INTERNATIONAL STANDARD

ISO/IEC 21000-7

First edition
2004-10-15

AMENDMENT 2
2007-07-01

Information technology — Multimedia framework (MPEG-21) —

Part 7:
Digital Item Adaptation

AMENDMENT 2: Dynamic and Distributed Adaptation

Technologies de l'information — Cadre multimédia (MPEG-21) —

Partie 7: Adaptation d'article numérique

AMENDEMENT 2: Adaptations dynamique et distribuée

Reference number
ISO/IEC 21000-7:2004/Amd.2:2007(E)

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 2 to ISO/IEC 21000-7:2004 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Information technology — Multimedia framework (MPEG-21) —

Part 7: Digital Item Adaptation

AMENDMENT 2: Dynamic and Distributed Adaptation

In subclause 1.4, replace the fourth list item with the following:

— *Bitstream Syntax Description* tools comprise the third major category of Digital Item Adaptation tools. A BSD describes the syntax – in most cases, the high level structure – of a binary media resource. Using such a description, a Digital Item resource adaptation engine can transform the bitstream and the corresponding description using editing-style operations such as data truncation and simple modifications. Streaming instructions enhance the BSD by defining a set of properties and attributes which describe the fragmentation, timing and random access point indication for the BSD and its described resource. They are used for streamed processing and transport, e.g., in dynamic and distributed adaptation scenarios.

In Clause 2, add the following text:

ISO/IEC 14977:1996, *Information technology — Syntactic metalanguage — Extended BNF*

ISO/IEC 14496-10, *Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding*

ISO/IEC 16262, *Information technology — ECMAScript language specification*

Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Recommendation 07 April 2004

Insert a new subclause 3.1.4 as follows:

3.1.4 Dynamic and Distributed Adaptation-specific terms and definitions

3.1.4.1
dynamic adaptation
adaptation of **Digital Item**s according to dynamically changing **usage environment**s

EXAMPLE The available bandwidth may drop during a streaming session and the Digital Item is consequently adapted to this new usage environment.

3.1.4.2
distributed adaptation
multiple adaptation steps successively performed on different ISO/IEC 21000 **peers**

EXAMPLE A same resource may be successively adapted on a server, network node and/or terminal.

3.1.4.3
process unit
well-formed fragment of a BSD or other XML metadata used for adaptation purposes, that can be consumed as such by the ISO/IEC 21000 **peer**, and to which a time information may be attached, indicating the point in time when it becomes available to the ISO/IEC 21000 **peer** for consumption

NOTE A process unit is a processing-oriented concept rather than a delivery-oriented concept. It does not depend on any encoding method used for delivering it.

3.1.4.4
XML fragmentation
authoring process by which an XML document is split into **process unit**s meaningful for consumption purposes

NOTE This process may attach time information to the output **process unit**s indicating the point in time when they become available to the ISO/IEC 21000 **peer** for consumption.

3.1.4.5
processing time
point in time when a **process unit** is available to the ISO/IEC 21000 **peer** for consumption

3.1.4.6
access unit
smallest segment of data that is atomic in time

EXAMPLE ISO/IEC 14496-1 access unit or ISO/IEC 15938-1 access unit (cf. TeM or BiM).

3.1.4.7
encoding
process by which a **process unit** is transformed into another representation for storage and/or delivery purposes

NOTE 1 The output of encoding is a sequence of one or more **access units**.

NOTE 2 Encoding does not necessarily imply compression (see for example TeM for XML).

3.1.4.8
random access point
point which indicates whether an **access unit** carries information that can be consumed by a ISO/IEC 21000 **peer** independent from any other **access units**

In subclause 3.2, append the following text at the end of this sublcause and order the resulting list alphabetically:

PU:	Process Unit
AU:	Access Unit
RAP:	Random Access Point
CTS:	Composition Time Stamp
DTS:	Decoding Time Stamp
PTS:	Processing Time Stamp

In subclause 3.4, append the following new rows to Table 1:

msi	urn:mpeg:mpeg21:2003:01-DIA-MSI-NS
si	urn:mpeg:mpeg21:2003:01-DIA-XSI-NS
ps	urn:mpeg:mpeg21:2003:01-DIA-PSS-NS
bs1x	urn:mpeg:mpeg21:2003:01-DIA-BSDL1x-NS
bs2x	urn:mpeg:mpeg21:2003:01-DIA-BSDL2x-NS

Insert a new subclause 4.4.4 as follows:

4.4.4 Classification scheme resolution mechanism

ISO/IEC 15938-5 specifies a descriptor named `mpeg7:ClassificationSchemeAliasType` for assigning an alias to a namespace. Additionally, this part of ISO/IEC 21000 specifies a more compact syntax for the same purpose by making use of the XML namespace binding context. The XML Namespaces recommendation [13] specified how to bind a prefix to a namespace. This namespace binding context is then inherited by the child elements of the element where the namespace is declared. This part of 21000 uses this namespace binding context to resolve the aliases used in classification schemes.

In case of conflicts between the namespace binding context and the binding specified by `mpeg7:ClassificationSchemeAliasType`, the namespace bound by the latter is used.

EXAMPLE 1 When a classification scheme term such as `:myAlias:myCSTerm`, the alias `myAlias` alias is resolved against the namespace binding context, i.e. the XML namespaces declaration.

EXAMPLE 2 Classification Scheme resolution mechanism using the XML namespace declaration. The alias `MV4` is resolved against the namespace binding context, i.e., the XML namespace declaration within the Description element which is inherited by its child elements (`xmlns:MV4="urn:mpeg:mpeg4:video:cs:syntacticalLabels"`).

```
<DIA xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS">
<Description xsi:type="gbsd:gBSDType"
    xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
    bs1:bitstreamURI="akiyo.mpg4"
    xmlns:MV4="urn:mpeg:mpeg4:video:cs:syntacticalLabels">

  <gBSDUnit syntacticalLabel=":MV4:VO" start="0" length="18" />
  <!-- ... and so on ... -->

 </Description>
</ DIA>
```

Remove subclause 5.2.3, Unsigned integer datatypes semantics.

In subclause 5.5.2, append the following text to the schema of this subclause:

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

<complexType name="ExternalVectorRefType" abstract="true">
   <complexContent>
      <extension base="dia:VectorDataType">
         <attribute name="uri" type="anyURI" use="required"/>
      </extension>
   </complexContent>
</complexType>

<!-- ################################################## -->
<!-- Definition of ExternalNMTokenVectorRefType         -->
<!-- ################################################## -->

<complexType name="ExternalNMTokenVectorRefType">
   <complexContent>
      <extension base="dia:ExternalVectorRefType"/>
   </complexContent>
</complexType>
```

In subclause 5.5.3, append the following text after the semantics of the `SemanticalDataRefType`*:*

Semantics of the `ExternalVectorRefType`:

`ExternalVectorRefType`	`ExternalVectorRefType` extends `VectorDataType` and provides a base abstract type for referencing external data by use of the XPointer framework.
`uri`	Describes the URI fragment identifier to be evaluated.

Semantics of the `ExternalNMTokenVectorRefType`:

`ExternalNMTokenVectorRefType`	Describes a vector of `xsd:NMTOKEN` values which must be returned by the URI fragment identifier evaluation.

In subclause 6.5.4.2, replace the text of the whole subclause with the following text:

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

<complexType name="CodecCapabilitiesType">
  <complexContent>
    <extension base="dia:TerminalCapabilityBaseType">
      <sequence>
        <element name="Decoding" type="dia:CodecCapabilityBaseType"
           minOccurs="0" maxOccurs="unbounded"/>
        <element name="Encoding" type="dia:CodecCapabilityBaseType"
           minOccurs="0" maxOccurs="unbounded"/>
        <element name="DecodingEncoding" type="dia:CodecCapabilityBaseType"
           minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
```

In subclause 6.5.4.3, append a new row after the definition of the `Encoding` element:

`DecodingEncoding`	Describes the decoding and encoding capabilities of the terminal.

In subclause 6.5.12.2, add a new attribute to the `AudioOutputCapabilitiesType`:

```
    <attribute name="silenceSuppression" type="boolean" use="optional"/>
```

In subclause 6.5.12.3, append a new row after the definition of the `numChannels` attribute:

`silenceSuppression`	Describes whether silence suppression shall be applied for the audio. NOTE This feature is usually applied to stop the audio transfer through the network during periods when the participants in an audio conference are currently not talking. Furthermore, this feature is applied to decrease the used bandwidth in moments of silence.

Insert the following as a new subclause 6.5.29 and increment subsequent clauses accordingly:

6.5.29 RELCapabilities

6.5.29.1 Introduction

This subclause specifies the rights expression languages a terminal supports. A terminal can support none, one, or multiple rights expression languages (RELs). In the case of supporting multiple RELs, the terminal can also specify the preference level for each one of them.

6.5.29.2 RELCapabilities syntax

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

<complexType name="RELCapabilitiesType">
   <complexContent>
      <extension base="dia:TerminalCapabilityBaseType">
         <sequence>
            <element name="RightsLanguage" type="dia:RELCapabilityType"
               minOccurs="0" maxOccurs="unbounded"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

6.5.29.3 RELCapabilities semantics

Semantics of the `RELCapabilitiesType`:

Name	*Definition*
`RELCapabilitiesType`	Tool for describing the rights expression language capabilities a terminal supports.
`RightsLanguage`	Describes a rights expression language supported by a terminal.

Insert the following as a new subclause 6.5.30 and increment subsequent clauses accordingly:

6.5.30 RELCapability

6.5.30.1 Introduction

This subclause specifies a rights language that a terminal supports.

6.5.30.2 RELCapability syntax

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

<complexType name="RELCapabilityType">
   <complexContent>
      <extension base="dia:DIABaseType">
         <sequence>
            <element name="Name" type="mpeg7:ControlledTermUseType"/>
         </sequence>
         <attribute name="preferenceLevel" type="positiveInteger"
            use="optional"/>
      </extension>
   </complexContent>
</complexType>
```

6.5.30.3 RELCapability semantics

Semantics of the `RELCapabilityType`:

Name	*Definition*
`RELCapabilityType`	Tool for describing a rights expression language a terminal supports.
`Name`	Describes the name of a rights language. A classification scheme that may be used for this purpose is the `RightsLanguagesCS` defined in Annex A.2.16.
`preferenceLevel`	Describes the preference level of a rights language among the supported ones by a terminal. The value of one indicates the top preference level.

In subclause 6.5.29 (will be 6.5.31 after incrementing) append the following text:

EXAMPLE 5 The following example provides an instantiation of the RELCapabilities of the terminal. In this example, the terminal is able to interpret the following languages in decreased level preference: ISO/IEC 21000-5 Base profile and ISO/IEC 21000-5.

```
<DIA>
  <Description xsi:type="UsageEnvironmentType">
    <UsageEnvironmentProperty xsi:type="TerminalsType">
      <Terminal>
        <TerminalCapability xsi:type="RELCapabilitiesType">
          <RightsLanguage xsi:type="RELCapabilityType" PreferenceLevel="1">
            <Name
               href="urn:mpeg:mpeg21:2003:01-DIA-RightsLanguagesCS-
NS:2003:RELProfile:Base">
              <mpeg7:Name xml:lang="en">MPEG-21 REL Base Profile</mpeg7:Name>
            </Name>
          </RightsLanguage>
          <RightsLanguage xsi:type="RELCapabilityType" PreferenceLevel="2">
            <Name
               href="urn:mpeg:mpeg7:2003:01-DIA-RightsLanguagesCS-NS:2003:REL">
              <mpeg7:Name xml:lang="en">MPEG-21 REL</mpeg7:Name>
            </Name>
          </RightsLanguage>
        </TerminalCapability>
      </Terminal>
    </UsageEnvironmentProperty>
  </Description>
</DIA>
```

In subclause 6.6.4.2, add a new attribute to the `NetworkCapabilityType`:

```
    <attribute name="maxPacketSize" type="positiveInteger" use="optional"/>
```

In subclause 6.6.4.3, append a new row after the definition of the `errorCorrection` attribute:

`maxPacketSize`	Describes the maximum size of a packet in bit. NOTE This attribute is considered for technologies that support packet transport (e.g. IP networks).

In subclause 7.2.2, append the following text to the schema:

```
    <!-- ######################################################## -->
    <!-- Definition of the StreamingRefType                        -->
    <!-- ######################################################## -->

    <complexType name="StreamingRefType">
        <complexContent>
            <extension base="dia:ReferenceType">
                <attribute name="minPUSize" type="positiveInteger"
                    use="optional"/>
                <attribute name="avgPUSize" type="positiveInteger"
                    use="optional"/>
                <attribute name="maxPUSize" type="positiveInteger"
                    use="optional"/>
            <attribute name="minBitstreamPUSize" type="positiveInteger"
               use="optional"/>
            <attribute name="avgBitstreamPUSize" type="positiveInteger"
               use="optional"/>
            <attribute name="maxBitstreamPUSize" type="positiveInteger"
               use="optional"/>
         </extension>
      </complexContent>
   </complexType>
```

In subclause 7.2.3, append the following text at the end of this subclause:

Semantics of the `StreamingRefType`:

Name	*Definition*
`StreamingRefType`	Describes a reference to the (generic) Bitstream Syntax Description or AdaptationQoS description which can be used by a streaming processor. The target of this reference shall be of `bs1:BSDType` or `dia:AdaptationQoSType`. NOTE In case it is used by the `SteeringDescriptionRef` element the reference shall be of `dia:AdaptationQoSType`. In case it is used by the `BSDRef` element the reference shall be of `bs1:BSDType`.
`minPUSize`	Describes the minimum size of the process unit in bytes.
`avgPUSize`	Describes the average size of the process unit in bytes.
`maxPUSize`	Describes the maximum size of the process unit in bytes.
`minBitstreamPUSize`	Describes the minimum size of the bitstream fragment in bytes which is described by a single process unit. NOTE 1 This attribute shall appear only once, e.g., within a single BSDLink description.
`avgBitstreamPUSize`	Describes the average size of the bitstream fragment in bytes which is described by a single process unit. NOTE 2 This attribute shall appear only once, e.g., within a single BSDLink description.
`maxBitstreamPUSize`	Describes the maximum size of the bitstream fragment in bytes which is described by a single process unit. NOTE 3 This attribute shall appear only once, e.g., within a single BSDLink description.

NOTE 4 The information within the `StreamingRefType` may be used by the buffer of an adaptation engine in a streaming scenario – similar to a decoding buffer of a media resource player – which is related to the processing part of the delivery and not the delivery itself.

In subclause 7.2.4, append the following text at the end of this subclause:

EXAMPLE This example demonstrates the usage of the `dia:StreamingRefType` within a BSDLink description for an ISO/IEC 14496-3 bit sliced arithmetic coding (BSAC) bitstream.

