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="namedComplexTypeType" >
 <xs:complexContent>
  <xs:extension base="complexTypeType">
   <xs:attribute name="name" type="nameStringType" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="anonymousComplexTypeType">
 <xs:complexContent>
  <xs:extension base="complexTypeType">
   <xs:attribute name="id" type="AnonymousTypeIDType" use="required"/>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>

<xs:complexType name="simpleTypeType" abstract="true">
 <xs:complexContent>
  <xs:extension base="typeType">
   <xs:sequence>
    <xs:choice>
     <xs:element name="list">
      <xs:complexType>
       <xs:sequence minOccurs="1">
        <xs:element name="itemTypeReference" type="typeReferenceType"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
     <xs:element name="union">
      <xs:complexType>
       <xs:sequence minOccurs="1" maxOccurs="unbounded">
        <xs:element name="memberTypeReference" type="typeReferenceType"/>
       </xs:sequence>
      </xs:complexType>
     </xs:element>
    </xs:choice>
    <xs:sequence minOccurs="0" maxOccurs="unbounded">
     <xs:element name="facet" type="facetType"/>
    </xs:sequence>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
<xs:complexType name="namedSimpleTypeType" >
   <xs:complexContent>
     <xs:extension base="simpleTypeType">
       <xs:attribute name="name" type="nameStringType" use="required"/>
     </xs:extension>
   </xs:complexContent>
</xs:complexType>

<xs:complexType name="anonymousSimpleTypeType">
   <xs:complexContent>
     <xs:extension base="simpleTypeType">
       <xs:attribute name="id" type="AnonymousTypeIDType" use="required"/>
     </xs:extension>
   </xs:complexContent>
</xs:complexType>
```

7.7.8.3.5.2 Semantics

Name	*Definition*
typeType	The abstract type of all types. This type defines derivation information for its subtypes `complexTypeType` or `simpleTypeType`.
derivation	defines the derivation methods by which a type is defined from its supertype (extension or restriction).
baseTypeReference	identifies the supertype of the types as defined in XML schema.
derivationType	identifies the derivation type (i.e. extension or restriction) as defined in XML schema.
complexTypeType	The abstract type of all complex type definitions. Its subtypes are `namedComplexTypeType` (used in case of type globally defined) and `anonymousComplexTypeType` (used in case of type locally defined)
attributes	The list of attributes declared within the complex type. The list shall be transmitted in the order defined in 8.5.3. In case of derivation by restriction, the entire list of attributes shall be listed. In case of derivation by extension, only new attributes shall be listed.
attribute	a locally defined attribute (cf. 7.7.8.3.8).
attributeRef	a reference to a global attribute defined in a schema (cf. 7.7.8.3.8).
anyAttribute	indicates the use of the anyAttribute (cf. 7.7.8.3.8).
content	defines the content model of a complexType. (cf.7.7.8.3.9).
namedComplexTypeType	A globally defined complex type. Its name shall be present.
anonymousComplexTypeType	A locally defined complexType. an ID is associated to each anonymous type (cf. 7.7.8.3.7)
simpleTypeType	The abstract type of all simple type definitions. Its subtypes are `namedSimpleTypeType` (used in case of type globally defined) and `anonymousSimpleTypeType` (used in case of type locally defined).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

list	If the Simple type is defined as a list, this element contains a reference to the item type which constitutes the element of the list
union	If the simple type is defined as an union, this elements contains a reference to the different possible items of the union. The order of the elements has the same semantics than the defined in XML schema.
facet	conveys the facets of a simple type
namedSimpleTypeType	a globally defined simple type. Its name shall be present.
anonymousSimpleTypeType	a locally defined simpleType (or anonymous type). An ID shall be associated to each anonymous type (cf. 7.7.8.3.7)

7.7.8.3.6 Type and Element Referencement

7.7.8.3.6.1 Syntax

```
<xs:complexType name="typeReferenceType">
  <xs:choice>
    <xs:element name="namedTypeReference">
      <xs:complexType>
        <xs:attribute name="NamespaceID" type="NamespaceIDType"
              use="optional"/>
        <xs:attribute name="TypeID" type="TypeIDRefType"
          use="required"/>
      </xs:complexType>
    </xs:element>
    <xs:element name="anonymousTypeReference">
      <xs:complexType>
        <xs:attribute name="idref" type="AnonymousTypeIDRefType"
          use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:choice>
</xs:complexType>

<xs:complexType name="elementReferenceType">
  <xs:sequence>
    <xs:element name="namedElementReference">
      <xs:complexType>
        <xs:attribute name="NamespaceID" type="NamespaceIDType"
          use="optional"/>
        <xs:attribute name="ElementID" type="ElementIDRefType"
          use="required"/>
      </xs:complexType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="NamespaceIDType">
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>

<xs:simpleType name="TypeIDRefType">
  <xs:restriction base="xs:nonNegativeInteger"/>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

```
</xs:simpleType>

<xs:simpleType name="ElementIDRefType">
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>

<xs:simpleType name="AnonymousTypeIDType">
  <xs:restriction base="xs:nonNegativeInteger"/>
</xs:simpleType>

<xs:simpleType name="AnonymousTypeIDRefType">
  <xs:restriction base="AnonymousTypeIDType"/>
</xs:simpleType>
```

7.7.8.3.6.2 Semantics

Name	*Definition*
typeReferenceType	A reference to a type. This type is used when an element refers to a type or when a type refers to its super type. Two kind of types can be referenced: a named type (globally defined) or a anonymous type (locally defined).
namedTypeReference	The 'NamespaceID' attribute gives the index of the namespace, in the imported namespace table, in which the type is defined. The 'TypeID' attribute gives the index of the global type in the schema identified by the NamespaceID. If 'NamespaceID' attribute is not present, the namespace of the global type is the target namespace of the Schema Update.
anonymousTypeReference	The idref attribute gives the index in the table of anonymous type..
elementReferenceType	A reference to an element.The 'NamespaceID' attribute gives the index of the namespace, in the imported namespace table, in which the element is defined. The 'ElementID' attribute gives the index of the global element in the schema identified by the NamespaceID. If 'NamespaceID' attribute is not present, the namespace of the global element is the target namespace of the Schema Update.
NamespaceIDType	Defines the namespace id and refers to the table of imported namespace defined in the Schema Update Unit.
TypeIDRefType	Defines the type id and refers to the table of types carried in the Schema Update Unit.
ElementIDRefType	Defines the element id and refers to the table of types carried in the Schema Update Unit.
AnonymousTypeIDType	Defines the type id of an anonymous type. The scope of this id is limited to the current schema update unit. Therefore, in case of non complete transmission, Id values can be reused to identify different anonymous types.
AnonymousTypeIDRefType	Defines a reference to an anonymous type.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