```
<DIA>
  <Description xsi:type="BSDLinkType">
    <SteeringDescriptionRef xsi:type="StreamingRefType"
       uri="bsac_AQoS.xml"
       maxPUSize="52"/>
    <BSDRef xsi:type="StreamingRefType"
       uri="bsac_gBSD.xml"
       maxPUSize="132"
       minBitstreamPUSize="98"
       maxBitstreamPUSize="515"
       avgBitstreamPUSize="228"/>
    <BitstreamRef uri="bsac.raw"/>
    <BSDTransformationRef uri="bsac.xslt"
       type="http://www.w3.org/1999/XSL/Transform"/>
    <Parameter xsi:type="IOPinRefType" name="nlayers">
      <Value>LAYERS_OF_SCALABLE_AUDIO</Value>
    </Parameter>
  </Description>
</DIA>
```

In subclause 8.1.6, append a new paragraph at the end of this subclause:

Subclauses 8.5 and 8.6 specify media resource and XML streaming instructions required for adapting resources in streaming scenarios, in particular for dynamic and distributed adaptation scenarios. The media resource and XML streaming instructions define a set of properties and attributes which describe the fragmentation, timing, and additional information of a BSD and its corresponding bitstream. Furthermore, subclause 8.7 specifies a so-called properties stylesheet enabling to set these properties without modifying the XML document.

In subclause 8.2.1, append a new paragraph at the end of this subclause:

Optional features

This part of ISO/IEC 21000 specifies a set of optional features in Annex L organized in modules. A conformant BSDtoBin or BintoBSD parser is not required to implement these features. Alternatively, when it implements, it shall conform to their specification in this part of ISO/IEC 21000. These features are organized in three modules:

— Use of ISO/IEC 16262 (ECMAScript) for user-defined datatypes.

— Use of ISO/IEC 16262 (ECMAScript) for user-defined XPath functions.

— Support for array of variables.

NOTE A set of non-normative features of BSDL used for extensibility and for debugging is defined in Annex M.

Information about the BS Schema

A `bs1:schemaInformation` element contained at the root of the schema document wraps information about the BS Schema. This information is provided as attributes in BSDL-1 or BSDL-2 namespaces. BSDL-1 attributes are relevant to both BSDtoBin and BintoBSD processors whereas BSDL-2 attributes are only relevant to BintoBSD.

When a BSDL-1 attribute is provided both in the BSD and the `bs1:schemaInformation` element of the BS Schema with conflicting values, the value used in the BSD supersedes the one used in the BS Schema.

In subclause 8.3.1.1, replace Table 4 with the following table:

See Section	Datatype name	Encoding length
3.2.1	`xsd:string`	Indefinite
3.3.1	`xsd:normalizedString`	Indefinite
3.2.4	`xsd:float`	4
3.2.5	`xsd:double`	8
3.2.15	`xsd:hexBinary`	Indefinite
3.2.16	`xsd:base64Binary`	Indefinite
3.3.16	`xsd:long`	8
3.3.17	`xsd:int`	4
3.3.18	`xsd:short`	2
3.3.19	`xsd:byte`	1
3.3.21	`xsd:unsignedLong`	8
3.3.22	`xsd:unsignedInt`	4
3.3.23	`xsd:unsignedShort`	2
3.3.24	`xsd:unsignedByte`	1

In subclause 8.3.1.1, replace the following paragraph:

All integer types (`xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`) and their unsigned derivatives (`xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, `xsd:unsignedByte`) are encoded in big endian. BSDL does not provide equivalent types for little endian, but the BS Schema author still has the possibility to specify a little endian value by decomposing it into several ordered bytes.

with:

All integer types (`xsd:long`, `xsd:int`, `xsd:short`, `xsd:byte`) and their unsigned derivatives (`xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, `xsd:unsignedByte`) are encoded in big endian. BSDL provides equivalent types for little endian in subclause 8.3.1.4.

In subclause 8.3.1.1, replace the following paragraph:

`xsd:normalizedString` is coded as US-ASCII. Note that the `xsd:string` type is not supported by BSDL since it allows carriage return and line feed characters whose processing is platform-dependent.

with:

`xsd:string` and `xsd:normalizedString` are coded as US-ASCII. Note that care should be taken with the `xsd:string` type since it allows carriage return and line feed characters whose processing is platform-dependent. In case such control characters with platform-dependent encoding are present in the stream it is recommended to escape these characters in the BSD with a character reference as specified in XML such as "&#x0D;" or "&#13;" instead of the Carriage Return character.

In subclause 8.3.1.1, replace the following paragraph:

BSDL built-in datatypes

In addition to the XML Schema built-in datatypes listed above, BSDL provides two built-in datatypes named `bs1:byteRange` and `bs1:bitstreamSegment`. Their syntax is defined in the *Schema for BSDL-1 extensions*. They carry specific semantics in the context of bitstream generation explained in subclause 8.3.1.4 and have an indefinite length similarly to `xsd:hexBinary` and `xsd:base64Binary`.

with:

BSDL built-in datatypes

In addition to the XML Schema built-in datatypes listed above, BSDL provides two built-in datatypes pointing to segments of data within the described bitstream, a set of built-in integer datatypes supporting little-endian encoding, two built-in datatypes corresponding to the exponentional Golomb variable-length coding scheme, and string datatypes supporting UTF-8 and UTF-16 encoding, as well as strings terminated with a null character. Their syntax is defined in the *Schema for BSDL-1 extensions*. They carry specific semantics in the context of bitstream generation explained in subclause 8.3.1.4.

Additionaly, BSDL provides datatypes for unsigned integer values encoded from 1 to 32 bits. Their syntax is defined in a separate schema document with no target namespace provided in subclause 8.3.1.2 and their semantics is defined in subclause 8.3.1.4. This schema document is included by the *Schema for BSDL-1 extensions* and the datatype consequently inherit the BSDL-1 namespace.

Extension datatypes

In addition to XML Schema and BSDL built-in datatypes, BSDL provides the possibility to specify proprietary extension datatypes and use them in a BS Schema. An extension datatype is identified by a URI comprising a base part followed by a fragment identifier. The `bs1:codec` attribute allows a schema author to override a datatype defined in the schema. For this, the simple type needs to be derived by extension by declaring a `bs1:codec` attribute with a fixed or default value equal to the URI identifying the extension datatype. It is then possible to extend the implementation of BSDtoBin and BintoBSD processors to implement the decoding (i.e., reading the syntactical element from the bitstream and instantiating its lexical representation) and encoding (i.e., binary encoding the lexical value and writing it to the bitstream) of these datatypes. This extension mechanism is further specified in Annex K.

Properties in BS Description

For processing purpose, BSDL defines a set of abstract element properties that enrich the infoset of BS Description elements and are considered by the BSDtoBin processor. Each has an associated attribute. These properties are ineritable from the parent element. Properties may be specified by the associated attribute provided by the BS Description or the BS Schema (via its *default* or *fixed* value) or inherited from the parent element in the following way:

For each BS Decsription element in document order:

— if the associated attribute is present in the BS Description element, then the property is obtained from the value specified by the attribute; the way the property is computed from the attribute value is specific to the property,

— otherwise, if the BS Schema declares a default or fixed value for the associated attribute, then the property is obtained from this value,

— otherwise, if the element has a parent element, the property is inherited from the property of the parent element,

— otherwise (root element), it takes the default value for the document.

The properties are the following:

— The *addressUnit* property is associated with the the `bs1:addressUnit` attribute. It indicates the address unit used by the `bs1:byteRange` and `bs1:bitstreamSegment` datatypes. Its value is *bit* or *byte*, the default value for the document is *byte*.

— The *ignore* property is associated with the the the `bs1:ignore` attribute. Its value is a *true* or *false*. If set to *true*, BSDtoBin shall ignore the current element. Otherwise, process the element normally. The default value for the document is *false*.

— The *bitstreamURI* property indicates the absolute URI of the described bitstream and is associated to the `bs1:bitstreamURI` attribute. If the `bs1:bitstreamURI` attribute indicates a relative URI, then the *bitstreamURI* property of the element is obtained by resolving the `bs1:bitstreamURI` attribute value against the *bitstreamURI* property of the parent element. Otherwise (absolute value), the *bitstreamURI* property is set to the value of the `bs1:bitstreamURI` attribute. The default value for the document is the absolute URI of the BSD document. If a `bs1:bitstreamURI` attribute is specified for the BSD root element (either explicitely in the BSD or by default in the BS Schema) with a relative value, the resulting bitstreamURI property is therefore obtained by resolving the relative URI against the absolute BSD URI.

— The *insertEmPrevByte* property is associated with the `bs1:insertEmPrevByte` attribute. Its value is an even number of hexadecimal strings or *none*. If the `bs1:insertEmPrevByte` attribute is empty, then the property value is *none*. The default value for the document is *none*.

In subclause 8.3.1.2, replace the text of the whole subclause with the following text:

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7/Amd.2 -->
<!-- Schema for BSDL-1 extensions -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="BSDL-1-AMD2.xsd"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:dia="urn:mpeg:mpeg21:2003:01-DIA-NS">

  <annotation>
    <documentation>
      Schema for BSDL-1 extensions
    </documentation>
  </annotation>

  <import namespace="urn:mpeg:mpeg21:2003:01-DIA-NS" schemaLocation="DIA.xsd"/>
  <include schemaLocation="UnsignedIntegers-AMD2.xsd"/>

  <!-- #################################### -->
  <!--     BitstreamSyntaxDescriptionType    -->
  <!-- #################################### -->

  <complexType name="BSDType" abstract="true">
    <complexContent>
      <extension base="dia:DIADescriptionType"/>
    </complexContent>
  </complexType>
  <element name="schemaInformation">
    <complexType>
      <complexContent>
        <restriction base="anyType">
          <anyAttribute processContents="lax"/>
        </restriction>
      </complexContent>
    </complexType>
  </element>

  <!-- #################################### -->
  <!--          BSDL Attributes             -->
  <!-- #################################### -->

  <attribute name="ignore" type="boolean"/>
  <attribute name="bitstreamURI" type="anyURI"/>

  <attribute name="addressUnit" type="bs1:unitsType"/>

  <attribute name="codec" type="anyURI"/>
  <attribute name="requiredExtensions" type="bs1:anyURIList"/>

  <!-- Attribute specifying the insertion of emulation prevention bytes -->
  <attribute name="insertEmPrevByte" type="bs1:hexBinaryStrings"/>

  <!-- Address unit definition -->
  <simpleType name="unitsType">
    <restriction base="NMTOKEN">
      <enumeration value="bit"/>
      <enumeration value="byte"/>
    </restriction>
  </simpleType>

  <!-- A list of hexBinary strings -->
  <simpleType name="hexBinaryStrings">
    <list itemType="hexBinary"/>
  </simpleType>

  <!-- A list of anyURI -->
  <simpleType name="anyURIList">
    <list itemType="anyURI"/>
  </simpleType>

  <!-- #################################### -->
  <!--       BSDL Built-in Datatype         -->
  <!-- #################################### -->

  <simpleType name="byteRange">
    <restriction>
      <simpleType>
        <list itemType="nonNegativeInteger"/>
      </simpleType>
      <length value="2" fixed="true"/>
    </restriction>
  </simpleType>

  <complexType name="bitstreamSegment">
    <complexContent>
      <restriction base="anyType">
        <attribute name="start" type="nonNegativeInteger" use="optional"/>
        <attribute name="length" type="nonNegativeInteger" use="optional"/>
      </restriction>
    </complexContent>
  </complexType>
```

```
<simpleType name="stringUTF16NT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16BENT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16LENT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF8NT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16BE">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16LE">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF8">
  <restriction base="string"/>
</simpleType>
<simpleType name="longLE">
  <restriction base="long"/>
</simpleType>
<simpleType name="intLE">
  <restriction base="int"/>
</simpleType>
<simpleType name="shortLE">
  <restriction base="short"/>
</simpleType>
<simpleType name="unsignedLongLE">
  <restriction base="unsignedLong"/>
</simpleType>
<simpleType name="unsignedIntLE">
  <restriction base="unsignedInt"/>
</simpleType>
<simpleType name="unsignedShortLE">
  <restriction base="unsignedShort"/>
</simpleType>
<simpleType name="align32">
  <restriction base="hexBinary">
    <length value="4"/>
  </restriction>
</simpleType>
<simpleType name="align16">
  <restriction base="hexBinary">
    <length value="2"/>
  </restriction>
</simpleType>
<simpleType name="align8">
  <restriction base="hexBinary">
    <length value="1"/>
  </restriction>
</simpleType>

<simpleType name="unsignedExpGolomb">
  <restriction base="unsignedInt"/>
</simpleType>
```

```
  <simpleType name="signedExpGolomb">
    <restriction base="int"/>
  </simpleType>

</schema>
```

The following schema document specifies a library of unsigned integer datatypes encoded on 1 to 32 bits. It is defined with no target namespace and is included by the Schema for BSDL-1 extensions. Consequently, all the datatypes inherit the BSDL-1 namespace.

NOTE Since this schema document has no target namespace and is included by the Schema for BSDL-1 extensions, the base types must be qualified with a prefix bound to the XMl Schema namespace in order to avoid namespace ambiguity.

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7:2004/Amd.2 -->
<!-- Unsigned Integer Datatypes Schema -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="UnsignedIntegers-AMD2.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <simpleType name="b32">
    <restriction base="xsd:unsignedInt"/>
  </simpleType>
  <simpleType name="b31">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="2147483648"/>
    </restriction>
  </simpleType>
  <simpleType name="b30">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="1073741824"/>
    </restriction>
  </simpleType>
  <simpleType name="b29">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="536870912"/>
    </restriction>
  </simpleType>
  <simpleType name="b28">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="268435456"/>
    </restriction>
  </simpleType>
  <simpleType name="b27">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="134217728"/>
    </restriction>
  </simpleType>
  <simpleType name="b26">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="67108864"/>
    </restriction>
  </simpleType>
  <simpleType name="b25">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="33554432"/>
    </restriction>
  </simpleType>
  <simpleType name="b24">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="16777216"/>
    </restriction>
  </simpleType>
  <simpleType name="b23">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="8388608"/>
    </restriction>
  </simpleType>
  <simpleType name="b22">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="4194304"/>
    </restriction>
  </simpleType>
  <simpleType name="b21">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="2097152"/>
    </restriction>
  </simpleType>
  <simpleType name="b20">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="1048576"/>
    </restriction>
  </simpleType>
  <simpleType name="b19">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="524288"/>
    </restriction>
  </simpleType>
  <simpleType name="b18">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="262144"/>
    </restriction>
  </simpleType>
  <simpleType name="b17">
    <restriction base="xsd:unsignedInt">
      <maxExclusive value="131072"/>
    </restriction>
  </simpleType>
  <simpleType name="b16">
    <restriction base="xsd:unsignedShort"/>
  </simpleType>
  <simpleType name="b15">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="32768"/>
    </restriction>
  </simpleType>
  <simpleType name="b14">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="16834"/>
    </restriction>
  </simpleType>
  <simpleType name="b13">
    <restriction base="xsd:unsignedShort">
```