7.7.8.3.7 Attribute definition

7.7.8.3.7.1 Syntax

```
<xs:complexType name="attributeDeclarationsType">
  <xs:sequence>
    <xs:element name="attribute" type="globalAttributeType"
      maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="attributeType" abstract="true">
  <xs:sequence>
    <xs:element name="typeReference" type="typeReferenceType"/>
  </xs:sequence>
  <xs:attribute name="name" type="nameStringType"  use="required"/>
  <xs:attribute name="defaultValue" type="xs:string" use="optional"/>
</xs:complexType>

<xs:complexType name="attributeRefType">
  <xs:attribute name="idref" type="xs:IDREF" use="required"/>
</xs:complexType>

<xs:complexType name="localAttributeType">
  <xs:complexContent>
    <xs:extension base="attributeType">
      <xs:attribute name="use" type="useType" use="required"/>
      <xs:attribute name="form" type="qualificationType" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="globalAttributeType">
  <xs:complexContent>
    <xs:extension base="attributeType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="anyAttributeType">
</xs:complexType>

<xs:simpleType name="useType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="optional"/>
    <xs:enumeration value="required"/>
  </xs:restriction>
</xs:simpleType>
```

7.7.8.3.7.2 Semantics

Name	*Definition*
attributeDeclarationsType	The list of all the global attribute declarations carried by this schema update. This list is ordered as specified in subclause 7.7.7.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

attributeType	An abstract type conveying the definition of an attribute. The type of the defined attribute is identified by a type reference. The name of the attribute shall be present. The defaultValue, if it exists, shall be encoded.
attributeRefType	A reference to a global attribute. It is used when a type references a global attribute.
localAttributeType	Defines the type of an attribute defined within a complex type. The `use` attribute shall be present. Its semantics is identical to the one defined in XML schema. The `form` attribute shall be instantiated, its semantics is identical to the one defined in XML schema.
globalAttributeType	Defines the type of a global attribute.
anyAttributeType	Indicates that any attribute of any schema can be present in the complexType.
useType	The type of the `use` attribute. It has two possible values : 'optional' or 'required'.

7.7.8.3.8 Content Model

7.7.8.3.8.1 Syntax

```
<xs:complexType name="contentModelType" abstract="true"/>

<xs:complexType name="emptyContentModelType">
  <xs:complexContent>
    <xs:extension base="contentModelType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="simpleContentModelType">
  <xs:complexContent>
    <xs:extension base="contentModelType">
      <xs:sequence>
        <xs:element name="simpleTypeReference"
          type="typeReferenceType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="complexContentModelType">
  <xs:complexContent>
    <xs:extension base="contentModelType">
      <xs:sequence>
        <xs:element name="particle" type="particleType"/>
      </xs:sequence>
      <xs:attribute name="mixed" type="xs:boolean" use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

7.7.8.3.8.2 Semantics

Name	*Definition*
contentModelType	This abstract type defines the content model of a complex type. It has three subtypes addressing the three content models defined by XML Schema: `emptyContentModelType`, `simpleContentModelType` and `complexContentModelType`.
emptyContentModelType	An empty content model.
simpleContentModelType	A simple content model. It includes a reference to the simple type which defines the content model of a complex type.
complexContentModelType	A complex content model. This model contains a particle (see XML schema) and its 'mixed' attribute shall be instantiated.

7.7.8.3.9 Facet Definition

7.7.8.3.9.1 Syntax

```
<xs:complexType name="facetType" abstract="true">
  <xs:attribute name="name" type="possibleFacet" use="required"/>
  <xs:attribute name="value" type="xs:string"/>
</xs:complexType>

<xs:simpleType name="possibleFacet">
  <xs:restriction base="xs:string">
    <xs:enumeration value="maxExclusive"/>
    <xs:enumeration value="minExclusive"/>
    <xs:enumeration value="minInclusive"/>
    <xs:enumeration value="maxInclusive"/>
    <xs:enumeration value="enumeration"/>
    <xs:enumeration value="length"/>
    <xs:enumeration value="minLength"/>
    <xs:enumeration value="maxLength"/>
  </xs:restriction>
</xs:simpleType>
```

7.7.8.3.9.2 Semantics

Name	*Definition*
facetType	Defines the possible facets associated to a simple type. A facet is composed of a name and a value. The facet mechanism is equivalent to the one defined in XML schema.
possibleFacet	The set of possible facets are limited to the ones used by BiM (see 8.5.4).

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.7.8.3.10 Particle Defintion

7.7.8.3.10.1 Syntax

```
<xs:complexType name="particleType" abstract="true">
  <xs:attribute name="minOccurs" type="xs:unsignedInt" default="1"/>
  <xs:attribute name="maxOccurs" type="occurrenceType" default="1"/>
</xs:complexType>

<xs:complexType name="anyParticleType">
  <xs:complexContent>
    <xs:extension base="particleType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="element">
  <xs:complexContent>
    <xs:extension base="particleType">
      <xs:sequence>
        <xs:element name="type" type="typeReferenceType"/>
      </xs:sequence>
      <xs:attribute name="name" type="nameStringType"/>
      <xs:attribute name="form" type="qualificationType"/>
      <xs:attribute name="nillable" type="xs:boolean"  use="required"/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="elementRef">
  <xs:complexContent>
    <xs:extension base="particleType">
      <xs:sequence>
        <xs:element name="ref" type="elementReferenceType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="modelGroupType" abstract="true">
  <xs:complexContent>
    <xs:extension base="particleType">
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="particle" type="particleType"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="sequence">
  <xs:complexContent>
    <xs:extension base="modelGroupType"/>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="all">
  <xs:complexContent>
    <xs:extension base="modelGroupType"/>
  </xs:complexContent>
```


Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

```
</xs:complexType>

<xs:complexType name="choice">
  <xs:complexContent>
    <xs:extension base="modelGroupType"/>
  </xs:complexContent>
</xs:complexType>

<xs:simpleType name="occurrenceType">
  <xs:union memberTypes="unboundedType xs:unsignedInt"/>
</xs:simpleType>

<xs:simpleType name="unboundedType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="unbounded"/>
  </xs:restriction>
</xs:simpleType>
```

7.7.8.3.10.2 Semantics

Name	*Definition*
particleType	An abstract type defining the type of all particles (cf. XML Schema). The range of occurrences are defined by the `minOccurs` and `maxOccurs` attributes.
anyParticleType	The any wildcard. This particle indicates that any global element of any namespace can be present in the description.
element	A local declaration of an element. This declaration is composed of a reference to a type, a name, a qualification form and a nillable property.
elementRef	A reference to an element globally defined.
modelGroupType	This particle is the super type of all groups: sequence, choice and all
sequence	A ‘sequence’ particle.
all	An ‘all’ particle.
choice	A ‘choice’ particle.
occurrenceType	The type for the `maxOccurs` attribute. Its value can be either an ‘xs:unsignedInt’ or the string value ‘unbounded’.

In subclause 8.1, replace the following sentence:

The binary fragment update payload syntax (`FUPayload`) is specified in subclause . It is composed of flags which define some decoding modes and a payload content which is either an `element` or a simple value (`simpleType`). …

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

by

The binary fragment update payload syntax (`FUPayload`) is specified in subclause 8.3. It is composed of flags which define some decoding modes and a payload content which is either an `element`, a simple value (`simpleType`) or a reference to a payload. …

In subclause 8.3.2.1, replace the DecodingModes syntax table by:

DecodingModes () {	**Number of bits**	**Mnemonic**
lengthCodingMode	2	bslbf
hasDeferredNodes	1	bslbf
hasTypeCasting	1	bslbf
hasNoFragmentReference	1	bslbf
ReservedBits	3	bslbf
}		

In subclause 8.3.2.2, add the following semantic after the `hasTypeCasting` semantic:

hasNoFragmentReference	A flag which specifies if this fragment update payload contains a fragment reference or the encoded fragment. This 1-bit flag can have the following values: — 0 – `hasNoFragmentReference` is equal to false, — 1 – `hasNoFragmentReference` is equal to true.

In subclause 8.4.1.1 replace the `Element` syntax table by:

Element (Enumeration SchemaModeStatus, SchemaComponent theType) {	**Number of bits**	**Mnemonic**
if (!hasNoFragmentReference) {		
FragmentReference()		
} else if (NumberOfSchemas >1) {		
if (SchemaModeStatus == "hot") {		
SchemaModeUpdate	1-3	vlclbf
}		
if (ElementContentDecodingMode == "mono"){		
Mono-VersionElementContent(ChildrenSchemaMode, theType)		
} else {		
Multiple-VersionElementContent(ChildrenSchemaMode, theType)		
}		
} else {		
Mono-VersionElementContent(“mono”, theType)		
}		
}		

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2005 – All rights reserved

In subclause 8.4.1.2, add the following semantic at the first position:

FragmentReference	See 8.4.8.

In subclause 8.4.3.3, replace the syntax element `SchemaIDOfSubstitution` by:

SchemaIDOfSubstitution	ceil(log2(`NumberOfSchemas + NumberOfAdditionalSchemas`))	bslbf

In subclause 8.4.3.4, replace the semantic of `SchemaIDOfSubstitution` by:

SchemaIDOfSubstitution	The version identifier which refers to the schema where the substitute element is defined. Its value is the index of the URI in the `SchemaURI` array defined in 7.2 (optionally extended with the list of additional schemas).

In subclause 8.4.4.1, replace the syntax element of `SchemaID` by:

SchemaID	ceil(log2(`NumberOfSchemas + NumberOfAdditionalSchemas`))	bslbf

In subclause 8.4.4.2, replace the semantic of `SchemaID` by:

SchemaID	Identifies the schema which is needed to decode this `ElementContentChunk`. Its value is the index of the URI in the `SchemaURI` array defined in 7.2 (optionally extended with the list of additional schemas).

In subclause 8.4.7.1, replace the syntax table of `SimpleType` by:

SimpleType(SchemaComponent theType) {	**Number of bits**	**Mnemonic**
If (! AdvancedOptimisedDecodersFlag) {		
if (useOptimisedDecoder(theType)) {		
optimisedDecoder(theType)		
} else {		
defaultDecoder(theType)		
}		
} else {		
if (numOfMappedOptimisedDecoder(theType) !=0) {		
optimisedDecoderID	ceil(log2(number of decoders associated to this type))	blsbf
advancedOptimisedDecoder(theType, optimisedDecoderID)		

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

} else {		
defaultDecoder(theType)		
}		
}		

In subclause 8.4.7.2, add the following text at the beginning of the subclause:

A simple type can be associated to a single default simple type decoder, a single simple optimised decoder and one or several optimised decoders. This syntax table describes the process to select the proper simple type decoder for each simple type to be decoded.

In subclause 8.4.7.2, add the following semantic after the `useOptimisedDecoder` *semantic:*

numOfMappedOptimisedDecoder ()	Returns the number of advanced optimised decoders associated to the type `theType` as described in Clause 9.

In subclause 8.4.7.2, replace the `optimisedDecoder` *semantic by:*

optimisedDecoder ()	Triggers the simple optimised type decoder associated to the type `effectiveType` as conveyed in the `DecoderInit` (refer to subclause 7.2).

In subclause 8.4.7.2, add the following semantic after the `defaultDecoder` *semantic:*

advancedOptimisedDecoder(aType, anInteger)	Triggers the advanced optimised decoder identified by the `optimisedDecoderID` field which is associated to the type `theType` in the optimised decoder mapping.

Add the following subclause (subclause 8.4.8):

8.4.8 FragmentReference

8.4.8.1 Syntax

FragmentReference () {	**Number of bits**	**Mnemonic**
isDeferred	1	bslbf
hasSpecificFragmentReferenceFormat	1	bslbf
ReservedBits	6	bslbf
If (hasSpecificFragmentReferenceFormat == '1') {		
FragmentReferenceFormat	8	bslbf
} else {		
FragmentReferenceFormat = SupportedFragmentReferenceFormat[0]		
}		

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

FragmentRefLength	8+	vluimsbf8
/** FragmentRef **/		
if(FragmentReferenceFormat == "0x01") {		