```
      <maxExclusive value="8192"/>
    </restriction>
  </simpleType>
  <simpleType name="b12">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="4096"/>
    </restriction>
  </simpleType>
  <simpleType name="b11">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="2048"/>
    </restriction>
  </simpleType>
  <simpleType name="b10">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="1024"/>
    </restriction>
  </simpleType>
  <simpleType name="b9">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="512"/>
    </restriction>
  </simpleType>
  <simpleType name="b8">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="256"/>
    </restriction>
  </simpleType>
  <simpleType name="b7">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="128"/>
    </restriction>
  </simpleType>
  <simpleType name="b6">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="64"/>
    </restriction>
  </simpleType>
  <simpleType name="b5">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="32"/>
    </restriction>
  </simpleType>
  <simpleType name="b4">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="16"/>
    </restriction>
  </simpleType>
  <simpleType name="b3">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="8"/>
    </restriction>
  </simpleType>
  <simpleType name="b2">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="4"/>
    </restriction>
  </simpleType>
  <simpleType name="b1">
    <restriction base="xsd:unsignedShort">
      <maxExclusive value="2"/>
    </restriction>
  </simpleType>
</schema>
```

In subclause 8.3.1.3, append the following new rows:

`schemaInformation`	Element to be included in `xsd:schema` component via the `xsd:annotation/xsd:appinfo` combination. It is meant to contain attributes providing information on the BS Schema.
`addressUnit`	Specifies the addressing unit (bit or byte) used by `bs1:byteRange` and `bs1:bitstreamSegment` datatypes.
`codec`	Specifies the extension datatype. This attribute may be used either in the BS Description or declared in the BS Schema with a default or fixed value. Its value is the URI identifying the extension datatype.
`insertEmPrevByte`	Specifies the insertion of emulation prevention bytes. It contains an even number of hexadecimal strings. For each pair of hexadecimal strings, the first string specifies the byte sequence that requires the insertion of emulation prevention bytes and the second string specifies the byte sequence that should be written to the output bitstream instead. When the attribute is empty, no emulation prevention byte is inserted.
`requiredExtensions`	The `bs1:requiredExtensions` attribute indicates the user-defined datatypes required by this schema document. Its value is a list of URIs. Each URI indicates either a single datatype or a library of datatypes. In the first case, it contains a fragment identifier pointing to datatype name itself. In the second case, it identifies the library. This attribute shall be used in the BSD or in the BS Schema within the `bs1:schemaInformation` element. When provided both in the BSD and the `bs1:schemaInformation` element of the BS Schema with conflicting values, the value used in the BSD supersedes the one used in the BS Schema.

In subclause 8.3.1.4, replace the semantics of byteRange and bitstreamSegment by the following new text:

Name	*Definition*
`byteRange`	BSDL built-in datatype indicating the byte range of the resource identified by the bitstream URI of the current element. It consists in a list of two non-negative integers. The first integer parameter indicates the offset of the relevant range of data and the second parameter indicates its length. These values are specified in bits or bytes according to the value of the addressUnit property. The offset is relative to the start of the bitstream.

Name	*Definition*
`bitstreamSegment`	BSDL built-in complex type indicating a bitstream segment. The `start` and `length` attributes respectively indicate the offset of the segment and its length. These values are specified in bits or bytes according to the value of the addressUnit property. The offset is relative to the start of the bitstream. The BSDtoBin parser copies the relevant data segment only if the current element has no child elements. This built-in datatype is provided for compatibility with gBS Schema (see subclause 8.3.2) and shall not be used for BS Description generation (BSDL-2).

In subclause 8.3.1.4, append the following new rows:

`longLE`, `intLE`, `shortLE`, `unsignedLongLE`, `unsignedIntLE`, `unsignedShortLE`,	Integer datatypes with the same lexical value as their XML Schema counterpart (respectively `xsd:long`, `xsd:int`, `xsd:short`, `xsd:unsignedLong`, `xsd:unsignedInt` and `xsd:unsignedShort`), but with little-endian encoding.
`stringUTF8`, `stringUTF16BE`, `stringUTF16LE`, `stringUTF16`	String datatypes with UTF-8 and UTF-16 character encoding. UTF8 refers to the eight-bit UCS Transformation Format, UTF16BE refers to the sixteen-bit UCS Transformation Format, big-endian byte order, UTF16LE refers to the sixteen-bit UCS Transformation Format, little-endian byte order, and UTF16 refers to the sixteen-bit UCS Transformation Format, byte order identified by an optional byte-order mark.
`stringUTF8NT`, `stringUTF16BENT`, `stringUTF16LENT`, `stringUTF16NT`	The same semantics apply as for `bs1:stringUTF8`, `bs1:stringUTF16BE`, `bs1:stringUTF16LE`, and `bs1:stringUTF16`, but with a terminating null character. Note that the terminating null character is not allowed in XML and hence does not appear in the lexical value. However BintoBSD must read it from the input bitstream, and BSDtoBin and gBSDtoBin must generate it into the output bitstream.
`align8`, `align16`, `align32`	Datatypes enabling reading a value from the input bitstream and writing padding bits to the output bitstream until it is aligned on a 1 byte (for `align8`), 2 byte (for `align16`) or 4 byte (for `align32`) word. BintoBSD reads bits from the input bitstream until it is aligned on respectively a 1 byte, 2 byte or 4 byte word and instantiates an empty element. When the bitstream is already correctly aligned, BintoBSD does not read any bit but still instantiates an empty element. BSDtoBin considers the element value in the PSVI, namely the lexical value indicated in the BS Description element if present, the fixed default specified in the BS Schema otherwise. If the BS Description element is empty and no fixed or default value is specified by the BS Schema, then BSDtoBin uses 0 as default value. BSDtoBin then encodes the obtained value, starting from the most significant bit until the output bitstream is adequately aligned. If the bitstream is already correctly aligned, BSDtoBin does not write any bit.

`unsignedExpGolomb`	BSDL built-in datatype corresponding to the unsigned exp-golomb encoding specified in ISO/IEC 14496-10.
`signedExpGolomb`	BSDL built-in datatype corresponding to the signed exp-golomb encoding specified in ISO/IEC 14496-10.
`b1`	A 1-bit unsigned integer value.
`b2`	A 2-bit unsigned integer value.
`b3`	A 3-bit unsigned integer value.
`b4`	A 4-bit unsigned integer value.
`b5`	A 5-bit unsigned integer value.
`b6`	A 6-bit unsigned integer value.
`b7`	A 7-bit unsigned integer value.
`b8`	A 8-bit unsigned integer value.
`b9`	A 9-bit unsigned integer value.
`b10`	A 10-bit unsigned integer value.
`b11`	A 11-bit unsigned integer value.
`b12`	A 12-bit unsigned integer value.
`b13`	A 13-bit unsigned integer value.
`b14`	A 14-bit unsigned integer value.
`b15`	A 15-bit unsigned integer value.
`b16`	A 16-bit unsigned integer value.
`b17`	A 17-bit unsigned integer value.
`b18`	A 18-bit unsigned integer value.
`b19`	A 19-bit unsigned integer value.
`b20`	A 20-bit unsigned integer value.
`b21`	A 21-bit unsigned integer value.
`b22`	A 22-bit unsigned integer value.
`b23`	A 23-bit unsigned integer value.
`b24`	A 24-bit unsigned integer value.
`b25`	A 25-bit unsigned integer value.

`b26`	A 26-bit unsigned integer value.
`b27`	A 27-bit unsigned integer value.
`b28`	A 28-bit unsigned integer value.
`b29`	A 29-bit unsigned integer value.
`b30`	A 30-bit unsigned integer value.
`b31`	A 31-bit unsigned integer value.
`b32`	A 32-bit unsigned integer value.

In subclause 8.3.1.6, append the following lines to the first paragraph:

Furthermore, when an element of the description is empty and its declaration in the schema specifies a value constraint (fixed or default attribute), the BSDtoBin processes this default value.

In subclause 8.3.2.2, replace the text of the whole subclause with following text.

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7/Amd.2 -->
<!-- generic Bitstream Syntax Schema (gBSSchema) -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="gBSSchema-AMD2.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  elementFormDefault="qualified" attributeFormDefault="unqualified">

  <annotation>
    <documentation>
      Definition of generic Bitstream Syntax Schema (gBSSchema)
    </documentation>
  </annotation>

  <import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
          schemaLocation="BSDL-1-AMD2.xsd"/>
  <include schemaLocation="UnsignedIntegers.xsd"/>

  <element name="gBSDUnit" type="gbsd:gBSDUnitType"/>

  <!-- #################################### -->
  <!--          gBSDType definition          -->
  <!-- #################################### -->

  <complexType name="gBSDType">
    <complexContent>
      <extension base="bs1:BSDType">
        <sequence>
          <element ref="gbsd:gBSDUnit" maxOccurs="unbounded"/>
        </sequence>
        <attributeGroup ref="gbsd:addressAttributes"/>
        <anyAttribute namespace="##other" processContents="lax"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- #################################### -->
  <!--      Attribute group definition      -->
  <!-- #################################### -->

  <attributeGroup name="addressAttributes">
    <attribute name="addressMode" type="gbsd:addressModeType" use="optional"/>
    <attribute name="addressUnit" type="gbsd:unitsType" use="optional"/>
    <attribute ref="bs1:bitstreamURI" use="optional"/>
  </attributeGroup>

  <!-- #################################### -->
  <!--         gBSDUnit definition          -->
  <!-- #################################### -->

  <complexType name="gBSDUnitType">
    <complexContent>
      <extension base="bs1:bitstreamSegment">
        <choice minOccurs="0" maxOccurs="unbounded">
          <element name="Parameter" type="gbsd:paramType"/>
          <element ref="gbsd:gBSDUnit"/>
        </choice>
        <attribute name="syntacticalLabel" type="gbsd:labelType" use="optional"/>
        <attribute name="marker" type="gbsd:markerType" use="optional"/>
        <attributeGroup ref="gbsd:addressAttributes"/>
        <anyAttribute namespace="##other" processContents="lax"/>
      </extension>
    </complexContent>
  </complexType>

  <!-- Resource address unit definition -->
  <simpleType name="unitsType">
    <restriction base="NMTOKEN">
      <enumeration value="bit"/>
      <enumeration value="byte"/>
    </restriction>
  </simpleType>

  <!-- Address mode definition -->
  <simpleType name="addressModeType">
    <restriction base="NMTOKEN">
      <enumeration value="Absolute"/>
      <enumeration value="Consecutive"/>
      <enumeration value="Offset"/>
    </restriction>
  </simpleType>

  <!-- ******* Marker definition ************ -->
  <simpleType name="markerType">
    <list itemType="NMTOKEN"/>
  </simpleType>
```

```
  <!-- ******* Label definition ************ -->
  <simpleType name="labelType">
    <union>
      <simpleType>
        <restriction base="NMTOKEN">
          <whiteSpace value="collapse"/>
          <pattern value=":[^:]+:[^:]+"/>
        </restriction>
      </simpleType>
      <simpleType>
        <restriction base="anyURI"/>
      </simpleType>
    </union>
  </simpleType>

  <!-- #################################### -->
  <!--          Parameter definition        -->
  <!-- #################################### -->

  <complexType name="paramType">
    <complexContent>
      <extension base="bs1:bitstreamSegment">
        <sequence>
          <element name="Value" type="anySimpleType"/>
        </sequence>
        <attribute name="name" type="gbsd:labelType" use="optional"/>
        <attribute name="marker" type="gbsd:markerType" use="optional"/>
        <attributeGroup ref="gbsd:addressAttributes"/>
        <anyAttribute namespace="##other" processContents="lax"/>
      </extension>
    </complexContent>
  </complexType>

</schema>
```

In subclause 8.3.2.4, bullet 3) i), replace bullet II) with following text:

II) `Consecutive`: The `length` attribute is mandatory whereas the `start` attribute is optional. If the `start` attribute is not present, the segment represented by this `gBSDUnit` starts at the next unit (`bit` or `byte`) after the end of the preceding-sibling element or at the same location than its parent element in case the element is the first child. The length of the segment is given by the value of the `length` attribute. If a start attribute is present, its value (n) provides the offset from the preceding-sibling or parent element as follows: the beginning of the data represented by this `gBSDUnit` element can be located n “units” (`bit` or `byte`) after the end of the previous preceding-sibling element or after the start of the parent element in case it is the first child element.

In subclause 8.3.2.4, bullet 3) ii), replace

In the case when the length attribute of the `Parameter` element specifies a length (L1) longer than the length (L2) implied by the datatype specified by the `xsi:type` attribute of the `Value` element,

with:

In the case when the length attribute of the `Parameter` element is provided and specifies a length (L1) longer than the length (L2) implied by the datatype specified by the `xsi:type` attribute of the `Value` element, *In subclause 8.3.2.5, append the following text:*

EXAMPLE 5 The following example illustrates the usage of the `bs1:datatype` attribute. It shows an excerpt of EXAMPLE 2 with the only difference that the datatype is defined by URI referring to the respective datatype as defined within the XML Schema definition. Thus, any datatype can be used as long as it can be referenced by URI, e.g., by using the namespace and XPointer's shorthand pointer.

```
<!--... and so on ...-->
<Parameter name=":J2K:LMarker" length="2">
  <Value bs1:codec="http://www.w3.org/2001/XMLSchema#unsignedShort">47</Value>
</Parameter>
<Parameter name=":J2K:Rsiz" length="2">
  <Value bs1:codec="http://www.w3.org/2001/XMLSchema#unsignedShort">0</Value>
</Parameter>
<Parameter name=":J2K:Xsiz" length="4">
  <Value bs1:codec="http://www.w3.org/2001/XMLSchema#unsignedInt">768</Value>
</Parameter>
<Parameter name=":J2K:Ysiz" length="4">
  <Value bs1:codec="http://www.w3.org/2001/XMLSchema#unsignedInt">512</Value>
</Parameter>
<!--... and so on ...-->
```

In subclause 8.4.1, replace the following text:

XPath expressions are evaluated relatively to a reference node, called *context node*. BSDL uses variables in the form of XPath expression for three cases:

— Number of occurrences of a particle (`bs2:nOccurs` attribute). In this case, the evaluation of the XPath expression should yield a *number* casted as an integer and the context node is the parent element of the current particle.

— Conditional occurrence of a particle (`bs2:if` attribute). In this case, the evaluation of the XPath expression should yield a *boolean* and the context node is the parent element of the current particle.

— Value of the `bs2:length` facet. In this case, the evaluation of the XPath expression should yield a *number* casted as an integer and the context node is the element, the type of which is constrained by the current facet.

with:

XPath expressions are evaluated relatively to a reference node, called *context node*. BSDL uses variables in the form of XPath expression for several cases:

— Number of occurrences of a particle (`bs2:nOccurs` attribute). In this case, the evaluation of the XPath expression should yield a *number* casted as an integer and the context node is the parent element of the current particle.

— Conditional occurrence of a particle (`bs2:if` attribute). In this case, the evaluation of the XPath expression should yield a *boolean* and the context node is the parent element of the current particle.

— Value of the `bs2:length` and `bs2:bitLength` facets. In this case, the evaluation of the XPath expression should yield a *number* casted as an integer and the context node is the element, the type of which is constrained by the current facet.

— Value of the `bs2:layerLength` attribute. In this case, the evaluation of the XPath expression should yield a *number* casted as an integer and the context node is the current element, i.e. the element whose compexType definition is constrained by the `bs2:layerLength` attribute.

— Value of bs2:startContext, bs2:redefineMarker, bs2: stopContext attributes. In this case, the evaluation of the XPath expression should yield a string. The context node is the current element.

In subclause 8.4.1, append the following text at the end of the section entitled Variables and before the section entitled Facets:

BSDL specifes the `bs2:log2()` function in addition to core XPath function library as follows: this function returns the logarithm in base 2 of the input argument. The input argument and output value are XPath *numbers*. This function is defined in the BSDL-2 namespace and must be used with a prefix bound to the BSDL-2 namespace. The namespace binding context used to resolve the prefixes used in an XPath expressions is the context of the element where this expression is used in the BS Schema.

XPath variables

BSDL supports the use of XPath variables within an XPath expression. Therefore, this part of ISO/IEC 21000 specifes three ways of assigning a value to an XPath variable.

— Firstly, the `bs2:parameter` element allows declaring a constant value for a variable.

— Secondly, the `bs2:assignPost` attribute assigns the value of a simple content element to a variable.

— Thirdly, the `bs2:assignPre` attribute assigns one or several values read directly upstream to variables.

In subclause 8.4.1, replace the following text:

Facets

BSDL-2 introduces a set of new facets to specify constraints on BSDL and XML Schema datatypes. Since XML Schema does not allow a user to add his/her own facets, they are declared in the BSDL-2 namespace and added to the `xsd:restriction` component via the annotation mechanism, i.e., the `xsd:annotation/xsd:appinfo` combination.

The list of BSDL-2 facets consists of `bs2:length`, `bs2:startCode` and `bs2:endCode`. They are used to constrain `xsd:hexBinary`, `xsd:base64Binary` and `bs1:byteRange` built-in datatypes. When applied to `bs1:byteRange`, note that the `xsd:length` facet does not indicate the number of bytes to read from the bitstream, but the number of integers in the datatype, namely always two, corresponding to the start and length information. This is why the *Schema for BSDL-1 extensions* forbids the use of `xsd:length` for further restricting `bs1:byteRange`. Alternatively, the `bs2:length` facet, which is ignored by XML Schema, can and shall be used with `bs1:byteRange` to indicate the length in bytes of the data segment to read.

Note that if a datatype with indefinite length (`xsd:hexBinary`, `xsd:base64Binary` and `bs1:byteRange`) is not constrained by either an `xsd:length`, `bs2:length`, `bs2:startCode` or `bs2:endCode` facet, then the bitstream is parsed until the end of file.

with:

BSDL-2 introduces a set of new facets to specify constraints on BSDL and XML Schema datatypes. Since XML Schema does not allow a user to add his/her own facets, they are declared in the BSDL-2 namespace and added to the `xsd:restriction` component via the annotation mechanism, i.e., the `xsd:annotation/xsd:appinfo` combination.

The list of BSDL-2 facets consists of `bs2:length`, `bs2:startCode`, `bs2:endCode`, `bs2:bitLength`, `bs2:escape` and `bs2:cdata`.

The bs2:length, bs2:startCode and bs2:endCode facets are used to constrain xsd:hexBinary, xsd:base64Binary and bs1:byteRange built-in datatypes. When applied to bs1:byteRange, note that the xsd:length facet does not indicate the number of bytes to read from the bitstream, but the number of integers in the datatype, namely always two, corresponding to the start and length information. This is why the *Schema for BSDL-1 extensions* forbids the use of xsd:length for further restricting bs1:byteRange. Alternatively, the bs2:length facet, which is ignored by XML Schema, can and shall be used with bs1:byteRange to indicate the length in bytes of the data segment to read.

Note that if a datatype with indefinite length (xsd:hexBinary, xsd:base64Binary and bs1:byteRange) is not constrained by either an xsd:length, bs2:length, bs2:startCode or bs2:endCode facet, then the bitstream is parsed until the end of file.

In subclause 8.4.2, replace the text of the whole subclause with following text:

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7/Amd.2 -->
<!-- Schema for Schema for BSDL-2 extensions -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="BSDL-2-AMD2.xsd"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS">

  <annotation>
    <documentation>
      Schema for Schema for BSDL-2 extensions
      This schema introduces new language constructs to be added to XML Schema
      through its two extension mechanisms:
        - attribute with non-schema namespace
        - appinfo
      The attributes and elements declared in this schema are therefore
      **not** validated by XML Schema !!
    </documentation>
  </annotation>

  <!-- #################################### -->
  <!--          BSDL Attributes             -->
  <!-- #################################### -->

  <!-- The following attributes apply to sequence, choice and all, -->
  <!-- element and group when they are not at the schema level     -->
  <attribute name="nOccurs" type="bs2:xPathExpression"/>
  <attribute name="if" type="bs2:xPathExpression"/>

  <!-- The following attributes apply to         -->
  <!-- sequence, choice, all, element and group -->
  <attribute name="ifNext" type="string"/>
  <attribute name="ifNextMask" type="hexBinary"/>
  <attribute name="lookAhead" type="unsignedShort"/>

  <!-- The following attribute applies to schema -->
  <attribute name="rootElement" type="QName"/>
  <attribute name="removeEmPrevByte" type="bs2:hexBinaryStrings"/>
  <attribute name="defaultTreeInMemory" type="boolean" default="true"/>
```

```
<!-- The following attribute applies to element -->
<attribute name="startContext" type="bs2:xPathExpression"/>
<attribute name="partContext" type="boolean" default="false"/>
<attribute name="redefineMarker" type="bs2:xPathExpression"/>

<!--The following attribute applies to element and choice -->
<attribute name="stopContext" type="bs2:xPathExpression"/>

<!-- The following attribute applies to complexType -->
<element name="layerLength" type="bs2:xPathExpression"/>

<!-- The following attribute applies to element and complexType -->
<element name="assignPre" type="string"/>

<!-- The following attribute applies to element -->
<element name="assignPost" type="string"/>

<!-- #################################### -->
<!--            BSDL Facets               -->
<!-- #################################### -->

<element name="length" type="bs2:numFacet"/>
<element name="bitLength" type="bs2:numFacet"/>
<element name="startCode" type="bs2:stringFacet"/>
<element name="endCode" type="bs2:stringFacet"/>
<element name="escape" type="bs2:booleanFacet"/>
<element name="cdata" type="bs2:booleanFacet"/>


<!-- #################################### -->
<!--       BSDL Schema Components         -->
<!-- #################################### -->

<!-- The following component is to be used as as child element   -->
<!-- of xsd:union via the xsd:annotation/xsd:appinfo combination -->
<element name="ifUnion">
  <complexType>
    <complexContent>
      <restriction base="anyType">
        <attribute name="value" type="bs2:xPathExpression" use="required"/>
      </restriction>
    </complexContent>
  </complexType>
</element>

<!-- The following component is to be used as as child element    -->
<!-- of xsd:schema via the xsd:annotation/xsd:appinfo combination -->
<element name="parameter">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="name" type="NCName" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
</element>
```

```
  <!-- #################################### -->
  <!--            Types definitions          -->
  <!-- #################################### -->