URIFragmentReference()		
} else {		
ReservedBits	FragmentRef Length	
}		
/** Fragment ref end **/		
}		

8.4.8.2 Semantics

Name	*Definition*
isDeferred	A flag which signals if the fragment is deferred, and therefore can be acquired later on by the terminal. If the `isDeferred` has a value of "0", the decoder shall resolve the reference and obtain the fragment payload according to the process described in subclause 5.8.1. If the `isDeferred` has a value of "1", then the decoded value of the fragment shall include a deferred reference containing the specified fragment reference as specified in subclause 5.8.2. A node that is represented by a deferred fragment reference shall be treated by the decoder as a deferred node.
hasSpecificFragmentReferenceFormat	A flag which signals if the fragment reference format is different from the default one defined in the `DecoderInit` (see subclause 7.2): — If `hasSpecificFragmentReferenceFormat` has a value of '1', the decoder shall use the `FragmentReferenceFormat` field as indication of the fragment reference format. — If `hasSpecificFragmentReferenceFormat` has a value of '0' then the decoder shall take the fragment reference format to be that defined within the `DecoderInit` i.e. the default setting.
FragmentReferenceFormat	An 8-bits value which uniquely identifies the format of the contained fragment reference as defined in Table AMD1-1. This field shall only be present when `hasSpecificFragmentReferenceFormat` has a value of '1'.
FragmentRefLength	The number of bits that follows this field, which denotes the size of the `FragmentRef` field.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Add the following subclause (subclause 8.4.9):

8.4.9 URIFragmentReference

8.4.9.1 Syntax

URIFragmentReference () {	**Number of bits**	**Mnemonic**
href	FragmentRefLength	bslbf
}		

8.4.9.2 Semantics

Name	*Definition*
href	Defines the URI of the URI fragment reference in UTF-8 format. The field is of variable length and its actual length shall be inferred from the `FragmentRefLength` field defined within the `FragmentReference` (see subclause 8.4.8).

In subclause 8.5.2.1, add the following definition after the element transitions definition:

— *Wildcard transitions:* Wildcard transitions, when crossed, specify to the decoder that an undefined element is present in the description.

In subclause 8.5.2.1, add the following definition after the shunt transitions definition:

— *Mixed transitions*: Mixed transitions are a special kind of code transitions. Their binary code value is always equal to 1. Mixed transitions, when crossed, specify to the decoder that a string is present between the previous decoded element and the next one.

In subclause 8.5.2.2.3.1, replace the following text:

A syntax tree associated to the complex type is generated based on the “reference-free effective content particle” generated in phase 1. The syntax tree associated to the type is composed of different syntax tree nodes: element declaration nodes, group nodes and occurrence nodes. Element declaration nodes associate an element name to its type. They are leaves of the syntax tree and are derived from element declaration particles. Group nodes define a composition group (sequence, choice or all) and are derived from group particles. A group node contains only occurrence nodes. Occurrence nodes are derived from the ‘min occurs’ and ‘max occurs’ property of the particle and contain group nodes or element declaration nodes.

by

A syntax tree associated to the complex type is generated based on the “reference-free effective content particle” generated in phase 1. The syntax tree associated to the type is composed of different syntax tree nodes: element declaration nodes, wildcard nodes, group nodes and occurrence nodes. Element declaration nodes associate an element name to its type. Wildcard nodes represent element wildcards. Both element declaration and wildcard nodes are leaves of the syntax tree and are derived respectively from their element declaration and wildcard particles. Group nodes define a composition group (sequence, choice or all) and are

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

derived from group particles. A group node contains only occurrence nodes. Occurrence nodes are derived from the 'min occurs' and 'max occurs' property of the particle and contain group nodes, element declaration or wildcard nodes.

In subclause 8.5.2.2.4, replace the following text:

Syntax tree normalization gives a unique name to every element declaration node and group node of the syntax tree. It allows to order the sibling nodes and assign a binary code to them. This code is used during the automata construction phase.

by

Syntax tree normalization gives a unique name to every element declaration node, wildcard node and group node of the syntax tree. It allows to order the sibling nodes and assign a binary code to them. This code is used during the automata construction phase.

In subclause 8.5.2.2.4, add the following text after the 3rd bullet (i.e. "Element declaration node signatures are equal to the expanded name of the element"):

- A wildcard node signature is constructed from the wildcard schema component it is associated to (see XML Schema – Part 1, Chapter 3.10.1) by the concatenation, whitespace separated, of:
  - the ":wildcard" key word
  - the "process contents" property of the wildcard schema component preceded by the character ':' (i.e. ":skip", ":strict" or ":lax"),
  - the "namespace constraint" property represented by a set of whitespace separated keywords, where
    - the 'any', 'not' and 'absent' values are respectively identified by the ":any", ":not", ":absent" keywords,
    - the ":any" or ":not" keywords are always first,
    - the namespaces and, if present the ":absent" keyword, are alphabetically sorted.

In subclause 8.5.2.2.4, add the following text at the end of the subclause:

Wildcard signature example

Given the following wildcards defined in a schema which target namespace is "http://www.mpeg7.org/example":

```
<xs:any processContents="skip"/>
<xs:any namespace="##other" processContents="lax"/>
<xs:any namespace="urn:example:namespaceB urn:example:namespaceA"/>
<xs:any namespace="##targetNamespace"/>
<xs:any namespace="##local"/>
```

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The signatures of these wildcards are respectively

:wildcard :skip :any

:wildcard :lax :not http://www.mpeg7.org/example

:wildcard :strict urn:example:namespaceA urn:example:namespaceB

:wildcard :strict http://www.mpeg7.org/example

:wildcard :strict :absent

In subclause 8.5.2.2.5.1, replace the following text:

— The complex content automaton of the complex type to decode is the node automaton produced by the root node of its syntax tree

by

— The complex content automaton of the complex type to decode is the node automaton produced by the root node of its syntax tree modified according to the following rule if the content type of the complex type is 'mixed':

  — a new final state is created. The old final state is linked with the new final state by two transitions, a shunt transition and a mixed transition.

In subclause 8.5.2.2.5.2, replace the following text:

— An automaton for an element declaration node is composed of two states, a start state and a final state, and a transition between them. It is used to specify the “element name” / “type” association declared in the complex type definition. The transition is an "element transition" to which the element name of the element declaration node is associated. The target state of the transition is a "type state" to which the element type of the element declaration node is

by

An automaton for an element declaration node can have two different forms depending on the content type of the complexType in which the element is declared:

— If the content type of the complexType is not 'mixed' the automaton is composed of two states, a start state and a final state, and a transition between them. It is used to specify the “element name” / “type” association declared in the complex type definition. The transition is an "element transition" to which the element name of the element declaration node is associated. The target state of the transition is a "type state" to which the element type of the element declaration node is associated.

— If the content type of the complexType is 'mixed', the automaton is composed of three states, a start state, an intermediate and a final one. The start state is linked with the intermediate state by two transitions, a shunt transition and a mixed transition. The intermediate state is linked with the final state by an "element transition" to which the element name of the element declaration node is associated. The final state of the transition is a "type state" to which the element type of the element declaration node is associated.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

In subclause 8.5.2.2.5.2, add the following figures after Figure 22:

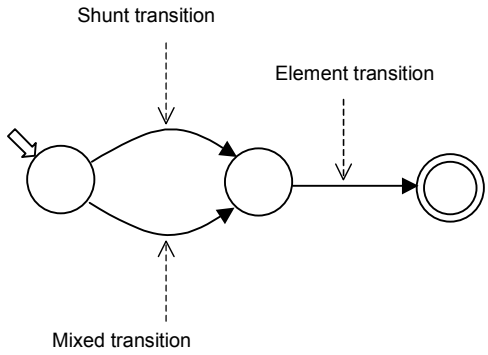


Figure AMD1-10 - Example of an element declaration node automaton in case of mixed content model

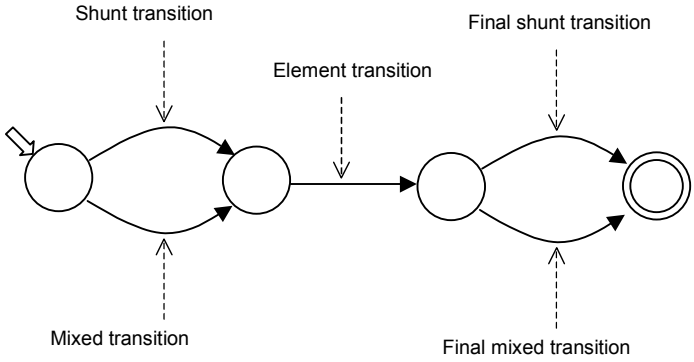


Figure AMD1-11 - Example of a complete complex content automaton in case of mixed content model

In subclause 8.5.2.2.5.2, add the following text at the end of the subclause:

Wildcard node automaton

The wildcard node automaton can have two different forms depending on the content type of the complexType in which the element is declared:

— If the content type of the complexType is not 'mixed' the automaton is composed of two states, a start state and a final state, and a wildcard transition between them.

— If the content type of the complexType is 'mixed', the automaton is composed of three states, a start state, an intermediate and a final one. The start state is linked with the intermediate state by two transitions, a shunt transition and a mixed transition. The intermediate state is linked with the final state by a wildcard transition.

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Add the following subclause (subclause 8.5.2.4.4):

8.5.2.4.4 Mixed transition behavior

Mixed transitions are used to model the decoding of the strings present between two elements contained in a element of 'mixed' content type. When a mixed transition is crossed by the token, a string shall be decoded according to syntax defined in 8.4.7 where the parameters `theType` is set to the XML schema “string” datatype.

Add the following subclause (subclause 8.5.2.4.5):

8.5.2.4.5 Wildcard transition behaviour

8.5.2.4.5.1 Introduction

Wildcard transitions, when crossed, specify to the decoder that an element not known a priori is present in the description. The decoder shall execute the following decoding procedure.

8.5.2.4.5.2 AnyElementDecoding Syntax

AnyElementDecoding() {	**Number of bits**	**Mnemonic**
GlobalElementSchemaID	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
SBC_Operand_Selector	ceil(log2(number_of_global elements in the schema identified by GlobalElementSchemaID))	bslbf
if (inPayloadDecoding()) {		
Element(ChildrenSchemaMode, theAnyType)		
}		
}		

8.5.2.4.5.3 AnyElementDecoding Semantics

Name	*Definition*
GlobalElementSchemaID	The schema in which the global element is defined. Its value is the index of the URI in the `SchemaURI` array defined in 7.2 (optionally extended with the list of additional schemas).
SBC_Operand_Selector	Selects one global element of the schema referenced by `GlobalElementSchemaID` using the `OperandTBC` table as specified in 7.6.5.2.3.
inPayloadDecoding()	returns true if the AnyElementDecoding procedure has been triggered from a payload decoding procedure.
Element()	see subclause 8.4.1

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

theAnyType	The type of the element identified by the `SBC_GlobalElement_SelectorCode` as defined in the schema identified by the `GlobalElementSchemaID`.

Replace subclause 8.5.3 content by the following subclauses 8.5.3.1, 8.5.3.2 and 8.5.3.3:

8.5.3.1 Attributes realization

The attributes of an element are decoded with the following rules:

— All the attributes definitions are collected from the super types of the complex type,

— All the attribute definitions defined as FIXED in the schema are suppressed,

— If existing, the “anyAttribute” definition is suppressed,

— All the attribute definitions are lexicographically ordered using their expanded name.

A sequence automaton is generated and used to decode the attributes. As a result, attributes are decoded by set of consecutive patterns. These patterns are composed of two components:

— an attribute flag, which defines the presence or the absence of an optional attribute,

— an attribute value decoded as defined in subclause 8.4.7.

The 1-bit attribute flag is only present for optional attributes. It is equal to 0 if the attribute is not coded or 1 if the attribute is coded. It is not present for mandatory attributes.

If an “anyAttribute” was present (and suppressed), the `AnyAttributeDecoding` procedure (see 8.5.3.2) shall be triggered after the decoding of all other attributes.

8.5.3.2 AnyAttribute Decoding

8.5.3.2.1 Syntax

AnyAttributeDecoding() {	**Number of bits**	**Mnemonic**
AnyAttributePresence	1	bslbf
If (AnyAttributePresenceFlag) {		
NumberOfAnyAttributes	5+	vluimsbf5
for (i=0;i< NumberOfAnyAttributes;i++) {		
SingleAnyAttributeDecoding()		
}		
}		
}		

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.5.3.2.2 Semantics

Name	*Definition*
AnyAttributePresence	indicates the presence of at least one attribute matched by the anyAttribute wildcard.
NumberOfAnyAttributes	indicates the number of attributes matched by the anyAttribute wildcard.

8.5.3.3 Single AnyAttribute Decoding

8.5.3.3.1 Syntax

SingleAnyAttributeDecoding() {	**Number of bits**	**Mnemonic**
GlobalAttributeSchemaID	ceil(log2(`NumberOfSchemas` + `NumberOfAdditionalSchemas`))	uimsbf
GlobalAttributeID	ceil(log2(number of attributes defined in schema `SchemaID`))	
if (inPayloadDecoding()) {		
SimpleType(theGlobalAttributeType)		
}		
}		

8.5.3.3.2 Semantics

Name	*Definition*
GlobalAttributeSchemaID	Identifies the schema in which the global attribute is defined. Its value is the index of the URI in the `SchemaURI` array defined in 7.2 (optionally extended with the list of additional schemas).