  <!-- An XPath expression -->
  <simpleType name="xPathExpression">
    <restriction base="string"/>
  </simpleType>

  <!-- A list of hexBinary strings -->
  <simpleType name="hexBinaryStrings">
    <list itemType="hexBinary"/>
  </simpleType>

  <!-- A facet which value is an XPath expression -->
  <complexType name="numFacet">
    <complexContent>
      <restriction base="anyType">
        <attribute name="value" use="required" type="bs2:xPathExpression"/>
        <attribute name="fixed" use="optional" default="false"
           type="boolean" />
      </restriction>
    </complexContent>
  </complexType>

  <!-- A facet which value is a string -->
  <complexType name="stringFacet">
    <complexContent>
      <restriction base="anyType">
        <attribute name="value" use="required" type="string"/>
        <attribute name="fixed" use="optional" default="false"
           type="boolean"/>
      </restriction>
    </complexContent>
  </complexType>

  <!-- A facet which value is a boolean -->
  <complexType name="booleanFacet">
    <complexContent>
      <restriction base="anyType">
        <attribute name="value" use="required" type="boolean"/>
        <attribute name="fixed" use="optional" default="false" type="boolean"/>
      </restriction>
    </complexContent>
  </complexType>

</schema>
```

In subclause 8.4.3, append the following text at the end of this subclause:

— The `bs2:startContext`, `bs2:partContext` and `bs2:redefineMarker` attributes shall be used for the `xsd:element` component.

— The `bs2:stopContext` attribute shall be used for the `xsd:element` and `xsd:choice` components.

— The `bs2:defaultTreeInMemory` attribute shall be used for the `xsd:schema` component.

— The `bs2:startContext` and `bs2:partContext` attributes shall not be used for the same `xsd:element` component.

— The `bs2:layerLength` attribute shall be used for the `xsd:complexType` component.

— The `bs2:assignPre` attribute shall be used for the `xsd:complexType` component, for the `xsd:element`, and `xsd:group` components not immediately within `xsd:schema`, and for the model groups `xsd:all`, `xsd:choice`, and `xsd:sequence`.

— The `bs2:assignPost` attribute shall be used for the `xsd:element` component.

— The `bs2:parameter` component shall be used within an `xsd:appinfo` component, itself included in an `xsd:annotation`, itself included in an `xsd:schema`.

In subclause 8.4.4, replace the semantics of the ifNext attribute with the following text:

`ifNext`	Attribute specifying a conditional expression on the occurrence of a particle, a global element or group. This expression is evaluated for each potential occurrence of the same particle. It contains one or two strings. The strings may represent a hexadecimal, binary or ASCII string value. A hexadecimal value is preceded by "0x", a binary value is preceded by "0b", and string values are wrapped in simple or double quotes (' or ") depending on what character is used for the attribute value. For backwards compatibility, if the string is not wrapped in quotes, does not start with "0b" or "0x", then the value is assumed to be provided in hexadecimal format. In case two values are specified, they define a range of possible values indicated by the minimum and maximum boundaries. For ASCII strings, the considered value is its ASCII code. If one string is specified, then the particle is parsed if the sequence of next bytes downstream is equal to the given value. If two strings are specified, then the particle is parsed if the sequence of next bytes is within the given range. If a `bs2:lookAhead` attribute is specified for the same particle, BintoBSD first skips the indicated number of bytes before performing the test. If a `bs2:nextMask` attribute is specified for the same particle, BintoBSD first applies the indicated mask to the next bytes before performing the test, and after skipping the potential number of bytes indicated by `bs2:lookAhead`.

In subclause 8.4.4, append the following text at the end of this subclause as follows:

`startContext`	The `bs2:startContext` attribute indicates that the element should be kept in memory as it is needed to evaluate a forthcoming XPath expression. If the element is conditional, the element will be kept in memory after a positive evaluation of the condition. Its value is an XPath expression that should be resolved in the in-memory stored BS Description as a string. The resulting string is used to identify the corresponding element.
`partContext`	When true, the `bs2:partContext` attribute indicates that the element should be kept in memory as it is needed to evaluate a forthcoming XPath expression. If the element is conditional, the element will be kept in memory after a positive evaluation of the condition. Note, this attribute is typically used when the element is necessary in the location step of an XPath expression.
`stopContext`	The `bs2:stopContext` attribute lists XPath expressions and every appearing XPath expression should be resolved in the in-memory stored BS Description as a string. The resulting strings identify the elements that must be removed from the memory. If the attribute is used for an `xsd:element`, the attribute is always evaluated, even if the element is conditional. If the attribute is used for an `xsd:choice`, the attribute is evaluated after the evaluation of the choice. Note, the `bs2:stopContext` attribute must be evaluated after the processing of any other context-related attribute.
`redefineMarker`	The `bs2:redefineMarker` attribute associates a new identifier to a previously identified element. If the element is conditional, the `bs2:redefineMarker` attribute is only processed after a positive evaluation of the condition. The `bs2:redefineMarker` lists couples of XPath expressions whereby every XPath expression should be resolved in the in-memory stored BS Description as a string. The first string evaluated of each couple is the existing identifier of the element and the second string evaluated of the couple is the new identifier.
`defaultTreeInMemory`	The `bs2:defaultTreeInMemory` provides an indication about the required memory for parsing. The default `boolean` value of True indicates that all the elements should be kept in memory. A value of False signals the BintoBSD Parser that memory consumption may be optimized by using the context-related attributes. NOTE This attribute may be also used without context information. For example, when the BS Schema contains no XPath expression or the XPath expressions used in the BS Schema do not use location paths, BintoBSD does not need storing the full BSD in memory. This may be indicated by this attribute.

`lookAhead`	The `bs2:lookAhead` attribute indicates the number of bytes to skip in the input bitstream prior to reading the next bytes tested for the `bs2:ifNext` and `bs2:ifNextMask` attributes.
`ifNextMask`	The `bs2:lookAhead` attribute indicates a binary mask to apply to the next bytes in the bitstream (after potentially skipping the number of bytes indicated by the `bs2:lookAhead`). The resulting value is then tested against the value of `bs2:ifNext`.
`remEmPrevByte`	Specifies the removal of emulation prevention bytes. It contains an even number of hexadecimal strings. For each pair of hexadecimal strings, the first string specifies the byte sequence from which the emulation byte should be removed and the second string specifies the byte sequence that should be read from the input bitstream instead. When the attribute is empty, no emulation prevention byte is discarded.
`assignPost`	The `bs2:assignPost` attribute assigns the lexical value of the bitstream field that has been read to the indicated variable. Its value is a variable reference. The variable assignement is performed after the processing of the element.
`assignPre`	The `bs2:assignPre` attribute assigns the lexical value of one or several bitstream fields indicated by their offset and length to the indicated variables. Its value is one or several tuples of string tokens. Each tuple contains three string tokens separated by white spaces. The first token indicates the variable name (non colonized name). The second and third tokens respectively indicate the offset and length in bits localizing the bitstream field. The bitstream field is parsed as an unsigned integer value. The variable assignement is performed prior the processing of the particle or complex type and prior the evaluation of potential `bs2:if` or `bs2:nOccurs` attributes.
`parameter`	A `bs2:parameter` component declares an XPath variable and assigns a constant value to it. This value may be overwritten at run-time by a value provided, e.g., by command line.

In subclause 8.4.5, replace the semantics of the startCode and endCode attributes with the following text:

`startCode` `endCode`	Facets constraining the data read by `xsd:hexBinary`, `xsd:base64Binary` and `bs1:byteRange` by specifying a flag or a range of flags until which the bitstream should be read. They contain one or two strings. The strings may represent a hexadecimal, binary or ASCII string value. A hexadecimal value is preceded by "0x", a binary value is preceded by "0b", and string values are wrapped in simple or double quotes (' or ") depending on what character is used for the attribute value. For backwards compatibility, if the string is not wrapped in quotes, does not start with "0b" or "0x", then the value is assumed to be provided in hexadecimal format. In case two values are specified, they define a range of possible values, including the minimum and maximum boundaries. For ASCII strings, the considered value is its ASCII code. If one string is specified, then the bitstream is parsed until the sequence of next bytes downstream is equal to the given value. If two strings specified, then the bitstream is parsed until the sequence of next bytes is within the given range. For `bs2:startCode`, the flag is exclusive, i.e., the bitstream is parsed up to the last byte *before* the given sequence of byte. For `bs2:endCode`, the flag is inclusive, i.e., the bitstream is parsed up to the last byte of the sequence. Several `bs2:startCode` and `bs2:endCode` facets may simultaneously be used in the same `xsd:restriction`. In this case, the bitstream is parsed until one of the flag is found. On the hand, it should not be used simultaneously with the `xsd:length` or `bs2:length` facets.

In subclause 8.4.5, append the following text at the end of this subclause:

`escape`	Facet constraining string datatypes and indicating to the BintoBSD processor that control characters should be escaped as an XML character reference in the output lexical value. For example, a line feed character read in the bitstream should be output as "&#x0A;"
`cdata`	Facet constraining string datatypes and indicating to the BintoBSD processor that the output lexical value should be marked as character data, *i.e.*, encapsulated in "`<![CDATA[`" and "`]]>`".
`layerLength`	Attribute constraining the overall length of complex type. Its value is an XPath expression that should be evaluated as an integer indicating the size in bytes of the bitstream segment described by the complex type.
`bitLength`	Facet constraining integer datatypes and indicating the number of bits to be read from the bitstream. Its value is an XPath expression that should be evaluated as an integer.

	BintoBSD shall instantiate a BSD element with a `xsi:type` attribute overriding the base type with one of the `bs1:b1` to `bs1:b32` datatypes according to the value obtained by the XPath expression evaluation. The result is not specified if the resulting value is greater than 32.

In subclause 8.4.8, append the following text at the end of the subclause:

The `bs2:redefineMarker` attribute shall be processed after the evaluation of a `bs2:startContext` or a `bs2:partContext` attribute, but before the evaluation of a `bs2:stopContext` attribute.

When an element is declared as abstract, BintoBSD shall consider all elements that are members of that element substitution group and shall instantiate the first possible substitution element in schema document order.

NOTE Some XML Schema components have no semantics with respect to BSDL and thefore should be ignored by the BSDtoBin and BintoBSD processors. This includes in particular `xsd:key`, `xsd:keyref`, `xsd:unique`, `xsd:selector`, `xsd:field` and `xsd:notation`.

In subclause 8.4.9, append the following text at the end of the subclause:

EXAMPLE 6 The following example of a BS schema for a fictive and simple coding format demonstrates the usage of the `bs2:startContext`, `bs2:partContext`, `bs2:stopContext`, `bs2:redefineMarker`, and `bs2:defaultTreeInMemory`.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema targetNamespace="example"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ex="example"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  elementFormDefault="qualified" bs2:rootElement="ex:bitstream"
  bs1:defaultTreeInMemory="false">
  <xsd:include schemaLocation="UnsignedIntegers.xsd"/>
  <xsd:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL0-NS"
              schemaLocation="BSDL-0.xsd"/>
  <xsd:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
              schemaLocation="BSDL-1.xsd"/>
  <xsd:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
              schemaLocation="BSDL-2.xsd"/>
  <!-- Start element -->
  <xsd:element name="bitstream" bs2:startContext="'bitstream'">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="header" bs2:nOccurs="1"/>
        <xsd:element ref="frame" bs2:nOccurs="unbounded"/>
      </xsd:sequence>
      <xsd:attribute ref="bs1:bitstreamURI"/>
    </xsd:complexType>
  </xsd:element>
  <!-- Definition of the header -->
  <xsd:element name="header" bs2:partContext="true">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="height" type="b6"/>
        <xsd:element name="width" type="b6"/>
        <xsd:element name="color" type="b3" bs2:partContext="true"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- Definition of a frame -->
  <xsd:element name="frame" bs2:startContext="'frame'"
bs2:stopContext="'oldFrame'">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="type" type="b4" bs2:startContext="'type'"
                     bs2:redefineMarker="'frame' 'oldFrame'"/>
        <xsd:element name="ifTypeIs1" type="xsd:unsignedByte"
                     bs2:if="./ex:type=1" bs2:stopContext="type"/>
        <xsd:element name="ifColorIs2" type="xsd:unsignedByte"
                     bs2:if="/ex:bitstream/ex:header/ex:color = 2"/>
        <xsd:element ref="stream"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <!-- Definition of a stream -->
  <xsd:element name="stream" bs2:startContext="'stream'">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="length" type="'xsd:unsignedByte"
                     bs2:partContext="true"/>
        <xsd:element name="payload">
          <xsd:simpleType>
            <xsd:restriction base="bs1:byteRange">
              <xsd:annotation>
                <xsd:appinfo>
                  <bs2:length value="./ex:length"/>
                </xsd:appinfo>
              </xsd:annotation>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element name="stopByte" type="xsd:unsignedByte"
                     bs2:stopContext="'stream'"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

In this example, one can see the functioning of the different attributes and how they steer the BintoBSD Parser. The impact of the attributes on the internal representation of the description is visualized in Figure AMD2.1. In Figure AMD2.1 (a), the internal representation of the description is given after having parsed a frame; in Figure AMD2.1 (b), the internal representation of the description is given before a payload element is parsed. The internal representation of the description is small during the complete parsing process, i.e. it will not continue to grow during processing which results in a constant memory consumption and a constant generation speed.

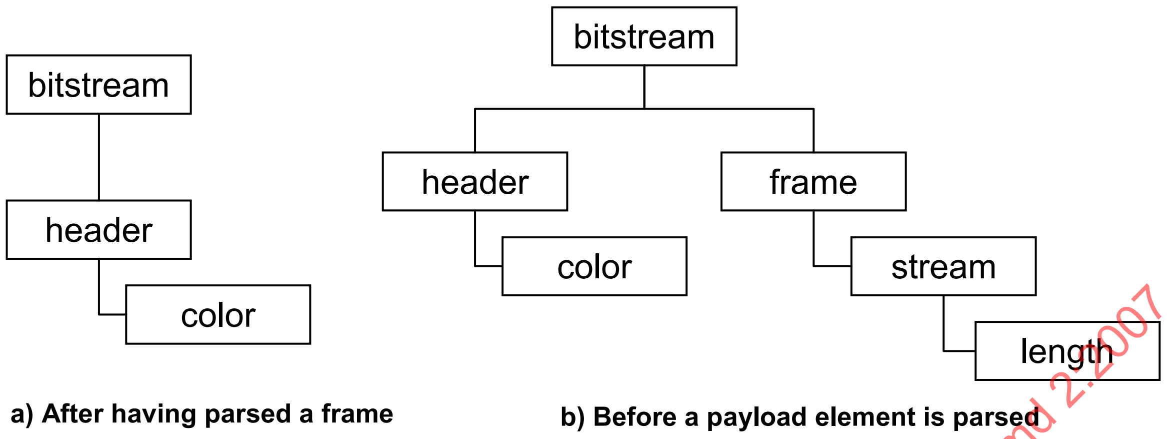
bitstream
header
color
a) After having parsed a frame
bitstream
header
frame
color
stream
length
b) Before a payload element is parsed

Figure AMD2.1 — Internal representation of the description by using context-related attributes

Insert new subclauses 8.5, 8.6 and 8.7 as follows:

8.5 Media resource streaming instructions

8.5.1 Introduction

The media resource streaming instructions specified in this subclause provide the information required for streaming a bitstream described by a BSD and possibly adapted on-the-fly, namely the indication of access units and their associated time information.

Access units are the fundamental units for transport of media resource streams. They are defined in this specification as the smallest data entity which is atomic in time, *i.e.*, to which a single decoding time can be attached. The media resource streaming instructions allow annotating a BSD in a coding format independent way to indicate the location of the access units in the described bitstream. Additionally, they indicate which access units are random access points, *i.e.*, can be decoded independently from previous access units.

In some cases, an access unit needs to be sub-divided into several parts in order to fit the requirements of the delivery channel. For example, an access unit needs to be fragmented into several Real-time Transport Protocol (RTP) packets when its size exceeds what is allowed by the Maximum Transmission Unit (MTU). For this purpose, some RTP payload formats propose a specific fragmentation based on the semantics of the bitstream. For example, for ISO/IEC 14496-2 visual streams, the RFC 3016 recommends to send a single video packet as a single RTP packet. The media resource streaming instructions provide a way for indicating access unit parts within an access unit in a coding format independent way. In this way, a streaming server that is not aware of the format of the streamed media resource may nevertheless meet the requirements of a specific RTP payload format.

Lastly, media resource streaming instructions provide information about the decoding and composition time stamps (DTS and CTS) of the access units. This information is widely used for streaming media resources such as ISO/IEC 13818 and 14496 visual streams.

For the purpose as mentioned above, this subclause introduces a set of abstract properties and XML attributes used for annotating the BSD.

The properties are abstract in the sense that they do not appear in the XML document, but augment the element information item in the document infoset. They can be specified statically by a set of XML attributes and dynamically by the so-called Properties Style Sheet specified in subclause 8.7. Additionally, an inheritance mechanism is defined for some of these properties: the value of the property is then inherited by all descendant elements until the property is defined with a different value which then supersedes the inherited value, and is itself inherited by the descendants. Lastly, a default value is specified for each property.

The property of an element information item is specified as follows:

— If a Properties Style Sheet is applied to the document, and if the element matches the pattern of one or more templates, the value indicated by the relevant template applies.

— Otherwise, if the element contains an attribute specifying the property, the attribute value applies.

— Otherwise, if the property is inheritable, its value is inherited from the parent element.

— Otherwise, the property takes the specified default value.

Subclause 8.5.2 specifies a set of properties for annotating a BSD to indicate:

— the access units, access unit parts and random access points of the described media resource bitstream.

— the decoding and composition time stamps (DTS/CTS) of the media resource access units.

Subclause 8.5.3 specifies a set of XML attributes for setting these properties.

NOTE Media resource streaming instructions specified in this subclause can be used to annotate both BSDs and gBSDs. For simplicity, the text only mentions BSD.

8.5.2 Properties

The media resource streaming instructions specify two sets of properties for annotating a BSD. The first set indicates the access units in the described bitstream, the random access points, and the subdivision into access unit parts. The second set provides the access unit time stamps.

A BSD linearly describes a bitstream. Each BSD element therefore corresponds to a point in the described bitstream. The start of an access unit is indicated by an element with an `au` property set to true. This element is named **anchor element**. The extent of the access unit then depends on the value of the `auMode` property of the anchor element. In the sequential mode, the access unit extends until a new element is found with an `au` property set to false or true. In the latter case (i.e., new anchor element), a new access unit immediately follows. If no element is found with an `au` property set to true or false, the access unit extends until the end of the bitstream. In the tree mode, the access unit is the bitstream segment described by the BSD sub-tree below the element flagged with the `au` property set to true.

Access unit parts are defined in a similar way. The start of a new access unit part in an access unit is indicated by an `auPart` property set to true and the extent is specified by the `auMode` property. In the sequential mode, the access unit part extends until a new element has an `auPart` property set to false or true (in the latter case, a new access unit part immediately follows), until the end of the access unit, or until the end of the bitstream. In the tree mode, the access unit part is the bitstream segment corresponding to the sub-tree below the element flagged by the `auPart` property. It is not necessary to indicate the first access unit part since it is supposed to start with the access unit.

Other information about access units is specified by the properties of the anchor element. In particular, the access unit is a random access point if the `rap` property of the anchor element is set to true. The `rap` property is inheritable, and it is therefore possible to factorize the information by setting the `rap` property of the BSD root element to true. The value of the property is then inherited by all descendant elements, including the anchor elements indicating the access units.

The time information of the access unit (CTS and DTS) is also specified by the properties of the anchor element as explained below.

The media resource streaming instructions use an absolute and a relative mode for specifying time information. In absolute mode, the CTS and DTS of an access unit are specified independently from other access units. In relative mode, the CTS and DTS are calculated relatively to the CTS and DTS of the previous access unit. Both modes can be used in a same document. For example, an absolute date can be applied to a given access unit, and the CTS and DTS of the following access units are calculated relatively to this access unit. In both modes, CTS and DTS are specified relatively to a time scale, *i.e.* they are specified as a number of ticks, where the duration of a tick is specified by the time scale which indicates the number if ticks per second. The scale is specified by the `timeScale` property.

In the absolute time mode, two properties `cts` and `dts` define the CTS and DTS of the access unit, expressed as an integer number of ticks. They are not inheritable and may be applied to an anchor element for specifying the CTS and DTS of the corresponding access unit.

In the relative time mode, two properties named `dtsDelta` and `ctsOffset` allow calculating the DTS and CTS of the access unit relatively to the previous access unit. The `dtsDelta` property indicates the time interval in ticks between the current access unit and the next one. The `ctsOffset` property indicates the time interval in ticks between the DTS and the CTS of the current access unit. The DTS of the first access unit is 0 by default.

NOTE Some media resources do not require a CTS information. In this case, the `cts` and `ctsOffset` properties are not used and may be undefined.

Table AMD2.1 — Table of media resource streaming instructions properties

Property name	Property value	Is inherited	Default value
`auMode`	{`tree`, `sequential`}	yes	`tree`
`au`	{`undefined`, `false`, `true`}	no	`undefined`
`auPart`	{`undefined`, `false`, `true`}	no	`undefined`
`rap`	{`undefined`, `false`, `true`}	yes	`undefined`
`timeScale`	{`undefined`, an integer value}	yes	`undefined`
`dts`	{`undefined`, an integer value}	no	`undefined`
`cts`	{`undefined`, an integer value}	no	`undefined`
`dtsDelta`	{`undefined`, an integer value}	yes	`undefined`
`ctsOffset`	{`undefined`, an integer value}	yes	`undefined`

8.5.3 Attributes

8.5.3.1 Introduction

Subclause 8.5.3 specifies a set of XML attributes with normative syntax and semantics associated to the properties defined in subclause 8.5.2. An attribute can be used to set a property for a given element.

NOTE 1 It is reminded that XML does not define any inheritance mechanism for an attribute. In other words, if an attribute is used for a given element, its descendant elements are agnostic about this attribute. Alternatively, the properties introduced in 8.5.1 are inheritable.

NOTE 2 These attributes are not the only means for setting properties, i.e., the property of an element may also be set by a properties stylesheet, inherited from the parent element or get a default value.

8.5.3.2 Syntax

```
<?xml version="1.0"?>
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="MSI-AMD2.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:msi="urn:mpeg:mpeg21:2003:01-DIA-MSI-NS"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-MSI-NS"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Declaration of attributes used for media resource streaming instructions
    </documentation>
  </annotation>

  <!-- The following attribute indicates the access unit mode -->
  <attribute name="auMode" type="msi:auModeType"/>
  <simpleType name="auModeType">
    <restriction base="string">
      <enumeration value="tree"/>
      <enumeration value="sequential"/>
    </restriction>
  </simpleType>

  <!-- The following attribute indicates media resource access units -->
  <attribute name="au" type="boolean"/>

  <!-- The following attribute indicates media resource access unit part -->
  <attribute name="auPart" type="boolean"/>

  <!-- The following attribute indicates Random Access Points -->
  <attribute name="rap" type="boolean"/>

  <!-- The following attributes define the time properties -->
  <attribute name="timeScale" type="unsignedInt"/>

  <attribute name="cts" type="unsignedInt"/>
  <attribute name="dts" type="unsignedInt"/>

  <attribute name="dtsDelta" type="unsignedInt"/>
  <attribute name="ctsOffset" type="unsignedInt"/>

</schema>
```

8.5.3.3 Semantics

Name	*Definition*
`auMode`	Attribute defining the `auMode` property.
`au`	Attribute defining the `au` property.
`auPart`	Attribute defining the `auPart` property.

Name	*Definition*
`rap`	Attribute defining the `rap` property.
`timeScale`	Attribute defining the `timeScale` property.
`dts`	Attribute defining the `dts` property.
`cts`	Attribute defining the `cts` property.
`dtsDelta`	Attribute defining the `dtsDelta` property.
`ctsOffset`	Attribute defining the `ctsOffset` property.

8.5.4 Processing

In off-line, BSD-based adaptation scenarios, the normative BSDtoBin and gBSDtoBin processors specified in subclauses 8.3.1.6 and 8.3.2.4 are used to generate an adapted bitstream from the (possibly transformed) BSD. However, in order to stream the generated bitstream, additional information is required to locate the access units and indicate their decoding and composition time stamps. This information can typically be provided by the sample tables of the ISO/IEC 14496-12 file format, but is no longer valid for an adapted bitstream. In particular, the sample offsets in the elementary stream or their DTS/CTS may have changed.

This subclause specifies two normative processors named BSDtoBinAU and gBSDtoBinAU which respectively extend the BSDtoBin and gBSDtoBin processors by providing information about access units in the output bitstream and their associated time stamps. In particular, these processors may be used by a server streaming a media resource adapted on-the-fly based on its BSD.

The behavior of the BSDtoBinAU and gBSDtoBinAU is identical. For conciseness purpose, the description below only mentions BSDtoBinAU.

The input of the BSDtoBinAU processor is the input bitstream and its BSD (gBSD for gBSDtoBinAU), plus media resource streaming instructions. Media resource instructions may be provided as attributes embedded in the BSD and/or as an external Properties Stylesheet, as pictured in Figure AMD2.2.

The output of the BSDtoBinAU processor is the output bitstream as produced by BSDtoBin, framed into access units and possibly access unit parts with associated time stamps. Moreover, a new description with the updated address information usually needs to be generated, for example in multi-step/distributed adaptation scenarios. This updating mechanism is not normative, but an example process is proposed in the informative Annex D.

NOTE 1 When using media resource streaming instructions in combination with XML streaming instructions in order to realize a distributed adaptation framework, the input to the BSDtoBinAU processor can consist of BSD PUs instead of a BSD which described the complete bitstream. The output would then be an updated BSD PU (according to Annex D) which can be encoded and streamed together with the adapted AU. Please refer to Annex P for a detailed description of the combined usage of XML streaming instructions and media resource streaming instructions.

The production of the output bitstream is identical to BSDtoBin. Additionally, the BSDtoBinAU processor defines the following set of properties for each access unit:

— `DTS` and `CTS`: Decoding and composition time stamp of the access unit, expressed as an integer number of ticks.

— `DTS_DELTA`: interval between the current access unit and the next one, expressed as an integer number of ticks.

— TIME_SCALE: time scale, i.e. number of ticks per second used for DTS, CTS and DTS_DELTA.

— RAP: random access point flag.

NOTE 2 These properties characterize access units and should not be confused with the media resource streaming instructions properties.

NOTE 3 Depending on the media resource streaming instructions provided, some values may remain undefined.

Additionally, the information about access units and their time stamps is calculated from the media streaming instructions properties, as specified in the subclause 8.5.2. This information consists in the Decoding and Composition Time Stamps (DTS/CTS), time scale, DTS delta and random access point flag. DTS, CTS and DTS Delta are provided as an integer number of ticks, where the number of ticks per second is provided by the time scale. The DTS delta specifies the interval in ticks between the current and next access unit. In the following, the following notation is used for referring to the information of the nth access unit: DTS(n), CTS(n), TIME_SCALE(n), DTS_DELTA(n) and RAP(n).

A BSD is parsed in a depth-first order. The media resource streaming instructions properties are calculated as specified in the subclause 8.5.2. Anchor elements (i.e., elements with the au property set to true) are ordered according to the elements scan order and so are the corresponding access units. An anchor element indicates the start of an access unit, the extent of which is specified by the auMode property.

The following pseudo-code specifies how to calculate the properties of each Access Unit according to the Media Resource Streaming Instructions properties.

For each anchor element, the properties of the corresponding access unit are then calculated as follows:

```
if isDefined(dts(n)) {
    DTS(n) = dts(n)
} else {
 if n = 0 {  // i.e., first access unit
   DTS(n) = 0
 } else {
   DTS(n) = ((DTS(n-1) + DTS_DELTA(n-1))/TIME_SCALE(n-1)) * TIME_SCALE(n)
 }
}
if isDefined(cts(n)) {
    CTS(n) = cts(n)
} else {
    CTS(n) = DTS(n) + ctsOffset(n)
}
TIME_SCALE(n) = timeScale(n)
DTS_DELTA(n) = dtsDelta(n)
RAP(n) = rap(n)
```

Where dts(n), cts(n), timeScale(n), dtsDelta(n), ctsOffset(n), rap(n) represent the media resource streaming instruction properties of the nth anchor element, and DTS(n), CTS(n), TIME_SCALE(n), DTS_DELTA(n) and RAP(n) represent the properties of the associated nth access unit.

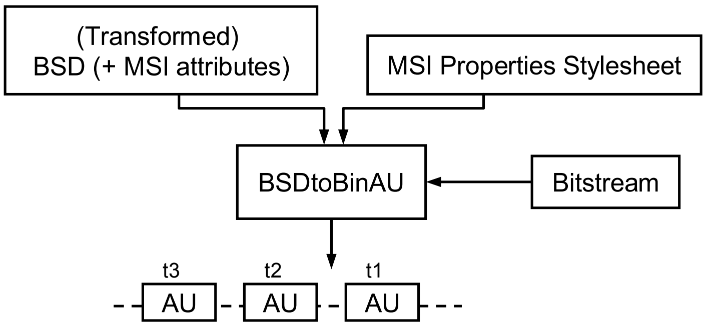

Figure AMD2.2 — Processing related to media resource streaming instructions

8.5.5 Examples

Examples of media resource streaming instructions can be found in Annex O.

8.6 XML streaming instructions

8.6.1 Introduction

The XML streaming instructions allow firstly to identify process units in an XML document and secondly to assign time information to them.

A process unit is a set of connected XML elements. It is specified by one element named anchor element and by a process unit mode indicating how other connected elements are aggregated to this anchor to compose the process unit. Depending on the mode, the anchor element is not necessarily the root of the process unit. Anchor elements are ordered according to the navigation path of the XML document. Process units may overlap, i.e. some elements (including anchor elements) may belong to several process units. Additionally, the content provider may require that a given process unit be encoded as a random access point, i.e. that the resulting access unit does not require any other access units to be decoded.

Subclause 8.6 specifies a set of properties and XML attributes for

— fragmenting an XML document into process units,

— indicating which process units shall be encoded as random access point,

— assigning time information (*i.e.*, processing time stamp) to these process units.

Properties are understood and set as in subclause 8.5.1.

8.6.2 Properties

Subclause 8.6.2 defines two sets of properties: a first set specifying how an XML document is fragmented into process units and a second set specifying how processing time stamps are assigned to these process units.

The properties related to fragmentation are specified as follows:

— `anchorElement` is one of {`undefined`, `false`, `true`}:

  — When set to true, this property indicates an anchor element.

  — This property *is not* inherited and its default value is `undefined`.

A process unit inherits its time properties from the time properties of its anchor element.

— `puMode` is one of {`undefined`, `self`, `ancestors`, `descendants`, `ancestorsDescendants`, `preceding`, `sequential`}:

  — The process unit mode property specifies how elements are aggregated to the anchor element to compose a process unit.

  — The semantics of the values are as follows:

    — `self`: the PU contains only the anchor element.

    — `ancestors`: the PU contains the anchor element and its ancestors stack, i.e., its ancestor elements.

    — `descendants`: the PU contains the anchor element and its descendant elements.

    — `ancestorsDescendants`: the PU contains the anchor element, its ancestor, and descendant elements.

    — `preceding`: the PU contains the anchor element, its descendant and parent elements, and all the preceding-sibling elements of its ancestor elements and their descendants.

    — `precedingSiblings`: the PU contains the anchor element, its descendant and parent elements, and all the preceding-sibling elements (and their descendants) of its ancestor elements.

    — `sequential`: the PU contains the anchor element, its ancestors stack, and all the following elements (i.e., descendants, following siblings, and their ancestors) until a next element is flagged as an anchor element.

  — This property *is* inherited and its default value is `undefined`.

EXAMPLE The following figures show examples of process unit definition following the `puMode` property. The anchor elements are pictured in white.

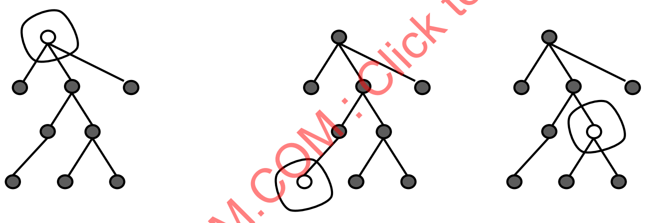

Figure AMD2.3 — Examples of process units - `puMode = self`

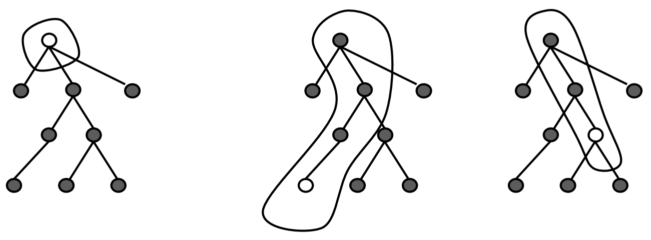

Figure AMD2.4 — Examples of process units - `puMode = ancestors`

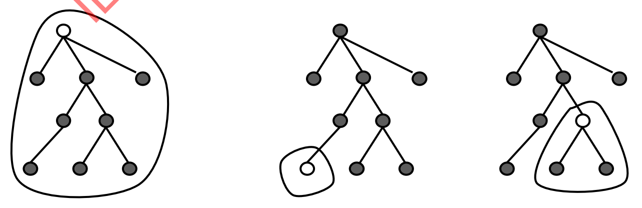

Figure AMD2.5 — Examples of process units - `puMode = descendants`

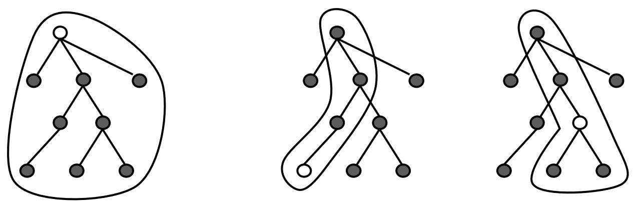

Figure AMD2.6 — Examples of process units - `puMode = ancestorsDescendants`

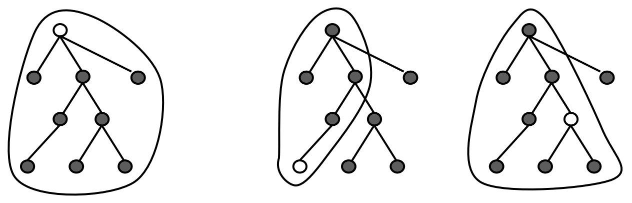

Figure AMD2.7 — Examples of process units - `puMode = preceding`

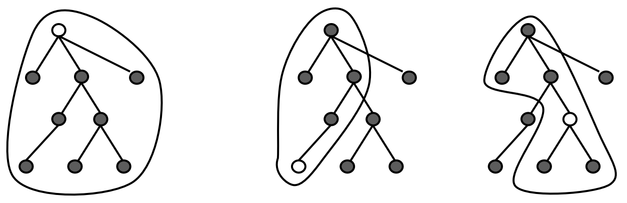

Figure AMD2.8 — Examples of process units - `puMode = precedingSibling`

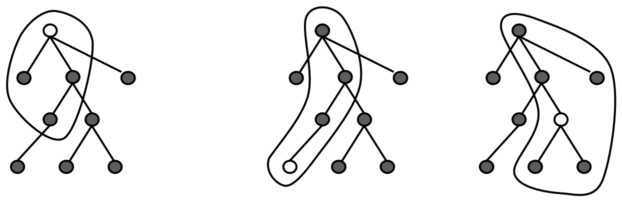

Figure AMD2.9 — Examples of process units - `puMode = sequential`

— `encodeAsRAP` is one of {`undefined`, `false`, `true`}:

  — When set to `true`, this property indicates that the process unit shall be encoded as a random access point.

  — This property *is* inherited and its default value is `undefined`.

Table AMD2.2 — Table of fragmentation-related properties

Property name	Property value	Is inherited	Default value
`anchorElement`	{`undefined`, `false`, `true`}	no	`undefined`
`puMode`	{`undefined`, `self`, `ancestors`, `descendants`, `ancestorsDescendants`, `preceding`, `precedingSiblings`, `sequential`}	yes	`undefined`
`encodeAsRap`	{`undefined`, `false`, `true`}	yes	`undefined`

8.6.2.3 Properties related to time information

The processing time stamp of a process unit defines the point in time where it is available to the application for consumption. It is specified by the set of time properties of the PU's anchor element. There are two defined time modes: an **absolute time mode** where the PUs' PTS are specified according to a same time origin and a **relative time mode** where they are defined relatively to the PTS of the previous PU.

The properties related to time information are specified as follows:

— `timeScale` is one of {`undefined`, an integer value}:

  — Specifies the time scale, i.e., the number of ticks per second.

  — This property *is* inherited and its default value is `undefined`.

— `ptsDelta` is one of {`undefined`, an integer value}:

  — The `ptsDelta` property specifies the relative interval in time ticks after the preceding anchor element.

  — The default value for the processing time stamp of the first anchor element is 0.

  — This property *is* inherited and its default value is `undefined`.

— `absTimeScheme` is one of {`undefined`, a string value}:

  — The `absTimeScheme` property specifies the absolute time scheme used.

  — This property *is* inherited and its default value is `undefined`.

EXAMPLE 1 Example time schemes are specified in other standards such as SMPTE or MPEG-7. For example, the value 'mp7t' identifies the MPEG-7 time scheme. See Annex C of ISO/IEC 21000-17:2006 for other examples of time schemes syntax.

— `absTime` is one of {`undefined`, a string value}:

  — The `absTime` property specifies the absolute time of the anchor element. Its syntax and semantics are specified according to the time scheme used (`absTimeScheme` property).

  — This property *is not* inherited and its default value is `undefined`.

EXAMPLE 2 In the MPEG-7 time scheme quoted above, an example absolute time value is 'T17:30:45:2F10'.

— `pts` is one of {`undefined`, an integer value}:

  — The `pts` property specifies the absolute time of the anchor element as the number of ticks since the origin.

  — This property *is not* inherited and its default value is `undefined`.

Table AMD2.3 — Table of time-related properties

Property name	Property value	Is inherited	Default value
`timeScale`	{`undefined`, an integer value}	yes	`undefined`
`ptsDelta`	{`undefined`, an integer value}	yes	`undefined`
`absTimeScheme`	{`undefined`, a string value}	yes	`undefined`
`absTime`	{`undefined`, a string value}	no	`undefined`
`pts`	{`undefined`, an integer value}	no	`undefined`

8.6.3 Attributes

8.6.3.1 Introduction

Subclause 8.6.3 specifies a set of XML attributes with normative syntax and semantics associated to the properties defined in subclause 8.6.2. An attribute can be used to set a property for a given element.

NOTE 1 It is reminded that XML does not define any inheritance mechanism for an attribute. In other words, if an attribute is used for a given element, its descendant elements are agnostic about this attribute. Alternatively, the properties introduced in 8.5.1 are inheritable.

NOTE 2 These attributes are not the only means for setting properties, i.e., the property of an element may also be set by a properties stylesheet, inherited from the parent element or get a default value.

8.6.3.1 Syntax

```
<?xml version="1.0"?>
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="XSI-AMD2.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
  elementFormDefault="qualified">

  <annotation>
    <documentation>
      Declaration of attributes used for XML streaming instructions
    </documentation>
  </annotation>

  <!-- The following attribute defines the process units -->
  <attribute name="anchorElement" type="boolean"/>

  <!-- The following attribute indicates that the PU shall be encoded as Random
Access Point -->
  <attribute name="encodeAsRAP" type="boolean"/>

  <attribute name="puMode" type="si:puModeType"/>
  <simpleType name="puModeType">
    <restriction base="string">
      <enumeration value="self"/>
      <enumeration value="ancestors"/>
      <enumeration value="descendants"/>
      <enumeration value="ancestorsDescendants"/>
      <enumeration value="preceding"/>
      <enumeration value="precedingSiblings"/>
      <enumeration value="sequential"/>
    </restriction>
  </simpleType>

  <!-- The following attributes define the time properties -->
  <attribute name="timeScale" type="unsignedInt"/>

  <attribute name="ptsDelta" type="unsignedInt"/>

  <attribute name="absTimeScheme" type="string"/>

  <attribute name="absTime" type="string"/>

  <attribute name="pts" type="nonNegativeInteger"/>

</schema>
```

8.6.3.2 Semantics

Name	*Definition*
`anchorElement`	Attribute defining the `anchorElement` property.
`puMode`	Attribute defining the `puMode` property.
`encodeAsRap`	Attribute defining the `encodeAsRap` property.
`timeScale`	Attribute defining the `timeScale` property.
`ptsDelta`	Attribute defining the `ptsDelta` property.
`absTimeScheme`	Attribute defining the `absTimeScheme` property.
`absTime`	Attribute defining the `absTime` property.
`pts`	Attribute defining the `pts` property.

8.6.4 Processing

This subclause specifies a normative processor named *Fragmenter*. This processor uses as input the XML document to be streamed and a set of XML streaming instructions provided either internally (as attributes) and/or externally (with a properties stylesheet). The output of the Fragmenter is a set of timed process units.

It is possible to use XML streaming instructions provided both as internal attributes and with a properties stylesheet. In case of conflicts, the priority rules are detailed in subclause 8.7.

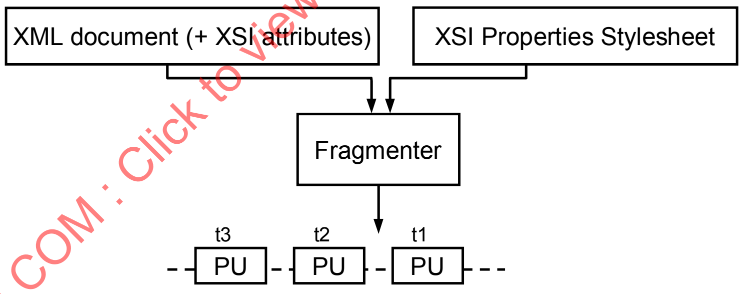

Figure AMD2.9 — Normative processing related to XML streaming instructions

The Fragmenter process is specified as follows:

The Fragmenter parses the XML document in a depth-first order. XML streaming instructions properties are computed as explained in subclause 8.5.1. An element with the `pu` property set to true indicates an anchor element and a new process unit. The process unit then comprises connected elements according to the `puMode` property of the anchor element.

NOTE 1 In the case that an anchorElement attribute is parsed before any puMode attribute, the behaviour of the Fragmenter is undefined.

Additionally, the Fragmenter defines a set of properties for the process unit:

— `ENCODE_AS_RAP`: specifies that the PU shall be encoded as a random access point.

— `PTS`: processing time stamp, expressed as an integer number of ticks.

— `PTS_DELTA`: interval between the current process unit and the next one, expressed as an integer number of ticks.

— `TIME_SCALE`: time scale, i.e. number of ticks per second used for `PTS` and `PTS_DELTA`.

NOTE 2 These properties characterize process units and should not be confused with the XML streaming instructions properties.

The following pseudo-code specifies how to calculate the properties of each process unit according to the XML Streaming Instructions properties.

For each anchor element, the properties of the corresponding process unit are then calculated as follows:

```
if isDefined(absTime(n)) {
  PTS(n) = absTime(n) * timeScale(n)
  // absTime is specified according to the time scheme specified by the value
  // of the absTimeScheme property and converted in seconds.

} else if isDefined(pts(n)) {
  PTS(n) = pts(n)
} else {
  if n = 0 {  // i.e., first process unit
    PTS(n) = 0
  } else {
    PTS(n) = ((PTS(n-1) + PTS_DELTA(n-1))/TIME_SCALE(n-1)) * TIME_SCALE(n)
  }
}
TIME_SCALE(n) = timeScale(n)
PTS_DELTA(n) = ptsDelta(n)
ENCODE_AS_RAP(n) = encodeAsRap(n)
```

Where `absTime(n)`, `pts(n)`, `timeScale(n)`, `ptsDelta(n)` and `encodeAsRap(n)`, represent the XML streaming instruction properties of the nth anchor element, and `PTS(n)`, `TIME_SCALE(n)` and `PTS_DELTA(n)` and `ENCODE_AS_RAP(n)` represent the properties of the associated nth process unit.

NOTE 3 Annex N provides an informative example of how this fragmenter in a chain which transmits process units from one peer to another.

8.6.5 Examples

8.6.5.1 General Examples

EXAMPLE 1 This example shows an example of BSD for an ISO/IEC 14496-2 visual elementary stream.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES">
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>0</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>23 4636</Payload>
   </VOP>
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>1</vop_coding_type>
      <Stuffing>17</Stuffing>
      <Payload>4664 93</Payload>
   </VOP>
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>2</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>4762 11</Payload>
   </VOP>
  <!-- etc -->
</Bitstream>
```

EXAMPLE 2 This example shows an example of BSD for an ISO/IEC 14496-2 visual elementary stream with XML streaming instructions attributes.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES"
           si:puMode="ancestorsDescendants"
           si:timescale="15" s:ptsDelta="1">
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>0</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>23 4636</Payload>
   </VOP>
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>1</vop_coding_type>
      <Stuffing>17</Stuffing>
      <Payload>4664 93</Payload>
   </VOP>
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
```

```
       <vop_coding_type>2</vop_coding_type>
       <Stuffing>16</Stuffing>
       <Payload>4762 11</Payload>
    </VOP>
  <!-- etc -->
</Bitstream>
```

EXAMPLE 3 This example shows an example of AdaptationQoS with XML streaming instructions attributes.

```
<?xml version="1.0" ?>
<DIA xmlns:si="urn:mpeg:mpeg21:200x:01-SI"
     xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS"
     xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   <DescriptionMetadata>
      <ClassificationSchemeAlias alias="AQoS" href="urn:mpeg:mpeg21:2003:01-DIA-
AdaptationQoSCS-NS"/>
   </DescriptionMetadata>
   <Description si:timescale="1000" si:ptsDelta="1280"
si:puMode="precedingSibling" xsi:type="AdaptationQoSType">
      <IOPin id="BANDWIDTH">
         <GetValue xsi:type="SemanticalDataRefType"
semantics="urn:mpeg:mpeg21:2003:01-DIA-AdaptationQoSCS-NS:6.6.5.3"/>
      </IOPin>
      <IOPin id="TARGET-BITRATE"/>
      <IOPin id="GOPNR"/>
      <Module xsi:type="LookUpTableSwitchType" switchIOPinRefs="GOPNR">
         <Axis iOPinRef="BANDWIDTH">
            <AxisValues xsi:type="IntegerVectorType">
               <Vector>25 50 75 100 125 150 175 200 225 250 275 300 325 350 375
400 425 1000</Vector>
            </AxisValues>
         </Axis>
         <ContentSwitch iOPinRef="TARGET-BITRATE">
            <ContentDataSwitch switchValues="1" xmlsi:anchorElement="true">
               <ContentValues xsi:type="IntegerMatrixType" mpeg7:dim="18">
                  <Matrix>25 50 75 100 125 150 175 200 225 250 275 300 325 350
375 400 425 1000000</Matrix>
               </ContentValues>
            </ContentDataSwitch>
            <ContentDataSwitch switchValues="2" xmlsi:anchorElement="true">
               <ContentValues xsi:type="IntegerMatrixType" mpeg7:dim="18">
                  <Matrix>25 50 75 100 125 150 175 200 225 250 275 300 325 350
375 400 425 1000000</Matrix>
               </ContentValues>
            </ContentDataSwitch>
            <!-- and so on ... -->
         </ContentSwitch>
      </Module>
   </Description>
</DIA>
```

EXAMPLE 4 This example shows the third process unit of EXAMPLE 2 with `puMode = self`.

```
<?xml version="1.0" ?>
<VOP xmlns="urn:MPEG4_VES" si:anchorElement="true"/>
```

EXAMPLE 5 This example shows the third process unit of EXAMPLE 2 with `puMode = ancestors`.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES"
           si:puMode="ancestorsDescendants"
           si:timescale="15" s:ptsDelta="1">
   <VOP si:anchorElement="true"/>
</Bitstream>
```

EXAMPLE 6 This example shows the third process unit of EXAMPLE 2 with `puMode = descendants`.

```
<?xml version="1.0" ?>
<VOP xmlns="urn:MPEG4_VES" si:anchorElement="true">
   <vop_start_code>000001B6</vop_start_code>
   <vop_coding_type>2</vop_coding_type>
   <Stuffing>16</Stuffing>
   <Payload>4762 11</Payload>
</VOP>
```

EXAMPLE 7 This example shows the third process unit of EXAMPLE 2 with `puMode = ancestorsDescendants`.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES"
           si:puMode="ancestorsDescendants"
           si:timescale="15" s:ptsDelta="1">
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>2</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>4762 11</Payload>
   </VOP>
</Bitstream>
```

EXAMPLE 8 This example shows the third process unit of EXAMPLE 2 with `puMode = preceding`.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES"
           si:puMode="ancestorsDescendants"
           si:timescale="15" s:ptsDelta="1">
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>0</vop_coding_type>
```

```
        <Stuffing>16</Stuffing>
        <Payload>23 4636</Payload>
    </VOP>
    <VOP si:anchorElement="true">
        <vop_start_code>000001B6</vop_start_code>
        <vop_coding_type>1</vop_coding_type>
        <Stuffing>17</Stuffing>
        <Payload>4664 93</Payload>
    </VOP>
    <VOP si:anchorElement="true">
        <vop_start_code>000001B6</vop_start_code>
        <vop_coding_type>2</vop_coding_type>
        <Stuffing>16</Stuffing>
        <Payload>4762 11</Payload>
    </VOP>
</Bitstream>
```

EXAMPLE 9   This example shows the second process unit of EXAMPLE 3 with `puMode = precedingSibling`.

```
<?xml version="1.0" ?>
<DIA xmlns:si="urn:mpeg:mpeg21:200x:01-SI"
     xmlns="urn:mpeg:mpeg21:2003:01-DIA-NS"
     xmlns:mpeg7="urn:mpeg:mpeg7:schema:2001"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <DescriptionMetadata>
        <ClassificationSchemeAlias alias="AQoS" href="urn:mpeg:mpeg21:2003:01-DIA-
AdaptationQoSCS-NS"/>
    </DescriptionMetadata>
    <Description si:timescale="1000" si:ptsDelta="1280"
si:puMode="precedingSibling" xsi:type="AdaptationQoSType">
        <IOPin id="BANDWIDTH">
            <GetValue xsi:type="SemanticalDataRefType"
semantics="urn:mpeg:mpeg21:2003:01-DIA-AdaptationQoSCS-NS:6.6.5.3"/>
        </IOPin>
        <IOPin id="TARGET-BITRATE"/>
        <IOPin id="GOPNR"/>
        <Module xsi:type="LookUpTableSwitchType" switchIOPinRefs="GOPNR">
            <Axis iOPinRef="BANDWIDTH">
                <AxisValues xsi:type="IntegerVectorType">
                    <Vector>25 50 75 100 125 150 175 200 225 250 275 300 325 350 375
400 425 1000</Vector>
                </AxisValues>
            </Axis>
            <ContentSwitch iOPinRef="TARGET-BITRATE">
                <ContentDataSwitch switchValues="2" xmlsi:anchorElement="true">
                    <ContentValues xsi:type="IntegerMatrixType" mpeg7:dim="18">
                        <Matrix>25 50 75 100 125 150 175 200 225 250 275 300 325 350
375 400 425 1000000</Matrix>
                    </ContentValues>
                </ContentDataSwitch>
            </ContentSwitch>
        </Module>
    </Description>
</DIA>
```

EXAMPLE 10 This example shows the second process unit of EXAMPLE 2 with `puMode = sequential`.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES"
           si:puMode="ancestorsDescendants"
           si:timescale="15" s:ptsDelta="1">
   <VOP si:anchorElement="true">
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>1</vop_coding_type>
      <Stuffing>17</Stuffing>
      <Payload>4664 93</Payload>
   </VOP>
</Bitstream>
```

8.7 Properties stylesheet

8.7.1 Introduction

Subclause 8.7 specifies a mechanism to dynamically apply media resource or XML streaming instructions to an XML document. This mechanism relies on an external document called properties stylesheet, which allows setting properties to given elements of an XML document without having to modify it.

Media resource and XML streaming instructions define a set of properties. Properties are to be understood in the meaning of the XML Infoset and characterize an element information item. Both the media resource streaming instructions and XML streaming instructions also specify a set of associated XML attributes which assign the corresponding properties to the element information item. These attributes are meant to be used within the XML document to be streamed. It is therefore required to modify the document. When properties are set according to a regular patter, it is also possible to specify them in an external document, which then avoids modifying the document.

NOTE A properties stylesheet does not specify an XML transformation like XSLT style sheets do. This term must be understood as in cascading style sheets (CSS) which define properties but do not modify the HTML document they are applied to.

8.7.2 Syntax

```
<?xml version="1.0"?>
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="PSS-AMD2.xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ps="urn:mpeg:mpeg21:2003:01-DIA-PSS-NS"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-PSS-NS">

  <annotation>
    <documentation>
      Declaration of properties stylesheet
    </documentation>
  </annotation>

  <element name="properties">
    <complexType>
      <sequence>
```

```
        <element name="template" minOccurs="0" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="property" minOccurs="0" maxOccurs="unbounded">
                <complexType>
                  <attribute name="name" type="QName" use="required"/>
                  <attribute name="namespace" type="anyURI" use="optional"/>
                  <attribute name="value" type="string" use="required"/>
                </complexType>
              </element>
            </sequence>
            <attribute name="match" type="string" use="required"/>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>

</schema>
```

8.7.3 Semantics

A properties stylesheet is an ordered sequence of `ps:template` elements. Each `ps:template` element specifies a set of properties defined by the `ps:property` elements to be applied to the elements that match the Lightweight XML Path expression language (LXPath) expression provided by the `match` attribute and specified in subclause 8.7.4. Each property is defined by a qualified name and a value. When the name has a prefix, it should be resolved against the namespace binding context of the property element. When it has no prefix, its namespace is the one indicated by the namespace attribute. If this attribute contains an empty string then the property has no namespace. If this attribute is not present, then the default namespace defined for the property element applies. If no such default namespace is defined then the property has no namespace. A `ps:property` element shall not have simultaneously a `namespace` attribute and a prefix.

Conflicts for setting properties may occur in two cases:

— When the property is already set in the original document (either specified by an attribute for the same XML element, or inherited from a parent element).

— When two templates of the properties stylesheet set the same properties for the same XML element.

In this case, the conflict is solved according to the following priority rule:

— A property set by a properties stylesheet overrides the one set by an attribute in the original document.

— In case two templates of the same properties stylesheet set the same properties with different values, the value of the last `ps:template` element in the document overrides the values previously set.

A properties stylesheet itself does not define any processing on the XML document. However, applying a properties stylesheet to an XML document may augment its infoset, but does not modify the document itself.

8.7.4 Lightweight XML Path expression language

8.7.4.1 Introduction

The syntax of Lightweight XML Path expression language (LXPath) is similar to XPath but its semantics differs. An XPath expression is resolved against a context which contains the full XML document. Conversely, the context used in LXPath is the ancestor stack of the context node and its position within sibling elements. Whereas evaluating an XPath expression requires loading the full document in memory, evaluating an LXPath expression only requires a limited context that can be constructed while parsing the XML document with an application programmable interface (API) such as the simple API for XML (SAX).

EXAMPLE In XPath, the expression `/node1/node2` returns a sequence containing all `node2` elements, whose parent element is the document element and is named `node1`. In LXPath, on contrary, the same expression returns a sequence containing a single node from this node-set; the one which is an ancestor of the current node.

Used as a filtering pattern, an LXPath expression is evaluated as a Boolean, i.e., true if the expression returns a non-null node-set, false otherwise.

The advantage of LXPath over XPath is that it can be used in streaming scenarios, especially in live streaming scenarios where the size of the XML document is infinite, which is not possible with XPath. It particularly fits SAX-based architectures.

8.7.4.2 Grammar for LXPath

Table AMD2.4 specifies the grammar of LXPath according to ISO/IEC 14977:1996, i.e. Extended Backus-Naur Form (EBNF) notation with `MatchPattern` as entry point.

Table AMD2.4 — EBNF for LXPath.

```
MatchPattern          ::=   BoolExpr
BoolExpr              ::=   Expression ( "|" Expression)*
Expression            ::=   ( "/" | "//" )? PathStep (( "/" | "//" ) PathStep )*
PathStep              ::=   (QName | WildCard) Predicate*
WildCard              ::=   "*" | ( "*" ":" NCName) | ( NCName ":" "*")
Predicate             ::=   "[" PredicateExpr "]"
PredicateExpr         ::=   OrExpr
OrExpr                ::=   AndExpr (("or" | "|") AndExpr )*
AndExpr               ::=   ComparisonExpr ( "and" ComparisonExpr )*
ComparisonExpr        ::=   AdditiveExpr ( GeneralComp AdditiveExpr )?
GeneralComp           ::=   "=" | "!=" | "<" | "<=" | ">" | ">="
AdditiveExpr          ::=   MultiplicativeExpr ( ("+" | "-") MultiplicativeExpr )*
MultiplicativeExpr    ::=   PrimaryExpr ( ("*" | "div" | "idiv" | "mod") PrimaryExpr )*
PrimaryExpr           ::=   AttrExpr | Function | StringLiteral | NumericLiteral
AttrExpr              ::=   "@" NCName
Function              ::=   "position()"
StringLiteral         ::=   "'" Char* "'"
NumericLiteral        ::=   IntegerLiteral | DecimalLiteral
IntegerLiteral        ::=   ("-" | "+")? Digits
DecimalLiteral        ::=   ("-" | "+")? ("." Digits) | (Digits "." [0-9]*)
Digits                ::=   [0-9]+
NCName                ::=   [http://www.w3.org/TR/REC-xml-names/#NT-NCName]
```

```
QName                    ::=   [http://www.w3.org/TR/REC-xml-names/#NT-QName]

Char                     ::=   [http://www.w3.org/TR/REC-xml/#NT-Char]
```

8.7.5 Examples

EXAMPLE 1 Let us consider the following XML document containing arbitrary XML elements which is used for all subsequent examples. For the subsequent examples, this document is referred to as the *original document*. In particular, the result of the fragmenter will be shown.

```
<?xml version="1.0" ?>
<el0 xmlns="urn:myExampleNS">
  <el1>Content of el1</el1>
  <el2 >
    <el21>
      <el211>Content of el211</el211>
    </el21>
    <el22 toto22="toto22">
      <el221/>
      <el222/>
    </el22>
  </el2>
  <el3/>
</el0>
```

EXAMPLE 2 The following XML document shows an example properties stylesheet which is used to assign properties to the original document as shown in the above example.

```
<?xml version="1.0"?>
<properties
    xmlns="urn:mpeg:mpeg21:2003:01-DIA-PSS-NS"
    xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
    xmlns:t="urn:myExampleNS"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS PSS.xsd">

    <template
      match="/t:el0/t:el1 | /t:el0/t:el2/t:el21/t:el211 | /t:el0/t:el2/t:el22">
        <property name="si:anchorElement" value="true"/>
    </template>

    <template match="t:el22">
        <property name="si:puMode" value="self"/>
    </template>

    <template match="/t:el0">
        <property name="si:puMode" value="descendants"/>
        <property name="si:timeScale" value="22050"/>
        <property name="si:ptsDelta" value="1024"/>
    </template>
</properties>
```

EXAMPLE 3 The following XML document shows the original document with the properties set by attributes instead of using the properties stylesheet.

```
<?xml version="1.0" ?>
<el0 xmlns="urn:myExampleNS" xmlns:si="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
  si:puMode="descendants"
  si:timeScale="22050"
  si:ptsDelta="1024">
  <el1 si:anchorElement="true">Content of el1</el1>
  <el2 >
    <el21>
      <el211 si:anchorElement="true">Content of el211</el211>
    </el21>
    <el22 toto22="toto22" si:puMode="self" si:anchorElement="true">
      <el221/>
      <el222/>
    </el22>
  </el2>
  <el3/>
</el0>
```

EXAMPLE 4 The following XML document shows an example BSD describing an ISO/IEC 14496-3 BSAC bitstream.

```
<?xml version="1.0" encoding="UTF-8"?>
<BSACBitstream xmlns="urn:mpeg:mpeg4:BSAC"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:gbsd="urn:mpeg:mpeg21:2003:01-DIA-gBSD-NS"
  xsi:schemaLocation="urn:mpeg:mpeg4:BSAC BSAC.xsd"
  gbsd:addressUnit="bit" gbsd:addressMode="Absolute"
  bs1:bitstreamURI="prom.raw">

  <frame>
    <base>
      <length>618</length>
      <stuff1>11 5</stuff1>
      <topLayer>48</topLayer>
      <stuff2>22 738</stuff2>
    </base>
    <layers>
      <layer>760 48</layer>
      <layer>808 48</layer>
      <!-- and so on... -->
    </layers>
  </frame>

  <frame>
    <base>
      <length>391</length>
      <stuff1>4955 5</stuff1>
      <topLayer>48</topLayer>
      <stuff2>4966 738</stuff2>
    </base>
    <layers>
      <layer>5704 48</layer>
      <layer>5752 48</layer>
      <!-- and so on... -->
    </layers>
  </frame>
  <!-- and so on... -->
</BSACBitstream>
```

EXAMPLE 5 The following XML document shows a properties stylesheet specifying which BSD elements are enriched with which properties from the media resource streaming instructions and its actual values.

```
<?xml version="1.0" encoding="UTF-8"?>
<properties
  xmlns="urn:mpeg:mpeg21:2003:01-DIA-PSS-NS"
  xmlns:msi="urn:mpeg:mpeg21:2003:01-DIA-MSI-NS"
  xmlns:b="urn:mpeg:mpeg4:BSAC"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS PSS.xsd">

  <template match="/b:BSACBitstream/b:frame">
    <property name="msi:au" value="true"/>
  </template>

  <template match="/b:BSACBitstream">
    <property name="msi:rap" value="true"/>
    <property name="msi:timeScale" value="22050"/>
    <property name="msi:dtsDelta" value="1024"/>
    <property name="msi:ctsOffset" value="0"/>
  </template>
</properties>
```

EXAMPLE 6 The following XML document shows an example of BSD for an ISO/IEC 14496-2 visual elementary stream.

```
<?xml version="1.0" ?>
<Bitstream xmlns="urn:MPEG4_VES" bs1:bitstreamURI="akiyo.mpg4"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xsi:schemaLocation="MPEG4_VES MPEG4_VES.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:mp4="MPEG4_VES">
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>0</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>23 4636</Payload>
   </VOP>
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>1</vop_coding_type>
      <Stuffing>17</Stuffing>
      <Payload>4664 93</Payload>
   </VOP>
   <VOP>
      <vop_start_code>000001B6</vop_start_code>
      <vop_coding_type>2</vop_coding_type>
      <Stuffing>16</Stuffing>
      <Payload>4762 11</Payload>
   </VOP>
  <!-- etc -->
</Bitstream>
```

EXAMPLE 7 The following XML document shows an example of a properties stylesheet specifying which BSD elements are enriched with which properties from the XML streaming instructions and its actual values.

```
<?xml version="1.0" encoding="UTF-8"?>
<properties
  xmlns="urn:mpeg:mpeg21:2003:01-DIA-PSS-NS"
  xmlns:m="urn:MPEG4_VES"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <template match="/m:Bitstream/m:VOP">
    <property name="anchorElement" namespace="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
              value="true"/>
  </template>

  <template match="/m:Bitstream">
    <property name="puMode" namespace="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
              value="ancestorsDescendants"/>
    <property name="timeScale" namespace="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
              value="15"/>
    <property name="ptsDelta" namespace="urn:mpeg:mpeg21:2003:01-DIA-XSI-NS"
              value="1"/>
  </template>
</properties>
```

In subclause 9.3.2, replace the AdaptationQoS syntax with the following syntax:

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

<complexType name="AdaptationQoSType">
   <complexContent>
      <extension base="dia:DIADescriptionType">
         <choice>
            <element name="Module"  type="dia:BaseModuleType"
               maxOccurs="unbounded"/>
            <element name="IOPin" type="dia:IOPinType" maxOccurs="unbounded"/>
            <element name="Constraints" type="dia:ReferenceType"
               minOccurs="0"/>
         </choice>
      </extension>
   </complexContent>
</complexType>
```

In subclause 9.5.2, replace the text of this subclause with the following text:

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

<complexType name="IOPinType">
   <sequence>
      <choice>
         <element name="GetValue" type="dia:GetValueType" minOccurs="0"/>
         <element name="GetVector" type="dia:VectorDataType" minOccurs="0"/>
      </choice>
      <element name="Axis" type="dia:AxisBaseType" minOccurs="0"/>
      <element name="Default" type="dia:ValueDataType" minOccurs="0"/>
   </sequence>
   <attribute name="id" type="ID" use="required"/>
   <attribute name="semantics" type="mpeg7:termReferenceType" use="optional"/>
   <attribute name="discrete" type="boolean" use="optional" default="true"/>
</complexType>
```

In subclause 9.5.3, append the following text at the end of the subclause:

`GetVector`	Describes the vector of values to be assigned to the IOPin.

In subclause 9.9.4, replace the value of the `minOccurs` *attribute of the* `Switch` *element with following value:* 1

In subclause 9.9.6, replace the value of the `minOccurs` *attribute of the* `ContentDataSwitch` *element with following value:* 1

In subclause 9.9.8, replace the value of the `minOccurs` *attribute of the* `Switch` *element with following value:* 1

In subclause 13.2, replace the definition of the complex type `TargetType` *with following text:*

```
  <complexType name="TargetType">
    <complexContent>
      <extension base="dia:DIABaseType">
        <attribute name="targetDescription" type="string" use="optional"/>
        <attribute name="semantics" type="mpeg7:termReferenceType"
use="optional"/>
      </extension>
    </complexContent>
  </complexType>
```

In subclause 13.3, append a new row after the definition of the `targetDescription` *attribute:*

`semantics`	Describes the semantics of the variable referenced. CSs that may be used for this purpose are the `AdaptationQoSCS` or `MediaInformationCS` defined in Annex A.2.2 and A.2.10 respectively. The *targetDescription* and *semantics* attributes are exclusive.

In Annex A.2.2, replace:

```
<Term termID="1.3.1.1">
  <Name xml:lang="en">B Frame Dropping</Name>
    <Definition xml:lang="en">Describes the number of B-frames to be dropped
between two successive anchor frames in each sub-GOP. It is assumed that B-frames
are dropped from the end of a sub-GOP.</Definition>
</Term>
<Term termID="1.3.1.2">
  <Name xml:lang="en">P Frame Dropping</Name>
    <Definition xml:lang="en">Describes the number of P-frames to be dropped in a
GOP. It is assumed that P-frames are dropped from the end of a GOP.</Definition>
</Term>
```

with:

```
<Term termID="1.3.1.1">
  <Name xml:lang="en">B Frame Dropping</Name>
    <Definition xml:lang="en">Describes the number of B-frames to be dropped
between two successive anchor frames in each sub-GOP.</Definition>
</Term>
<Term termID="1.3.1.2">
  <Name xml:lang="en">P Frame Dropping</Name>
    <Definition xml:lang="en">Describes the number of P-frames to be dropped in a
GOP.</Definition>
</Term>
```

In Annex A.2.2, insert the following classification scheme terms after the term 1.3.8.1:

```
<Term termID="1.3.9">
  <Name xml:lang="en">SVC Adaptation</Name>
  <Definition xml:lang="en">Describes adaptation operators that adapt the
incoming SVC bitstream by selecting the specified values of spatial layers,
temporal levels and quality reduction.</Definition>
  <Term termID="1.3.9.1">
    <Name xml:lang="en">Spatial Layers</Name>
    <Definition xml:lang="en">Indicates the number of enhancement layers for
spatial resolution to be truncated from the input bitstream. It is assumed that
the highest enhancement layer is truncated first.</Definition>
  </Term>
  <Term termID="1.3.9.2">
    <Name xml:lang="en">Temporal Levels</Name>
    <Definition xml:lang="en">Indicates the number of enhancement layers for
temporal resolution to be truncated from the input bitstream. It is assumed that
the highest enhancement layer is truncated first.</Definition>
  </Term>
  <Term termID="1.3.9.3">
    <Name xml:lang="en">Quality Reduction </Name>
    <Definition xml:lang="en">Indicates the SNR enhancement fraction that should
be truncated from the input bitstream. It is assumed that the Quality Reduction
for no truncation is "0.00"and Quality Reduction for full truncation is
"1.00".</Definition>
  </Term>
  <Term termID="1.3.9.4">
    <Name xml:lang="en">Quality levels </Name>
    <Definition xml:lang="en">Indicates the number of quality layers indictaed by
the quality_layer parameter to be truncated from the input bitstream. It is
assumed the highest value is truncated first</Definition>
  </Term>
  <Term termID="1.3.9.5">
    <Name xml:lang="en">Priority id</Name>
    <Definition xml:lang="en">Indicates the maximum allowed value for
priority_id. NAL units with a priority_id greater than this value shall be
dropped.</Definition>
  </Term>
</Term>
```

In Annex A.2.2, insert the following classification scheme terms after the term 6.4 as follows:

```
   <Term termID="6.4.4">
    <Name xml:lang="en">UserInfo</Name>
    <Definition xml:lang="en">Describes the UserInfo as defined in this part of
ISO/IEC 21000.</Definition>
   </Term>
   <Term termID="6.4.5">
    <Name xml:lang="en">UsagePreferences</Name>
    <Definition xml:lang="en">Describes the UsagePreferences as defined in this
part of ISO/IEC 21000.</Definition>
   </Term>
   <Term termID="6.4.6">
    <Name xml:lang="en">UsageHistory</Name>
    <Definition xml:lang="en">Describes the UsageHistory as defined in this part
of ISO/IEC 21000.</Definition>
   </Term>
```

Insert a new Annex A.2.16 as follows:

```
<ClassificationScheme
   uri="urn:mpeg:mpeg21:2003:01-DIA-RightsLanguagesCS-NS" >
  <Header xsi:type="DescriptionMetadata">
    <FreeTextAnnotation xml:lang="en">
      List of rights expression languages (RELs).
    </FreeTextAnnotation>
  </Header>
  <Term termId="REL">
    <Name xml:lang="en">MPEG-21 REL</Name>
    <Definition xml:lang="en">
      MPEG-21 Rights Expression Language as defined in ISO/IEC 21000-5.
    </Definition>
    <Term termId="RELProfile:Base">
      <Name xml:lang="en">MPEG-21 REL Base Profile</Name>
      <Definition xml:lang="en">
        MPEG-21 Rights Expression Language Base Profile as defined in
ISO/IEC 21000-5.
      </Definition>
    </Term>
    <Term termId="RELProfile:DAC">
      <Name xml:lang="en">MPEG-21 REL DAC Profile</Name>
      <Definition xml:lang="en">
        MPEG-21 Rights Expression Language Dissemination and Capture
        Profile as defined in ISO/IEC 21000-5.
      </Definition>
    </Term>
  </Term>
</ClassificationScheme>
```

Insert new Annexes K, L, M, N, O and P as follows:

Annex K
(normative)

Defining extension datatypes library for BSDL

This annex further specifies how to specify and use extension datatypes in a BS Schema.

For specifying a library of extension datatypes, a BS Schema author needs to:

— Specify a namespace identifying the datatypes library.

— Specify a local name for each datatype.of the library.

Each datatype of this library can then be uniquely addressed via a URI constructed as follows:

— The base URI is the URI of the library namespace

— The fragment identifier is the local name of the datatype

For assigning the extension datatype to an BSD element, the BS Schema author can use the `bs1:codec` attribute either in the BSD or in the BS Schema as illustrated in the example below.

Additionally, the BS Schema author needs to specify the decoding and encoding schemes of the datatype, i.e., how to read the syntactical element from the bitstream and instantiate its lexical representation and binary encode the lexical value and writing it to the bitstream.

NOTE 1 Note that since a BSD is required to be valid (from a XML Schema point of view) with respect to its BS Schema, the lexical value generated by the BintoBSD processor needs to conform to the relevant datatype indicated in the BS Schema.

NOTE 2 The behavior of a BintoBSD or BSDtoBin processor not supporting an extension datatype is not specified.

EXAMPLE The following BS Schema snippet illustrates how to use the `bs1:codec` attribute in a BS Schema. A BS Schema author specifies a library of extension datatypes with a namespace `urn:example:myLibrary`. This library contains a datatype with a local name equal to `myCodecSpecificType`. The complex type `myType` is defined as a derivation by extension of the `xs:int` datatype by declaring a `bs1:codec` attribute with a default value equal to the URI of `myCodecSpecificType`. Both elements `myElement1` and `myElement2` are declared with a type equal to `myType`. When parsing the BSD snippet shown below, the BSDtoBin parser builds the Post Schema Validated Infoset of the BSD. In the infoset, both elements have a `bs1:codec` attribute indicating the extension datatype which then overrides the type for the BSDtoBin processor. Reciprocally, a BintoBSD processor instantiating an element of type `myType` considers the value of the `bs1:codec` attribute in the BS Schema to override the `myType` datatype. Note that since it is a BSD is required to be valid (from a XML Schema point of view) with respect to its BS Schema, the lexical value generated by the BintoBSD processor needs to conform to the indicated datatype (here, `xsd:int`).

```
<!-- Example of BS Schema snippet using the bs1:codec attribute -->

<xs:element name="myElement1" type="myType"/>
<xs:element name="myElement2" type="myType"/>

<xs:complexType name="myType">
  <xs:simpleContent>
     <xs:extension base="xs:int">
       <x:attribute ref="bs1:codec"
          default="urn:example:myLibrary#myCodecSpecificType"/>
     </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

```
<!-- Example of BSD snippet using the bs1:codec attribute -->
<myElement1 bs1:codec="myType">1</myElement1>
<myElement2>2</myElement2>
```

Annex L
(normative)

Optional features for BSDL

L.1 Introduction

This annex specifies optional BSDL features. A conformant BSDtoBin or BintoBSD parser does not have to implement the features specified by this annex. However, if a conformant parser does implement functionality corresponding to the features in this annex, then the implementation shall conform to this specification. Each of the three features L.2, L.3 and L.4 may be implemented independently.

The optional features are specified in the following namespaces:

— Schema for BSDL-1 optional extensions: urn:mpeg:mpeg21:2003:01-DIA-BSDL1x-NS

— Schema for BSDL-2 optional extensions: urn:mpeg:mpeg21:2003:01-DIA-BSDL2x-NS

L.2 ECMAScript-based extension datatypes

L.2.1 Introduction

This feature provides a mechanism to express the parsing process for extension datatypes within a BS Schema. This allows a BSDL parser to process data structures that cannot be specified using other BSDL syntax elements.

EXAMPLE The ISO/IEC 16262 (ECMAScript) extension datatype feature may be used to allow a BSDL Parser to process Variable Length Codes, such as Exponential Golumb-coded integers, Huffman codes, or Arithmetic-coded values.

An extension datatype shall be derived in a BS Schema by restriction of the `bs1x:extensionType` built-in type. Conformant BintoBSD and BSDtoBin parsers shall process the extension datatype according to the `BintoBSD()` and `BSDtoBin()` functions declared by a `bs1x:script` facet which shall be present as an `appinfo annotation` to the restriction of `bs1x:extensionType`.

`BintoBSD()` and `BSDtoBin()` may call the functions specified in subclause L.2.5 to access data from the BintoBSD and BSDtoBin parsers. `BintoBSD()` and `BSDtoBin()` may also instantiate Objects specified by the ISO/IEC 16262 (ECMAScript) bindings for the Document Object Model (DOM) Level 3 Core Specification.

L.2.2 BS Schema Syntax

```
<schema
  version="ISO/IEC 21000-7:2004/FDAM2"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1x-NS"
  xmlns="http://www.w3.org/2001/XMLSchema">

  <simpleType name="extensionType">
    <restriction base="string">
      <annotation><documentation>
        Extension data types shall restrict this type with a bs1x:script node.
      </documentation></annotation>
    </restriction>
  </simpleType>
```

```
<element name="script">
    <annotation><documentation>
      This element shall be used to specify the BintoBSD and BSDtoBin parsing
      processes for an extension datatype, via BintoBSD() and BSDtoBin(value)
      functions (respectively). Functions are declared in either
      element content or an external file referenced by the ref
      attribute, but not both.
    </documentation></annotation>
    <complexType>
      <simpleContent >
        <extension base="string">
          <attribute name="ref" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>
</schema>

<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7:2004/Amd.2 -->
<!-- Schema for BSDL-1 extensions : optional features -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="BSDL-1x-AMD2.xsd"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1x-NS"
  xmlns="http://www.w3.org/2001/XMLSchema">

  <annotation>
    <documentation>
      Schema for BSDL-1 optional extensions: ISO/IEC 16262-based extension
datatypes
    </documentation>
  </annotation>

  <simpleType name="extensionType">
    <restriction base="string">
      <annotation><documentation>
        Extension data types shall restrict this type with a bs1x:script node.
      </documentation></annotation>
    </restriction>
  </simpleType>

  <element name="script">
    <annotation><documentation>
      This element shall be used to specify the BintoBSD and BSDtoBin parsing
      processes for an extension datatype, via BintoBSD() and BSDtoBin(value)
      functions (respectively). Functions are declared in either
      element content or an external file referenced by the ref
      attribute, but not both.
    </documentation></annotation>
    <complexType>
      <simpleContent >
        <extension base="string">
          <attribute name="ref" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

</schema>
```

L.2.3 Semantics of datatype extensions

The following datatype shall be used in the BS Schema to implement the extension datatype feature.

Name	*Definition*
`bs1x:extensionType`	Derived types in a BS Schema which implement extension datatypes according to this clause shall restrict this Type. Restrictions of this type shall declare a `bs1x:script` facet.

L.2.4 Semantics of structural extensions

The following facet shall be used in the BS Schema to implement the extension datatype feature.

Name	*Definition*
`bs1x:script`	Facet which specifies the BintoBSD and BSDtoBin parsing processes for an extension datatype, via ISO/IEC 16262 (ECMAScript). ISO/IEC 16262 (ECMAScript) functions may be declared in either element content or an external file referenced by the ref attribute, but not both. Only one of element content or ref attribute shall be present on any `bs1x:script` facet. A `bs1x:script` facet shall declare both of the following functions: – `BintoBSD()`: This function shall be used to specify the process followed by the BintoBSD parser to instantiate a BSD element conforming to the extension datatype. – `BSDtoBin(value)`: This function shall be used to specify the process followed by the BSDtoBin parser to read a BSD element and write data to the output bitstream. The value parameter shall be a String object containing the instantiated value of the extension type.

L.2.5 Syntax and Semantics of BSDL-defined ISO/IEC 16262 (ECMAScript) functions

The following functions shall be provided by a conformant BSDL parser that implements the extension datatype feature.

Name	*Definition*
`read(bits)`	This function shall be provided by a BintoBSD parser and may be called by the `BintoBSD()` function of a `bs1x:script` facet. When this function is called, a BintoBSD shall read from the bitstream the number of bits specified by the integer value of the `bits` parameter, and return the unsigned integer value of the bits read.

Name	*Definition*
`write(value,bits)`	This function shall be provided by a BSDtoBin parser and may be called by the `BSDtoBin(value)` function of a `bs1x:script` facet. When this function is called, a BSDtoBin parser shall write to the bitstream the unsigned integer value of the `value` parameter, using the number of bits specified by the integer value of the `bits` parameter.
`xpath(exp,type)`	This function shall be provided by a BintoBSD parser and may be called by the `BintoBSD()` function of a `bs1x:script` facet. When this function is called, a BintoBSD shall execute the XPath expression declared by the string value of the `exp` parameter, and return the value of the result of the expression. The expression shall be evaluated in the context of the partially instantiated BSD. The type of object returned by the function shall be specified by the value of the type parameter, and shall be one of the following symbolic constants: Symbolic Constant — Return Type NODESET — A NodeList object. NODE — A Node object BOOLEAN — A Boolean primitive STRING — A String primitive NUMBER — A Number primitive NOTE The NodeList and Node types are defined by the Document Object Model, Level 3: Core, and the Boolean, String and Number primitive types are defined by ISO/IEC 16262 (ECMAScript).

L.2.6 Example

EXAMPLE The example below shows how to specify an extension datatype in a BS Schema for the Exponential Golumb type defined by ISO/IEC 14496-10 Advanced Video Coding.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs1x="urn:mpeg:mpeg21:2003:01-DIA-BSDL1x-NS"
  elementFormDefault="qualified">
   <simpleType name="expGolomb">
    <restriction base="bs1x:extensionType">
      <annotation><appinfo>
        <bs1x:script>
          <![CDATA[
          function BintoBSD() {
            var nBits = 0;
            var ret = 0;

            while ((ret = read(1)) == 0) nBits++; //read 0's
            if (ret == -1) throw "extensionType Error";

            ret = read(nBits);                  //read the rest
            if (ret == -1) throw "extensionType Error";

            return parseInt((1 << nBits) - 1 + ret);
          }
          function BSDtoBin(value) {
            var nBits = 0;
            var tmp = value + 1;
            while ((tmp >>= 1) > 0) nBits++; //count how many zeros to write

            tmp = 1;
            var i = 0;
            for (i = 0; i < nBits; i++) {
                tmp <<= 1;
            }

            write(0, nBits);               //write leading zeros
            write(1, 1);                   //write a one
            write(value + 1 - tmp, nBits);//write rest of code
          }
          ]]>
        </bs1x:script>
      </appinfo></annotation>
    </restriction>
  </simpleType>

  <!-- ... -->
</schema>
```

L.3 ECMAScript-based XPath extension functions

L.3.1 Introduction

This feature allows functions to be defined by a BS Schema which may be subsequently called from XPath expressions within the Schema.

EXAMPLE An example of where XPath extension functions may be used is in the `bs2:bitLength` facet. The number of bits required for a field may be expressed as the maximum value to be contained by the field. In this case, the value of the `bs2:bitLength` facet should be the logarithm to the base 2 of the maximum value of the field. Because XPath does not provide a logarithm function, such a function must be specified by this extension mechanism.

L.3.2 BS Schema Syntax

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7:2004/Amd.2 -->
<!-- Schema for Schema for BSDL-2 extensions : optional features -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="BSDL-2x1-AMD2.xsd"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL2x-NS"
  xmlns="http://www.w3.org/2001/XMLSchema">

  <annotation>
    <documentation>
      Schema for Schema for BSDL-2 optional extensions:
        ISO/IEC 16262-based XPath extension functions
    </documentation>
  </annotation>
  <element name="xpathScript">
    <annotation><documentation>
      This element shall be used to declare extension functions which may be
      called within XPath expressions. Functions are declared in either
      element content or an external file referenced by the ref
      attribute, but not both.
    </documentation></annotation>
    <complexType>
      <simpleContent >
        <extension base="string">
          <attribute name="ref" use="optional"/>
        </extension>
      </simpleContent>
    </complexType>
  </element>

</schema>
```

L.3.3 Semantics of structural extensions

The following facet shall be used in the BS Schema to implement the XPath extension function feature.

Name	*Definition*
`bs2x:xpathScript`	Facet which declares extension functions, via ISO/IEC 16262 (ECMAScript) language. Functions declared by the `bs2x:xpathScript` facet may be called from XPath expressions within the BS Schema. The scope of a function declared by a `bs2x:xpathScript` facet shall include any XPath expressions declared by the BS Schema element on which the facet is instantiated, as well as any XPath expressions declared by descendents of this element. ISO/IEC 16262 (ECMAScript) functions may be declared in either element content or an external file referenced by the ref attribute, but not both. Only one of element content or ref attribute shall be present on any `bs2x:xpathScript` facet.

L.3.4 Example

EXAMPLE The example below shows how to specify an XPath extension function in a BS Schema using the `bs2x:xpathScript` facet.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:bs2x="urn:mpeg:mpeg21:2003:01-DIA-BSDL2x-NS"
  elementFormDefault="qualified">
  <annotation><appinfo>
      <bs2x:xpathScript>
        <![CDATA[
          function log2(operand) {
          return Math.log(operand)/Math.log(2);
          }
        ]]>
      </bs2x:xpathScript>
  </appinfo></annotation>

  <!-- ... -->
</schema>
```

L.4 XPath variable declaration

L.4.1 Introduction

This feature allows a BS Schema to declare variables, which may subsequently be referenced from within XPath expressions.

EXAMPLE Variables may be used to simplify XPath expressions by replacing Node-tests with variable references. This can enhance the readability and execution speed of the expression.

NOTE 1 The syntax defined by the XML Path Language Version 1.0 is used to reference variables within an XPath Expression.

All variables declared using this feature shall have global scope. That is, they may be referenced from any XPath expression within a BS Schema.

NOTE 2 The global scoping of variables is necessary due to the potential mismatch between hierarchical structure of the BSD and the hierarchical structure of the BS Schema. That is, it is often necessary to refer to bitstream/BSD structures from a separate part of the BS Schema.

L.4.2 BS Schema Syntax

```
<?xml version="1.0"?>
<!-- Digital Item Adaptation ISO/IEC 21000-7:2004/Amd.2 -->
<!-- Schema for Schema for BSDL-2 extensions : optional features -->
<schema
  version="ISO/IEC 21000-7:2004/Amd.2"
  id="BSDL-2x2-AMD2.xsd"
  targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL2x-NS"
  xmlns="http://www.w3.org/2001/XMLSchema">

  <annotation>
    <documentation>
      Schema for Schema for BSDL-2 optional extensions:
        XPath variable declaration
    </documentation>
  </annotation>

  <element name="variable">
    <annotation><documentation>
      This facet may be used within a BS Schema to declare a variable
      which may be subsequently referenced from within XPath expressions.
    </documentation></annotation>
    <complexType>
      <attribute name="name" type="QName" use="required"/>
      <attribute name="value" type="normalizedString" default="."/>
      <attribute name="position" type="normalizedString" default="0"/>
    </complexType>
  </element>

</schema>
```