GlobalAttributeID	Identifies the global attribute amongst all the attributes defined in `GlobalAttributeSchemaID`. Global attributes code words are assigned in lexicographical order of their expanded names.
inPayloadDecoding()	returns true if the `AnyElementDecoding` procedure has been triggered from a payload decoding procedure.
SimpleType()	see subclause 8.4.7.
theGlobalAttributeType	The type of the global attribute identified by the `GlobalAttributeID` defined in the schema identified by the `GlobalAttributeSchemaID`.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

Add the following clause (Clause 9):

9 Advanced optimised decoders

9.1 Overview

The simple types of an XML document are decoded by a set of datatype decoders which can be the default ones, simple or advanced optimised decoders. This subclause describes the advanced optimised decoder decoding mechanisms, more specifically:

— How the advanced optimised decoders are initialized and mapped to data types;

— How the decoder should manage the set of active advanced optimised decoders;

— How the advanced optimised decoders should behave through a logical interface.

9.2 Decoder behaviour

9.2.1 Advanced optimised decoder mapping and parameters

During the decoding of a BiM bitstream, the decoder maintains 3 tables :

— the `AdvancedOptimisedDecoderTypeURI[]` which contains all the advanced optimised decoders that can be instantiated by the decoder. This table is initialised at decoder initialisation phase and remain identical for the entire binary description stream.

— the `AdvancedOptimisedDecoderInstances[]` which contains all instances of advanced optimised decoders currently available for decoding. This table can be reinitialised before each fragment update unit according to the value of the `OptimisedDecodersReparameterization` field defined in subclause 7.4.2.

— The type-decoder mapping table where all schema types are associated to one or several optimised decoders. This table can be reinitialised before each fragment update unit according to the value of the `OptimisedDecodersReparameterization` field defined in subclause 7.4.2.

Both the `AdvancedOptimisedDecoderInstances` and the type-decoder mapping table can be redefined during binary description stream life time. The default set of advanced optimised decoder instances and mapping is defined by the `DecoderInit`. The decoder can switch to this initial configuration or define new one before the decoding of each fragment update unit as defined in subclause 7.4.2.

Advanced optimised decoder type ID	Optimised decoder type
0	Mpeg7:ZLib
1	Mpeg7:UniformQuantizer

Figure AMD1-12 - Example of the `AdvancedOptimisedDecoderTypeURI` table

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Advanced optimised decoder instance ID	Advanced optimised decoder instance type	Advanced optimised decoder instance parameters
0	0 (Mpeg7:ZLib)	none
1	1 (Mpeg7:UniformQuantizer)	(0,1,8)
2	1 (Mpeg7:UniformQuantizer)	(-1,1,16)

Figure AMD1-13 - Example of the `AdvancedOptimisedDecoderInstances` table

Type identification			Advanced optimised decoder instance ID
Schema ID	Type ID	*equiv type name*	
1	1	*xsd:string*	0
1	2	*mpeg7:floatVector*	1, 2
1	3	*xsd:float*	default, 2
2	…	…	…

Figure AMD1-14 - Example of the type-decoder mapping table

9.2.2 Logical interface of an advanced optimised decoder

An advanced optimised decoder shall support the following operations:

Methods	*Definition*
readParameters()	Reads the optimised decoder parameters in the bitstream.
decode()	Decodes a value.

In addition, a contextual advanced optimised decoder shall support the following operation:

Methods	*Definition*
reset()	Reset the optimised decoder if it is contextual. This operation shall be performed after the decoding of each fragment update unit, before the decoding process enters a skippable subtree and after it exits a skippable subtree.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3 Advanced Optimised Decoder Initialization

9.3.1 Syntax

AdvancedOptimisedDecodersConfig () {		
/** Definition of the optimised decoder instances table **/		
NumOfAdvancedOptimisedDecoderInstances	8+	vluimsbf8
for (i=0; i< NumOfAdvancedOptimisedDecoderInstances; i++) {		
AdvancedOptimisedDecoderInstantiationLength	8+	
/** AdvancedOptimisedDecoderInstantiation **/		
AdvancedOptimisedDecoderInstances[i].Type	*ceil(log2(NumOfAdvancedOptimisedDecoders))*	bslbf
AdvancedOptimisedDecoderInstances[i].Params()		
nextByteBoundary()		
/** AdvancedOptimisedDecoderInstantiation end**/		
}		
/** definition of the type-decoder mappings **/		
NumOfMappings	8+	vluimsbf8
for (k=0; k< NumOfMappings;k++) {		
/** the advanced optimized decoders in the mapping **/		
PreserveDefaultDecoderInMapping	1	bslbf
Reserved	7	bslbf
NumOfAdvancedOptimisedDecodersInMapping[k]	8+	vluimsbf8
for (i=0; i< NumOfOptimisedDecodersInMapping [k]; i++) {		
AdvancedOptimisedDecoderInstanceID[k][i]	ceil(log2(NumOfAdvancedOptimisedDecoderInstances))	uimsbf
}		
/** the types on which the optimised decoders are mapped **/		
NumOfTypesInMapping[k]	8+	vluimsbf8
for (i=0; i< NumOfTypesInMapping[k]; i++) {		
SchemaID	ceil(log2 (NumberOfSchemas + NumberOfAdditionalSchemas))	uimsbf
TypeIdentificationCode[k][i]	8+	vluimsbf8
}		
}		
}		

© ISO/IEC 2005 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.3.2 Semantics

Name	*Definition*
NumOfAdvancedOptimisedDecoderInstances	Indicates the total number of optimised decoder instances present in the optimised decoder instances table.
AdvancedOptimisedDecoderInstantiationLength	Indicates the length in bytes of the decoder instantiation excluding the length.
AdvancedOptimisedDecoderInstances[i].Type	Indicates the advanced optimised decoder type that is instantiated. It refers to a type identified in the `AdvancedOptimisedDecoderTypeURI` table.
AdvancedOptimisedDecoderInstances[i].Params	Parameters associated to a optimised decoder. This field is dependent of the optimised decoder and can have a null size.
NumOfMappings	Indicates the number of mappings between sets of types and sets of optimised decoders.
PreserveDefaultDecoderInMapping	Indicates that the the types contained in the mapping referred by the index k can be decoded by their default decoder. This default decoder is always referred by the index 0 in the set of optimised decoders associated to each data type.
NumOfAdvancedOptimisedDecodersInMapping [k]	Indicates the number of optimised decoders that are associated to teh data types contained in the mapping referred by the index k.
AdvancedOptimisedDecoderInstanceID[k][j]	Indicates the instance to be added to the mapping referred by the index k. It is an index to the `AdvancedOptimisedDecoderInstances[]` table.
NumOfTypesInMapping [k]	Indicates the number of data types which accept the optimised decoders defined in the mapping.
SchemaID	Identifies the schema from the list of `schemaURI`s transmitted in the `DecoderInit (`optionally extended by a list of additional schemas) in which the type subject to the mapping is defined.
TypeIdentificationCode[k][j]	Selects one data type from the set of all data types contained in the schema with index `SchemaIndex`. The syntax and semantics of `TypeIdentificationCode[k][j]` is the same as of the type identification code defined in subclause 7.6.5.4 except that here it is represented using vluimsbf8. The `TypeIdentificationCode[k][j]` assumes the “anyType” as base type. The advanced optimised decoder instances present in the mapping are associated to this type and all the subtypes of this type defined in the schemaID. For instance if the type "string" is mapped with an optimised decoder, all inherited types from string will be bound to the same instance of the optimised decoder. In order to bind an inherited subtype with another optimised decoder, this has to be made explicit in another mapping. Once again, all subtypes of this subtype, will refer to the latter mapping.

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.4 Advanced Optimised Decoder Classification scheme

The advanced optimised decoders defined in this specification are uniquely identified by the following classification scheme:

```
<ClassificationScheme
      uri="urn:mpeg:mpeg7:systems:SystemsAdvancedOptimisedDecodersCS:2004">
   <Term termID="1">
      <Name xml:lang="en">Zlib</Name>
      <Definition xml:lang="en">A contextual advanced optimised decoder
dedicated to the compression of strings</Definition>
   </Term>
   <Term termID="2">
      <Name xml:lang="en">UniformQuantizer</Name>
      <Definition xml:lang="en">An advanced optimised decoder to compress large
series of floating point numbers using uniform quantization
technique</Definition>
   </Term>
   <Term termID="3">
      <Name xml:lang="en">NonUniformQuantizer</Name>
      <Definition xml:lang="en"> An advanced optimised decoder to compress large
series of floating point numbers using non-uniform quantization technique
</Definition>
   </Term>
</ClassificationScheme>
```

9.5 UniformQuantizer advanced optimised decoder

9.5.1 Overview of scalar quantization (informative)

A scalar quantization is a mathematical operation which maps a set of consecutive, non overlapping intervals of real values (a partition) to a set of integers (the labels) where each interval is labeled with a unique number. Therefore a quantization operation is always associated with loss of data. For this reason the quantization operation is also referred to as a kind of lossy compression in contrast to lossless compression such as e.g. zip, run length coding, Huffmann codes, and so on. However, this loss can be made arbitrarily small by choosing an appropriate number of bits to encode the integer value (representing the interval label) and by the choice of the characteristics of the quantizier. Depending on the mapping function, two types of quantizers are distinguished, namely uniform and non-uniform quantizers. In fact uniform quantizers can be considered as a special case of non-uniform quantizers.

The mapping is the process of finding the quantization interval into which the real value to be quantized fits. This interval label is then encoded with a certain limited number of bits. The reverse operation, called inverse quantization or reconstruction, restitutes a real value according to the interval label. These processes are controlled by two parameter sets: the first gives the boundaries of the non-overlapping intervals (a partition) and the second one the reconstruction values, called output levels. In order to yield minimal quantization error the probability density function (PDF) of the signal to be quantized must either be known or must be estimated based on a representative set of data. With uniform quantizers the error becomes minimal only if the input signal exhibits an even probability distribution over the numerical range to be covered.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.5.2 Decoders parameters

9.5.2.1 syntax

UniformQuantizerParameters() {	**Number of bits**	**Mnemonic**
sgn(v_{min})	1	bslbf
abs(v_{min})	5+	vluimsbf5
sgn(v_{max})	1	bslbf
abs(v_{max})	5+	vluimsbf5
nbits-1	5	uimsbf
}		

9.5.2.2 Semantics

Name	*Definition*
sgn(v_{min})	Sign of v_{min}. If v_{min} is positive ; 0, else 1.
abs(v_{min})	Absolute value of v_{min}.
sgn(v_{max})	Sign of v_{max}. If v_{max} is positive ; 0, else 1.
abs(v_{max})	Absolute value of v_{max}.
nbits	Number of bits used to encode quantized numbers. `nbits` can take the values from 1 to 32. But the field `nbits`-1 will be written.

9.5.3 Decoding process

9.5.3.1 Syntax

UniformQuantizerDecode(parameters) {	**Number of bits**	**Mnemonic**
vq	`nbits`	uimsbf
}		

9.5.3.1.1 Semantics

Name	*Definition*
vq	An `nbits` bits field defining the quantized value v_q. The restituted decoded value $\overline{v}$ is : $$\overline{v}=v_{\min}+v_q\frac{v_{\max}-v_{\min}}{2^{nbits}-1}$$

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2005 – All rights reserved
Not for Resale

9.6 NonUniformQuantizer optimized decoder

9.6.1 Overview of scalar quantization (informative)

A scalar quantization is a mathematical operation which maps a set of consecutive, non overlapping intervals of real values (a partition) to a set of integers (the labels) where each interval is labelled with a unique number. Therefore a quantization operation is always associated with loss of data. For this reason the quantization operation is also referred to as a lossy compression in contrast to lossless compression such as e.g. zip, run length coding, Huffmann codes, and so on. However, this loss can be made arbitrarily small by choosing an appropriate number of bits to encode the integer value (representing the interval label) and by the choice of the characteristics of the quantizer. Depending on this characteristics, two types of quantizers are distinguished, namely uniform and non-uniform quantizers. In fact, uniform quantizers can be considered as a special case of non-uniform quantizers.

The forward quantization process consists of finding the quantization interval into which the real value to be quantized fits. This interval label is then encoded with a certain limited number of bits. The reverse operation, called inverse quantization or reconstruction, restitutes a real value according to the interval label. These processes are controlled by two parameter sets: the first gives the boundaries of the non-overlapping intervals (partitions) and the second one the reconstruction values, called output levels. In order to yield minimal quantization error, the probability density function (PDF) of the signal to be quantized must either be known or estimated from a representative set of data.

The general scheme for the non-uniform quantization is defined by the explicit interval boundaries and the corresponding output levels. The algorithm for the quantization part comprises of finding the interval in which the input value resides. The reconstruction is performed through a table lookup where the output level is retrieved from the quantized value which serves as a table index. It is known that this form of quantization yields the minimum error when the sets of parameters (interval borders and output levels) are derived from the signal's PDF. One algorithm for finding such parameter sets (characteristic) is the well known Lloyd-Max quantizer. If the signal to be quantized is a sequence of vectors (e.g. containing data derived from different frequency bands in each vector component), each vector component may have different statistical properties and thus require different quantizer characteristics for optimum representation.

9.6.2 Decoder parameters

9.6.2.1 Syntax

NonUniformQuantizerParameters() {	**Number of bits**	**Mnemonic**
nBitsData-1	5+	vluimsbf5
nCharacteristics	5	uimsbf
/** uniformQuantizerParameters ****/**		
sgn(vmin)	1	bslbf
abs(vmin)	5+	vluimsbf5
sgn(vmax)	1	bslbf
abs(vmax)	5+	vluimsbf5
nbitsCharacteristics-1	5	uimsbf
/** uniformQuantizerParameters end **/		
for (j=0; j<nCharacteristics ; j++) {		
for (k=0; k<2^nbitsData ; k++) {		
quantMatrix[j][k]	nbitsCharacteristics	uimsbf
}		
}		
}		

--`,,```,,,,````-`-,,`,,`,`,,`---

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.6.2.2 Semantics

Name	*Definition*
nBitsData	Indicates the number of bits used for the encoding of the data.
nCharacteristics	Indicates the total number of individual quantizer characteristics that are defined for representing the data. This number is usually chosen as 1 (for a series of simple scalar values) or as the number of vector components (for a series of vectors).
uniformQuantizerParameters	For each of the quantizer characteristics, the corresponding set of output levels is transmitted by means of a uniformly quantized representation. The uniformQuantizerParameters hold the definition information for this representation.
sgn(v_{min})	Sign of v_{min}. If v_{min} is positive ; 0, else 1.
abs(v_{min})	Absolute value of v_{min}.
sgn(v_{max})	Sign of v_{max}. If v_{max} is positive ; 0, else 1.
abs(v_{max})	Absolute value of v_{max}.
nBitsCharacteristics	Number of bits used to encode quantized numbers.
quantMatrix[j][k]	This Matrix holds the output levels (index k) for each of the quantizer characteristics (index j) for the binary payload data.

9.6.3 Decoding process

9.6.3.1 Syntax

NonUniformQuantizerDecode(parameters) {	*Number of bits*	*Mnemonic*
vqIndex	nBitsData	uimsbf
}		

9.6.3.2 Semantics

Name	*Definition*
vqIndex	The restituted decoded value $\overline{v}$ is computed in the following way: $$\overline{v} = v_{\min} + v_q \frac{v_{\max} - v_{\min}}{2^{nbits} - 1}$$ Where v_q = quantMatrix[c][vqIndex] The quantizer characteristics c used to represent / reconstruct each transmitted value is determined by the order of the transmitted values. The first value uses characteristics c=0, the second value c=1 and so forth, incrementing by one and falling back to 0, once c=nCharacteristics has been reached (i.e. "increment by one, modulo nCharacteristics).

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

9.7 Zlib advanced optimised decoder

9.7.1 Overview

The ZLib decoder is a contextual optimised decoder which doesn't have any specific parameters. It uses the Zlib encoding scheme as defined in RFC 1950.

9.7.2 Decoding process

9.7.2.1 Syntax

String ZLibDecoder() {	**Number of bits**	**Mnemonic**
resultString=""		
tempChar = GiveNextCharInBuffer();		
while (tempChar != separatorChar){		
resultString = concat(resultString, tempChar)		
tempChar = GiveNextCharInBuffer()		
}		
return resultString		
}		

Char GiveNextCharInBuffer() {	**Number of bits**	**Mnemonic**
If isEmpty (charsBuffer){		
ZLibChunkLength	8+	vluimsbf8
ZLibChunk	8* ZLibChunkLength	bslbf
charsBuffer = inflate(ZLibChunk)		
}		
return nextChar(charsBuffer)		
}		

9.7.2.2 Semantics

If the ZLib codec is asked to deliver a string to the BiM framework, either:

— the current ZLib buffer is empty : the ZLib decoder reads a compressed chunk in the bitstream, and decompresses it into the current ZLib buffer,

— the buffer is not empty : the ZLib decoders delivers the first string of the buffer.

Strings are separated by a separatorChar. If a string is encoded over two or more chunks, the decoder decompresses all the needed chunks and returns the concatenation of all characters extracted from the buffers until the separator character.

In order to obtain the next string, the decoder reads the charsBuffer until it gets a separatorChar. If the charsBuffer is totally consumed before reaching a separatorChar, the charsBuffer is refilled by decompressing the ZLib compressed chunk available in the bitstream.

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Name	*Definition*
resultString	This is a field representing the string expected.
tempChar	This is a field representing the character read in the charsBuffer.
separatorChar	This is a constant representing the separatorChar. It is equal to 0x00.
ZLibChunkLength	Indicates the size in bytes of the ZLibChunk. A value of zero is forbidden.
ZLibChunk	This is an UTF-8 representation of a compressed list of characters. The compression algorithm used is ZLib, in default compression mode.
inflate(buffer)	This function is a part of the ZLib API. It decompresses a ZLib buffer.
nextChar(charsBuffer)	This function returns the first character of the charsBuffer and removes it from the charsBuffer.
isEmpty (charsBuffer)	This function returns true if the charsBuffer is empty, otherwise false.
concat(aString, aCharacter)	This function returns the concatenation of aString and aCharacter.
charsBuffer	This is a local buffer of decompressed characters. It contains a list of strings separated by a separatorChar.

9.7.3 Encoding process (informative)

At the encoding phase, the fixed-size buffer is fed with strings. When the buffer is full, it is compressed and the resulting compressed chunk of data is placed in the expected position of the first string compressed in this buffer. Figure AMD1-15 represents a regular BiM binary stream, without the ZLib codec. Strings (in gray) are dispatched along the entire bitstream. Figure AMD1-16 represents only the two phases of the ZLib codec encoding process : strings are first gathered on the fly into a fixed-size buffer, and then, this buffer is compressed by the ZLib algorithm, into a compressed chunk, which may contains several strings. The location of the compressed chunk ensures that during the decoding process, when a string is required, either the ZLib codecs will read a compressed chunk, decompress it and delivers the string from the decompressed buffer or if its decompressed buffer isn't empty, it delivers the string from its decompressed buffer.

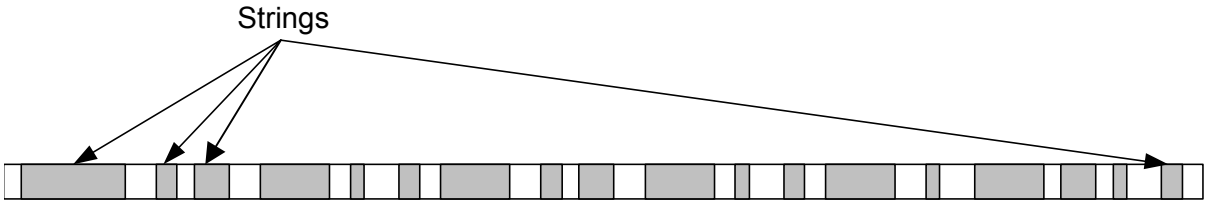


Figure AMD1-15 - Regular BiM bitstream

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

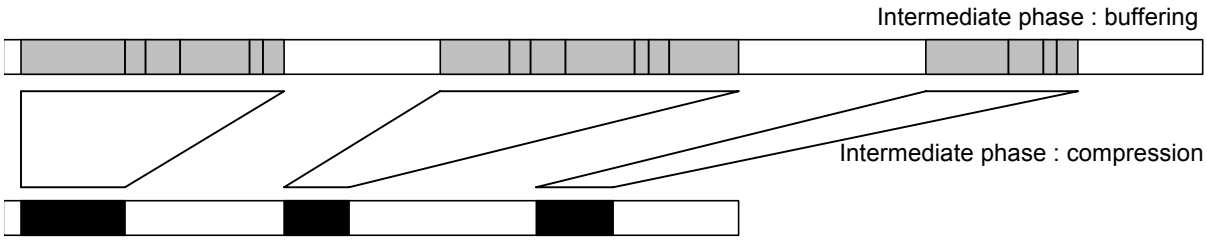


Figure AMD1-16 - BiM bitstream using Zlib encoded value

In the bibliography annex, add the following reference:

ZLIB Compressed Data Format Specification Version 3.3, RFC 1950, P. Deutsch, J. Gailly, May 1996, http://www.ietf.org/rfc/rfc1950.txt

© ISO/IEC 2005 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 79 pages

© ISO/IEC 2005 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale