INTERNATIONAL STANDARD

ISO/IEC 23001-5

First edition
2008-02-15

Information technology — MPEG systems technologies —

Part 5: Bitstream Syntax Description Language (BSDL)

Technologies de l'information — Technologies des systèmes MPEG —

Partie 5: Langage de description de la syntaxe bitstream (BSDL)

Reference number
ISO/IEC 23001-5:2008(E)

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

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

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

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

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

ISO/IEC 23001-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

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

— *Part 1: Binary MPEG format for XML*

— *Part 2: Fragment request units*

— *Part 3: XML IPMP messages*

— *Part 4: Codec configuration representation*

— *Part 5: Bitstream Syntax Description Language (BSDL)*

Introduction

This international standard specifies BSDL (Bitstream Syntax Description Language), a language based on W3C XML Schema to describe the structure of a bitstream with an XML document named BS Description.

BSDL provides a normative grammar to describe in XML the high-level syntax of a bitstream; the resulting XML document is called a Bitstream Syntax Description (BS Description, BSD). This description is not meant to replace the original binary format, but acts as an additional layer, similar to metadata. In most cases, it will not describe the bitstream on a bit-per-bit basis, but rather address its high-level structure, e.g., how the bitstream is organized in layers or packets of data. Furthermore, the BS Description is itself scalable, which means it may describe the bitstream at different syntactic layers, e.g., finer or coarser levels of detail, depending on the application.

This language was initially developed in the context of Digital Item Adaptation (ISO/IEC 21000-7) as a generic tool for adapting scalable multimedia content. However, its use is not restricted to adaptation and may be relevant for any application parsing a bitstream. This is why this International Standard extracts the BSDL specification from ISO/IEC 21000-7 to make it available to other contexts.

Information technology — MPEG systems technologies —

Part 5:
Bitstream Syntax Description Language (BSDL)

1 Scope

This part of ISO/IEC 23001 specifies BSDL (Bitstream Syntax Description Language), a language based on W3C XML Schema to describe the structure of a bitstream with an XML document named BS Description.

BSDL provides a normative grammar to describe in XML the high-level syntax of a bitstream; the resulting XML document is called a Bitstream Syntax Description (BS Description, BSD). This description is not meant to replace the original binary format, but acts as an additional layer, similar to metadata. In most cases, it will not describe the bitstream on a bit-per-bit basis, but rather address its high-level structure, e.g., how the bitstream is organized in layers or packets of data. Furthermore, the BS Description is itself scalable, which means it may describe the bitstream at different syntactic layers, e.g., finer or coarser levels of detail, depending on the application.

This language was initially developed in the context of Digital Item Adaptation (ISO/IEC 21000-7) as a generic tool for adapting scalable multimedia content. However, its use is not restricted to adaptation and may be relevant for any application parsing a bitstream. This is why this part of ISO/IEC 23001 extracts the BSDL specification from ISO/IEC 21000-7 to make it available to other contexts.

2 Normative References

Namespaces in XML, World Wide Web Consortium 14 January 1999

Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000

XML Schema Part 1: Structures, W3C Recommendation 2 May 2001

XML Schema Part 2: Datatypes, W3C Recommendation 2 May 2001

XML Path Language (XPath), Version 1.0, W3C Recommendation 16 November 1999

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1
bitstream
resource consisting of a structured sequence of binary symbols

3.1.2
Bitstream Syntax Description
XML document describing the high-level structure of a **bitstream** using **Bitstream Syntax Description Language**

3.1.3
Bitstream Syntax Description Language
language specified in this part of ISO/IEC 23001 for defining a **Bitstream Syntax**

3.1.4
Bitstream Syntax Schema
schema written in the **Bitstream Syntax Description Language** describing the syntax of a given coding representation format

3.1.5
upstream
part of the bitstream that has not been parsed yet

3.1.6
downstream
part of the bitstream that has already been parsed

3.2 Abbreviated terms

BSDL:	Bitstream Syntax Description Language
BSD:	Bitstream Syntax Description
BS Schema:	Bitstream Syntax Schema
PSVI:	Post Schema Validated Infoset
XML:	Extensible Markup Language

4 Schema documents

4.1 General

This part of ISO/IEC 23001 specifies several schema documents. Each schema document has a version attribute, the value of which is "ISO/IEC 23001-5". Furthermore, an informative identifier is given as the value of the id attribute of the schema component. This identifier is non-normative and used as a convention in this specification to reference another schema document.

4.2 Use of prefixes in this specification

For clarity, throughout this part of ISO/IEC 23001, consistent namespace prefixes are used.

"`xsd:`" prefix is not normative. It is a naming convention in this document to refer to an element of the XML Schema namespace (`http://www.w3.org/2001/XMLSchema`). Similarly, "`xsi:`" refers to the `http://www.w3.org/2001/XMLSchema-instance` namespace.

"`xml:`" and "`xmlns:`" are normative prefixes defined in [9]. The prefix "`xml:`" is by definition bound to "`http://www.w3.org/XML/1998/namespace`". The prefix "`xmlns:`" is used only for namespace bindings and is not itself bound to any namespace name.

All other prefixes used in either the text or examples of this specification are not normative, e.g. "`bs1:`" and "`bs2:`".

In particular, most of the informative examples in this standard are provided as XML fragments without the normally required XML document declaration and thus miss a correct namespace binding context declaration. In these descriptions fragments, the different prefixes are bound to the namespaces as given in the following table.

Table 1 — Mapping of prefixes to namespaces in examples and text

Prefix	Corresponding namespace
xsi	http://www.w3.org/2001/XMLSchema-instance
xsd	http://www.w3.org/2001/XMLSchema
bs1	urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS
bs2	urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS
bs1i	urn:mpeg:mpeg21:2003:01-DIA-BSDL1-inf-NS
bs2i	urn:mpeg:mpeg21:2003:01-DIA-BSDL2-inf-NS

4.3 XML, Schema, XML Schema overview

XML Schema [3][4] is a language specified by W3C to specify constraints on XML Documents. The Document written in XML Schema and specifying constraints is called a Schema. The example Schema represented below expresses the following constraints:

— the el0 element shall contain an element named el1 and an element named el2

— the content of el1 shall be an integer

— the content of el2 shall be an hexadecimal value

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
  <xs:element name="el0">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="el1" type="xs:int"/>
        <xs:element name="el2" type="xs:hexBinary"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The document below represents an example of XML Document meeting these constraints.

```
<?xml version="1.0" encoding="UTF-8"?>
<el0 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:noNamespaceSchemaLocation="example1.xsd">
  <el1>1</el1>
  <el2>FF</el2>
</el0>
```

The XML Schema language allows validating the XML Document against its Schema, i.e. checking whether it meets the constraints expressed in the Schema. It is possible to express two types of constraints:

— on the structure of the XML Document (e.g. the el0 element shall contain an element named el1 and an element named el2)

— on the datatypes used (the content of el1 shall be an integer)

The XML Schema specification is structured in two parts dedicated to structural and datatypes aspects:

— Part 1: Structures

— Part 2: Datatypes

NOTE Part 0 of XML Schema [2] provides a useful introduction to XML Schema.

Note that other languages or grammars exist, that specify constraints on XML documents, such as Document Type Definitions (DTD) or RelaxNG. The key advantage of XML Schema for BSDL is its extensive library of datatypes.

4.4 BS Description, BS Schema, BSDL

4.4.1 BS Description

A binary media resource consists of a structured sequence of binary symbols, this structure being specific to the coding format. A bitstream is defined as the sequence of binary symbols representing this resource. XML is used to describe the high-level structure of a bitstream; the resulting XML document is called a Bitstream Syntax Description (BS Description or BSD). This description is not meant to replace the original binary format, but acts as an additional layer, similar to metadata. In most cases, it will not describe the bitstream on a bit-per-bit basis, but rather address its high-level structure, e.g., how the bitstream is organized in layers or packets of data. Furthermore, the BS Description is itself scalable, which means it may describe the bitstream at different syntactic layers, e.g., finer or coarser levels of detail, depending on the application.

With such a description, it is then possible for an application to access, use and process the contents of a bitstream via its BS Description.

EXAMPLE The document below is an example of a BS Description of an AVC bitstream. This example shows some of the datatypes available to represent the different bitstream symbols: whereas the value of the `startCode` element is provided in hexadecimal format, `nal_unit_type` is in decimal format. The datatype used for the `payload` element has a specific semantics: it points to a data segment in the original bitstream identified by its offset and length in bytes. In this case, and unlike the `startCode` and `nal_unit_type` elements, the bitstream data are not directly embedded in the BSD, but referenced by the BSD. This example also shows the scalability of a BSD: whereas some binary symbols in the header are described individually (`startCode`, `forbidden_zero_bit`...), the BSD describes the data segment following the header (`payload` element) as a whole without detailing its contents. It should be also noted that the element names used here are not normatively defined by BSDL, but are application dependent.

```
<?xml version="1.0" encoding="UTF-8"?>
<Bitstream
  xmlns="urn:mpeg:mpegb:example:AVC"
  bs1:bitstreamURI="football_40_frames.avc"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xsi:schemaLocation="urn:mpeg:mpegb:example:AVC SimpleAVC.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  bs1:bsdlVersion="ISO/IEC 23001-5">
  <seqParameterSet>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>3</nal_ref_idc>
    <nal_unit_type>7</nal_unit_type>
    <payload>5 9</payload>
  </seqParameterSet>
  <pictParameterSet>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>3</nal_ref_idc>
    <nal_unit_type>8</nal_unit_type>
    <payload>19 4</payload>
```

```
  </pictParameterSet>
  <sliceIDR>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>3</nal_ref_idc>
    <nal_unit_type>5</nal_unit_type>
    <payload>28 15653</payload>
  </sliceIDR>
  <sliceNonIDR>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>2</nal_ref_idc>
    <nal_unit_type>1</nal_unit_type>
    <payload>15686 10270</payload>
  </sliceNonIDR>
  <sliceNonIDR>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>2</nal_ref_idc>
    <nal_unit_type>1</nal_unit_type>
    <payload>25961 5916</payload>
  </sliceNonIDR>

  <!-- and so on... -->

</Bitstream>
```

4.4.2 BS Schema

As introduced above, a BS Description describes the contents of a bitstream. When this bitstream follows a given format, it is expected that other bitstreams following the same format may be described by similar BS Descriptions. It is therefore possible to design a model that specifies a set of constraints on the structure and datatypes of the BS Descriptions describing various bitstreams belonging to the same format.

This model can be written in the XML Schema language, it is then named a Schema.

In the context of BSDL, such a Schema is named a BS Schema. It carries specific semantics that can be used by a generic parser to parse the bitstream and generate its BS Description and vice-versa.

EXAMPLE The document below is an example of a BS Schema of an AVC bitstream corresponding to the BS Description example provide above. It specifies the structure and datatypes of the BSD above. For example, it specifies that the `startCode` element must be a four byte long hexadecimal string and that `nal_unit_type` must be a five bit long unsigned integer.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  targetNamespace="urn:mpeg:mpegb:example:AVC"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  xmlns:avc="urn:mpeg:mpegb:example:AVC"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  bs2:removeEmPrevByte="000003 0000"
  bs2:rootElement="avc:Bitstream"
  bs2:bsdlVersion="ISO/IEC 23001-5">

  <xs:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
             schemaLocation="../Schemas/MPEG-B-BSDL-1.xsd"/>
```

```
<xs:element name="Bitstream">
  <xs:complexType>
    <xs:sequence>
      <xs:choice maxOccurs="unbounded">
        <xs:element ref="avc:seqParameterSet"/>
        <xs:element ref="avc:pictParameterSet"/>
        <xs:element ref="avc:sliceIDR"/>
        <xs:element ref="avc:sliceNonIDR"/>
        <xs:element ref="avc:otherNAL"/>
      </xs:choice>
    </xs:sequence>
    <xs:attribute ref="bs1:bitstreamURI"/>
    <xs:attribute ref="bs1:bsdlVersion"/>
  </xs:complexType>
</xs:element>

<xs:element name="seqParameterSet"
       bs2:ifNext="0000000107" bs2:ifNextMask="FFFFFFFF1F"
       type="avc:NALUnitType"/>
<xs:element name="pictParameterSet"
       bs2:ifNext="0000000108" bs2:ifNextMask="FFFFFFFF1F"
       type="avc:NALUnitType"/>
<xs:element name="sliceIDR"
       bs2:ifNext="0000000105" bs2:ifNextMask="FFFFFFFF1F"
       type="avc:NALUnitType"/>
<xs:element name="sliceNonIDR"
       bs2:ifNext="0000000101" bs2:ifNextMask="FFFFFFFF1F"
       type="avc:NALUnitType"/>
<xs:element name="otherNAL"
       type="avc:NALUnitType"/>

<!-- ********************************************************************** -->
<xs:complexType name="NALUnitType">
  <xs:complexContent>
    <xs:extension base="avc:NALUnitHeaderType">
      <xs:sequence>
        <xs:element name="payload" type="avc:PayloadTypeNoStartCode"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="NALUnitHeaderType">
  <xs:sequence>
    <xs:element name="startCode" type="avc:StartCodeType" fixed="00000001"/>
    <xs:element name="forbidden_zero_bit" type="avc:b1" fixed="0"/>
    <xs:element name="nal_ref_idc" type="avc:b2"/>
    <xs:element name="nal_unit_type" type="avc:b5"/>
  </xs:sequence>
</xs:complexType>

<xs:simpleType name="StartCodeType">
  <xs:restriction base="xs:hexBinary">
    <xs:length value="4"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="PayloadTypeNoStartCode">
  <xs:simpleContent>
    <xs:extension base="avc:PayloadType">
```

```
        <xs:attribute ref="bs1:insertEmPrevByte"
                      default="000000 00000300
                               000001 00000301
                               000002 00000302
                               000003 00000303"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

  <xs:simpleType name="PayloadType">
    <xs:restriction base="bs1:byteRange">
      <xs:annotation>
        <xs:appinfo>
          <bs2:startCode value="00000001"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="b5">
    <xs:restriction base="xs:unsignedShort">
      <xs:maxExclusive value="32"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="b2">
    <xs:restriction base="xs:unsignedShort">
      <xs:maxExclusive value="4"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="b1">
    <xs:restriction base="xs:unsignedShort">
      <xs:maxExclusive value="2"/>
    </xs:restriction>
  </xs:simpleType>

</xs:schema>
```


4.4.3 BSDL

The role of XML Schema is to define a set of constraints on XML documents. BSDL is defined on top of XML Schema by adding a new functionality, which is to enable a generic BSDL processor to generate a BS Description for a given bitstream, and to re-generate a bitstream using a corresponding BS Description.

For this, and in a first step, BSDL defines a set of built-in datatypes and attributes in a schema named *Schema for BSDL-1 extensions* having the URN `urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS` as namespace. BS Schemas may then import this schema and use these built-in datatypes and attributes, which carry specific semantics in the context of bitstream generation.

In a second step, BSDL introduces a set of new language extensions in the form of attributes and schema components, which carry specific semantics in the context of description generation. These language constructs are added to XML Schema as application-specific annotations, and their syntax is specified in a schema named *Schema for Schema for BSDL-2 extensions* having the URN `urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS` as namespace. Note that since they are language extensions and contrarily to the *Schema for BSDL-1 extensions*, they are ignored in the XML Schema validation process.

Furthermore, BSDL defines a set of restrictions on the use of XML Schema components, which are specified in this subclause, but are not reflected in the schemas introduced above.

In the following, the set of extensions and restrictions required for the bitstream generation process is referred to as **BSDL-1**, and the set required for the BS Description generation process is referred to as **BSDL-2**, where BSDL-1 is a subset of BSDL-2.

Note that all the restrictions introduced by BSDL only apply to the elements that take part in the bitstream generation. There is no restriction on attributes since they are ignored by BSDL parsers, nor on potential elements declared in the schema, but not used in the BS Description. For these reasons, some BSDL restrictions can only be checked while generating the bitstream (BSDL-1) or the BS Description (BSDL-2).

Lastly, the BS Schema does not force the conformance of a bitstream to a given format, since this usually implies verifying the value of the parameters, while BSDL is only concerned on how they are binary-encoded. The bitstream conformance is therefore outside the scope of BSDL.

The figure below illustrates the relation between XML Schema and BSDL: XML Schema is a language providing syntax and semantics for expressing a Schema, itself specifying constraints on XML Documents. Similarly, BSDL provides syntax and semantics on top of XML Schema for expressing a BS Schema, itself specifying constraints on BS Descriptions. A BS Schema is a Schema in the XML Schema meaning and a BSD Description is an XML Document. Additionally, BSDL specifies semantics for parsing a bitstream and generating the corresponding BSD and vice versa.

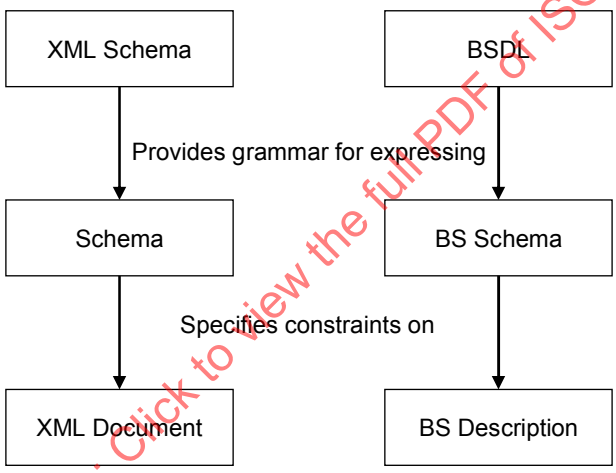


Figure 1 — Relation between XML Schema and BSDL

4.4.4 BSDL parsers: BSDtoBin and BintoBSD

A *BSDL parser* is a generic term encompassing the following two parsers:

— A ***BSDtoBin parser***, which is a generic processor using a BS Schema to parse BS Description and generate the corresponding bitstream.

— A ***BintoBSD parser***, which is a generic processor using a BS Schema to parse a bitstream and generate the corresponding BS Description.

BSDL-1 extensions may be used by both parsers, while BSDL-2 extensions are used by the BintoBSD parser only.

The figures below illustrate the functionality of BSDtoBin and BintoBSD parsers.

The BSDtoBin parser uses the information conveyed by the BS Schema to parse the BSD to generate the corresponding bitstream. In most cases, some elements of the BSD reference data segments in the original bitstream, which is then required by the the bitstream generation process.

Symmetrically, the BintoBSD parser uses the information conveyed by the BS Schema to parse a bitstream and generate its BSD.

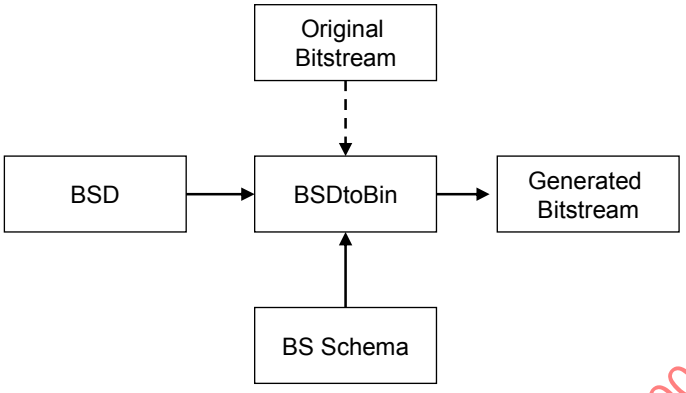


Figure 2 — BSDtoBin parser

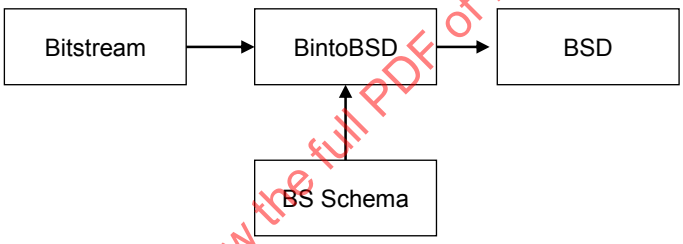


Figure 3 — BintoBSD parser

EXAMPLE For the BSD and corresponding BS Schema provided above, the BSDtoBin progressively parses the BSD and generates the output bitstream by encoding the element values according to their datatypes and iteratively appending them to the output bitstream. Firstly, it writes the hexadecimal value 0x00000001 (`startCode`), then the decimal value 0 encoded on one bit (`forbidden_zero_bit`), the decimal value 3 encoded on two bits (`nal_ref_idc`), the decimal value 7 encoded on five bits (`nal_unit_type`). For the `payload` element, it reads the 9 byte long data segment starting at offset 5 from the input bitstream and appends into the output bitstream. The input bitstream is identified by the `bs1:bitstreamURI` attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<Bitstream xmlns="urn:mpeg:mpegb:example:AVC"
           bs1:bitstreamURI="football_40_frames.h264"
           xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
           xsi:schemaLocation="urn:mpeg:mpegb:example:AVC SimpleAVC.xsd"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <seqParameterSet>
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>3</nal_ref_idc>
    <nal_unit_type>7</nal_unit_type>
    <payload>5 9</payload>
  </seqParameterSet>
  <!-- and so on ... -->
</Bitstream>
```

4.4.5 Advanced use of BS Descriptions

Extensibility of a BSD

BSDL specifies a limited set of constraints on a BS Description. However, this does not preclude an application to design a BS Description in such a way it may serve other purposes.

In particular, BSDL does not specify any constraint on the name of elements. An application is therefore free to design a BS Schema with given element names and possibly specific semantics.

Furthermore, BSDL ignores attributes other than those in the BSDL or XML Schema Instance namespaces. A BSD may therefore include any kind of information provided as attributes.

Lastly, BSDL provides an escaping mechanism identifying BSD sub-trees as to be ignored by BSDL. There is then no constraint set by BSDL on the indicated sub-trees.

In conclusion, an application can easily build on BSDL to add several layers of information to a BSD without interfering with the BSDL specific semantics.

Transformed BSD

A BSD may undergo transformation. In this case, it may not necessarily reflect the exact structure of the initial bitstream, as illustrated in the example below. Such BSD is said *transformed*.

EXAMPLE In the BSD below, the first element points to the first ten bytes of `myBitstream.bin`, the second element to the following ten bytes and so on.

```
<?xml version="1.0" encoding="UTF-8"?>
<bitstream bs1:bitstreamURI="myBitstream.bin"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS">
  <segment>0 10</segment>
  <segment>10 10</segment>
  <segment>20 10</segment>
  <segment>30 10</segment>
</bitstream>
```

It is possible to apply a transformation suppressing every second element, resulting in the transformed BSD shown bellow.

```
<?xml version="1.0" encoding="UTF-8"?>
<bitstream bs1:bitstreamURI="myBitstream.bin"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS">
  <segment>0 10</segment>
  <segment>20 10</segment>
</bitstream>
```

This new, transformed BSD does not fully reflect the structure of `myBitstream.bin`, since some segments do not appear. However, this is still a valid BSD, and it is possible to compile a new bitstream with BSDtoBin from this BSD, that is the concatenation of the first and third ten byte segments of `myBitstream.bin`.

It is therefore possible to modify a bitstream by first transforming its BSD and then generating with BSDtoBin a new bitstream from the transformed BSD. This method can be used in particular for the adaptation of scalable content and is further described in ISO/IEC 21000-7.

BSD pointing related to several bitstreams

BSDL uses the `bs1:byteRange` datatype to point to a data segment in the bitstream identified by the `bs1:bitstreamURI` attribute. However, a BSD is not restricted to point a single bitstream. By declaring elements pointing to several bitstreams, it is possible to multiplex them with BSDtoBin.

EXAMPLE In the example below, the first element points to the first ten bytes of `myBitstream1.bin`, the second element to the first twenty bytes of `myBitstream2.bin` and so on. The output bitstream generated by BSDtoBin applied to this BSD consists in the concatenation of the first ten bytes of `myBitstream1.bin`, the first twenty bytes of `myBitstream2.bin`, the following ten bytes of `myBitstream1.bin` and the following twenty bytes of `myBitstream2.bin`. This example shows how a BSD and BSDtoBin can be used to multiplex two or more bitstreams.

```
<?xml version="1.0" encoding="UTF-8"?>
<bitstream
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xsi:noNamespaceSchemaLocation="exampleMultiplexing.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <segment bs1:bitstreamURI="myBitstream1.bin">0 10</segment>
  <segment bs1:bitstreamURI="myBitstream2.bin">0 20</segment>
  <segment bs1:bitstreamURI="myBitstream1.bin">10 10</segment>
  <segment bs1:bitstreamURI="myBitstream2.bin">20 20</segment>

</bitstream>
```

4.5 Examples of applications for BSDL

The original application of BSDL was as part of a format independent framework for multimedia adaptation (see ISO/IEC 21000-7). By providing a generic XML abstraction for multimedia content, BSDL simplifies the adaptation process into one of XML transformation. The latter is well understood and easily implemented in numerous languages.

Other applications include

— Content multiplexing, by utilizing a BSD with that describes multiple bitstreams;

— Demultiplexing, by using a transformation engine to fragment the BSD;

— Augmenting a BSD with further semantic markup to facilitate multimedia content processing.

4.6 Relation with ISO/IEC 21000-7

BSDL was initially developed and specified in the context of Digital Item Adaptation (ISO/IEC 21000-7), in particular for adapting scalable multimedia resources. Advanced features have been added in ISO/IEC 21000-7:2005/Amd.2.

This International Standard extracts the BSDL specification from ISO/IEC 21000-7 (including Amendment 2) to make it available to other context beyond Digital Item Adaptation. No technical change was carried at this occasion. The BSDL specification in this standard is therefore technically the same as standardized in ISO/IEC 21000-7:2004 and its Amendments 1 and 2, but the text was significantly rewritten for better readability.

5 BSDL-1 and BSDtoBin

5.1 Constraints on BS Descriptions

This subclause specifies a set of constraints on BS Descriptions with respect to the BSDL-1 specification. An XML Document meeting these constraints is said BSDL-1 valid. BSDL-1 validity is required by the BSDtoBin parser.

NOTE Alternatively, the BSDL-1 validity does not guaranty that the bitstream generation process will succeed or produce the expected bitstream, in case for example if the indicated data ranges are incorrect.

The words in italic below refer to the XML and XML Schema terminologies [3][4]. In particular, *Post Schema Validated Infoset (PSVI)* is defined in [3] and refers to the augmented infoset which results from the XML Schema conformant processing defined in [3].

A **Bitstream Syntax Description** is said **BSDL-1 valid** if the following conditions are met:

- It is a *well-formed* XML document.
- There exists a schema conforming to W3C XML Schema, *validating* the BSD and producing a *Post Schema Validated Infoset (PSVI)*.
- No non-**ignored** element has a mixed content.
- The PSVI can assign to each non-**ignored**, non-empty, simple content element a **BSDL-1 valid datatype**.

A BSDL-1 valid datatype is one of:

- XML Schema built-in datatypes supported by BSDL as specified in 5.2.5
- BSDL built-in datatypes as specified in 5.2.6, or
- A datatype derived from the above types, using the derivation mechanisms specified in 5.2.4

An element in the BS Description is **ignored** if its *ignore* property is true as specified in 5.3.2.

NOTE The BSDL-1 validity does not define any constraint on the values of the start and length attributes of the `bs1:bitstreamSegment` datatype, or the content of the `bs1:byteRange` elements. In particular, it does not check if the data range indicated by these types fall outside of the bitstream range.

NOTE The BSDL-1 validity itself does not require importing the *Schema for BSDL-1 extensions*: this schema is required only if BSDL-1 datatypes or attributes are used in the BS Schema. In other words, any Schema is a valid BS Schema as long as it meets the constraints defined above.

NOTE A BSD element may be defined with an `xsd:anyType` or `xsd:anySimpleType` in the BS Schema and still be BSDL-1 valid as long as the type is overridden in the BSD with an `xsi:type` attribute indicating a BSDL-1 valid datatype.

NOTE Only **simple content elements** (i.e. which contain text data and no children elements), and elements with the `bs1:bitstreamSegment` datatype, directly write bits to the bitstream,

NOTE **Mixed content models** (i.e. element containing both text data and child elements) are not supported by BSDL, since in this case the character data inserted between the child elements have no type assigned by the schema.

NOTE **Attributes** declared in the BS Schema and used in the BSD are ignored by BSDtoBin except for attributes in BSDL-1 namespace that carry specific semantics.

5.2 Datatypes in BSDL

5.2.1 Overview

XML Schema defines a datatype as a 3-tuple consisting of (see subclause 2.1 of [4]):

- a set of distinct *values*, called its *value space*,
- a set of lexical representations, called its lexical space, and
- a set of *facets* that characterize properties of the value space, individual values or lexical items.

BSDL defines a datatype by adding a fourth component:

- a set of *binary representations*, called its *binary space*, uniquely defined for a given value.

BSDL restricts the use of XML Schema built-in datatypes to those for which a binary representation can be defined. For example, `xsd:integer` represents the mathematical concept for an unbounded integer. No implicit binary representation may be assigned to this type, which is therefore excluded from BSDL. On the other hand, `xsd:int` is derived from `xsd:integer` by restricting its value space to the values that may be represented on four bytes (`xsd:minInclusive=-2147483648` and `xsd:maxInclusive=2147483647`). BSDL imports this type and assigns a binary representation on four bytes.

Additional, BSDL specifies a set of built-in datatypes with specific semantics for the bitstream generation process.

For some XML Schema or BSDL datatypes such as `xsd:int`, the encoding length is fixed. They are said to have a *definite length*. For other datatypes such as `xsd:hexBinary`, the length may not be specified by the BS Schema, and in this case depends on the actual instance value. Such datatypes are said to have an *indefinite length*.

5.2.2 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. 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 5.3.5.

5.2.3 Facets

XML Schema facets characterize a *value space* along independent axes or dimensions (see 2.4 of [4]). Since BSDL does not consider the *values* of types but only their *binary representations*, XML Schema facets are ignored by BSDtoBin, except for the `xsd:maxExclusive` facet as explained below.

The `xsd:maxExclusive` indicates the number of bits with which an unsigned integer (`xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort` or `xsd:unsignedByte`) value is encoded by BSDtoBin. It is ignored by BSDtoBin for other datatypes.

The number of bits is calculated as the logarithm in base 2 of the `xsd:maxExclusive` value, rounded up to the next integer value.

5.2.4 Simple type derivation

As in XML Schema, the author of a schema may define his own datatypes by deriving them from built-in types. In the *derivation by restriction*, the facets constraining the values are ignored by BSDL, except for `xsd:maxExclusive` as seen above.

In the *derivation by list*, the different items of the list are binarized following the coding scheme of the base type and successively appended to the bitstream.

Lastly, the *derivation by union* is also supported by BSDL with the following behaviour: when no `xsi:type` attribute explicitly states the type in the instance, BSDtoBin considers the first member datatype in the `xsd:union`.

5.2.5 XML Schema built-in datatypes supported by BSDL

The list of XML Schema built-in datatypes supported in BSDL is given in the Table 2.

The second column indicates the number of bytes on which the type is encoded when it has a definite length. For the `xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort`, and `xsd:unsignedByte` datatypes, this corresponds to the encoding length when no `xsd:maxExclusive` facet is used. The third and fourth column indicate the applicable facets for BSDtoBin and BintoBSD. All other facets are consequently ignored by BSDtoBin (respectively BintoBSD). The behaviour of BSDtoBin with the `xsd:maxExclusive` facet has been explained in 5.2.3. The behaviour of BintoBSD with the XML Schema and BSDL facets is specified in 6.1.3.

Table 2 — List of XML Schema datatypes supported by BSDL

Datatype name	Encoding length	Applicable facets for BSDtoBin	Applicable facets for BintoBSD

`xsd:string`	Indefinite		`xsd:length`
			`bs2:length`
			`bs2:cdata`
			`bs2:escape`

`xsd:normalizedString`	Indefinite		`xsd:length`
			`bs2:length`
			`bs2:cdata`
			`bs2:escape`

`xsd:float`	4 bytes		
`xsd:double`	8 bytes		

`xsd:hexBinary`	Indefinite		`xsd:length`
			`bs2:length`
			`bs2:startCode`
			`bs2:endCode`

`xsd:base64Binary`	Indefinite		`xsd:length`
			`bs2:length`
			`bs2:startCode`
			`bs2:endCode`

`xsd:long`	8 bytes		
`xsd:int`	4 bytes		
`xsd:short`	2 bytes		
`xsd:byte`	1 byte		
`xsd:unsignedLong`	8 bytes	`xsd:maxExclusive`	`xsd:maxExclusive` `bs2:bitLength`
`xsd:unsignedInt`	4 bytes	`xsd:maxExclusive`	`xsd:maxExclusive`

			bs2:bitLength
xsd:unsignedShort	2 bytes	xsd:maxExclusive	xsd:maxExclusive bs2:bitLength
xsd:unsignedByte	1 byte	xsd:maxExclusive	xsd:maxExclusive bs2:bitLength

xsd:string and xsd:normalizedString are encoded in the output bitstream as US-ASCII.

NOTE 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 "
" or "
" instead of the Carriage Return character.

NOTE As any XML document, a BS Description may be itself encoded with different character encoding schemes such as UTF-8 or UTF-16. The encoding of an xsd:string or xsd:normalizedString by BSDtoBin is independent from the encoding of the BS Description.

xsd:float and xsd:double are defined in XML Schema as single (respectively double) precision 32-bit (respectively 64-bit) floating point type of IEEE 754-1985 [5], and should be encoded as such.

For xsd:hexBinary and xsd:base64Binary datatypes, the encoding length is determined by the lexical value of the element in the BSD, which shall conform to the constraint specified by the xsd:length or bs2:length facets if present in the schema.

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 5.2.6.

Other XML Schema types are excluded from BSDL, either because they are not used in a multimedia bitstream, such as types related to dates, time or duration, or because they have no implicit binary representation.

5.2.6 BSDL built-in datatypes

In addition to the XML Schema built-in datatypes listed above, BSDL provides a set of built-in datatypes listed in Table 3.

The number in the second column indicates the number of bytes or bits on which the type is encoded when it has a definite length. The third and fourth column indicate the applicable facets for BSDtoBin and BintoBSD. All other facets are consequently ignored by BSDtoBin (respectively BintoBSD). The behaviour of BSDtoBin with the xsd:maxExclusive facet has been explained in 5.2.3. The behaviour of BintoBSD with the XML Schema and BSDL facets is specified in 6.3.

The syntax of BSDL built-in datatypes is defined in the *Schema for BSDL-1 extensions*, except for the bs1:b1 to bs1:b32 datatypes that are specified in the *Schema for Unsigned Integers*, itself included in the *Schema for BSDL-1 extensions*.

As for XML Schema datatypes, each BSDL-1 built-in datatype can be uniquely identified via a URI constructed as follows:

— the base URI is the URI of the BSDL-1 namespace

— the fragment identifier is the name of the datatype

For example, to address the bs1:byteRange datatype, the URI is:

urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS#byteRange

Table 3 — List of BSDL built-in datatypes

Datatype name	Encoding length in bytes (with no facet)	Applicable facets for BSDtoBin	Applicable facets for BintoBSD
bs1:byteRange	Indefinite		bs2:length bs2:startCode bs2:endCode
bs1:bitstreamSegment	Indefinite		
bs1:stringUTF16NT bs1:stringUTF16BENT bs1:stringUTF16LENT bs1:stringUTF8NT bs1:stringUTF16 bs1:stringUTF16BE bs1:stringUTF16LE bs1:stringUTF8	Indefinite		xsd:length bs2:length bs2:cdata bs2:escape
bs1:longLE	8 bytes		
bs1:intLE	4 bytes		
bs1:shortLE	2 bytes		
bs1:unsignedLongLE	8 bytes		
bs1:unsignedIntLE	4 bytes		
bs1:unsignedShortLE	2 bytes		
bs1:align32	Between 0 and 32 bits		
bs1:align16	Between 0 and 16 bits		
bs1:align8	Between 0 and 8 bits		
bs1:unsignedExpGolomb bs1:signedExpGolomb	Indefinite number of bits		
bs1:b1 - bs1:32	1 to 32 bits		

bs1:byteRange indicates 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.

NOTE From an XML Schema point of view, bs1:byteRange is defined as a list of integers. When applied to bs1:byteRange, 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, shall be used with bs1:byteRange to indicate the length in bytes of the data segment to read.

NOTE The *bitstreamURI* property may contain a fragment identifier and thus point to a resource fragment. In this case, the offset information of bs1:byteRange is relative to the start of the fragment and not of the entire resource, that is, 0 is the first byte of the fragment.

bs1:bitstreamSegment indicates 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.

NOTE The *bitstreamURI* property may contain a fragment identifier and thus point to a resource fragment. In this case, the offset information of `bs1:bitstreamSegment` is relative to the start of the fragment and not of the entire resource, that is, 0 is the first byte of a fragment.

NOTE The `bs1:bitstreamSegment` datatype is provided for compatibility with gBS Schema [12] and should not be used for BS Description generation (BSDL-2).

NOTE XML Schema uses the term "datatype" for simple types. Strictly speaking, `bs1:bitstreamSegment` is therefore not a "datatype", but a built-in *complex type*. For simplification, however, this specification considers it as a BSDL-1 datatype.

`bs1:longLE`, `bs1:intLE`, `bs1:shortLE`, `bs1:unsignedLongLE`, `bs1:unsignedIntLE`, `bs1:unsignedShortLE` are 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.

`bs1:stringUTF8`, `bs1:stringUTF16BE`, `bs1:stringUTF16LE` and `bs1:stringUTF16` represent string 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.

`bs1:stringUTF8NT`, `bs1:stringUTF16BENT`, `bs1:stringUTF16LENT` and `bs1:stringUTF16NT` have the same semantics as `bs1:stringUTF8`, `bs1:stringUTF16BE`, `bs1:stringUTF16LE`, and `bs1:stringUTF16`, but with a terminating null character.

NOTE 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 must generate it into the output bitstream.

NOTE When an `xsd:length` or `bs2:length` facet constrains one of the null-terminated string datatypes, the facet value specifies the string length in characters, excluding the null character.

`bs1:align8`, `bs1:align16` and `bs1:align32` enable reading a value from the input bitstream and writing padding bits to the output bitstream until it is aligned on a 1 byte (for `bs1:align8`), 2 byte (for `bs1:align16`) or 4 byte (for `bs1: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.

`bs1:unsignedExpGolomb` and `bs1:signedExpGolomb` correspond to the unsigned and signed exp-golomb encoding specified in ISO/IEC 14496-10 [10]. For convenience, the encoding description is provided in Annex C.

`bs1:b1` to `bs1:b32` are provided for convenience and represent unsigned integer values with most significant bit first with an encoding length from 1 to 32 bits.

5.3 BSDL-1 attributes

5.3.1 Overview

This subclause specifies a set of attributes that carry specific semantics for the bitstream generation process. BSDL-1 attributes are declared in BS Schemas and used in BSD to provide information to the BSDtoBin parser.

It is also possible to declare the BSDL-1 attributes with a default or fixed value in the BS Schema without the need for adding them to the BSD: since BSDtoBin processes the PSVI of the BSD, an attribute declared with a default or fixed value in the BS Schema has the same effect from a PSVI perspective (and thus for BSDtoBin) as if the same attribute had been explicitly used in the BSD.

Some BSDL-1 attributes enrich the BSD PSVI by assigning **abstract properties** to the elements that may be inherited by child elements and used by the BSDtoBin parser. They are specified along with the corresponding attribute in the subclauses below. 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 Description 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 (for the root element), it takes the default value for the document as specified for each property.

EXAMPLE In the example below, the `bs1:addressUnit` attribute is applied to the `el1` element and assigns a property named *addressUnit* to the `el1` element, which is inherited by the `el2` element. According to its enriched PSVI, the *addressUnit* property of `el2` is therefore set to "`bit`", and BSDtoBin will process `el2` accordingly. Note that from a strict XML point of view, the `bs1:addressUnit` attribute only adds information to `el1`, and is unknown to `el2`.

```
<!-- Description example -->
<el1 bs1:addressUnit="bit">
  <el2>2 5</el2>
</el1>
```

The syntax of BSDL-1 attributes is specified by the *Schema for BSDL-1 Extensions*. When applicable, some additional constraints that cannot be expressed by the *Schema for BSDL-1 Extensions* are specified in text.

For each BSDL-1 attribute, the subclauses below specify:

- Associated property when applicable
- Additional constraints on syntax when applicable
- Semantics for BSDtoBin

5.3.2 bs1:ignore attribute

Introduction

The `bs1:ignore` attribute instructs the BSDtoBin parser to ignore an element (and its descendants, if any) in the BSD. This allows information to be added to the BSD in the form of elements and attributes in any namespace, without disrupting the BSDtoBin process.

Associated property

The boolean *ignore* property is associated with the `bs1:ignore` attribute. If this property has the value *true*, BSDtoBin shall ignore the element and its descendants, if any. Otherwise, BSDtoBin shall process the element. The default value for the property is *false*.

As an exception to the default inheritance mechanism specified in 5.3.1, the `bs1:ignore` attribute is ignored if the element's parent already has the ignore property set to true. That is, once an element is tagged as ignored by the `bs1:ignore` attribute, then all its descendants shall also be ignored, regardless of the potential addition of other `bs1:ignore` attribute further down in the sub-tree.

Syntax

No further constraint beyond *Schema for BSDL-1 Extensions*.

Semantics

Attribute indicating, when set to true, that the BSDL parser should ignore the current element and its descendants. This allows the BS Schema author to add application-specific information in the form of an XML fragment in the BS Description without interfering with the bitstream generation process. Note that this attribute may also be declared in the BS Schema with an `xsd:fixed` or `xsd:default` attribute set to `true`. In this case, the BSDL parsers will consider its default value and hence ignore the corresponding element.

Example

The following example illustrates the use of the `bs1:ignore` attribute. In BSDL-1, the `bs1:ignore` attribute is used in the BS Description to indicate, if set to true, that the element it characterizes should be skipped by the BSDtoBin parser. Note that it is also possible to set its default value in the BS Schema. In the example below, the `someOtherAnnotation` element is declared in the schema with a default value for its attribute `bs1:ignore` set to true. It is thus not necessary to repeat the attribute in the BS Description.

This mechanism allows adding application-specific information in the description without interfering with the bitstream generation. Such information may be the output of a post-processing step or any annotation mechanism and, unlike the regular description elements, is not read from the bitstream. If the BS Schema is also used for BSDL-2, it is then necessary to indicate that the BintoBSD parser should skip the element declaration by setting the default or fixed value of the attribute `bs1:ignore` to true. In the example below, the `someOtherAnnotation` element declaration is skipped by the BintoBSD parser. On the other hand, the `someAnnotation` element declaration cannot be used in a BSDL-2 schema.

```
<!-- Schema example -->
<xsd:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
            schemaLocation="BSDL-1.xsd"/>

<!-- This element declaration should not be used for BintoBSD -->
<xsd:element name="someAnnotation">
  <xsd:complexType>
    <!-- Whatever simple or complex content here -->
    <xsd:attribute ref="bs1:ignore"/>
  </xsd:complexType>
</xsd:element>

<!-- This element is ignored by BintoBSD -->
<xsd:element name="someOtherAnnotation">
```

```
  <xsd:complexType>
    <!-- Whatever simple or complex content here -->
    <xsd:attribute ref="bs1:ignore" default="true"/>
  </xsd:complexType>
</xsd:element>
<!-- and so on -->
```

```
<!-- Description example -->
<rootElement>
  <someAnnotation bs1:ignore="true">This text is ignored by BSDtoBin
  </someAnnotation>
  <someOtherAnnotation>This text is also ignored by BSDtoBin
  </someOtherAnnotation>
</rootElement>
```

5.3.3 bs1:bitstreamURI attribute

Introduction

A BSD element with a `bs1:byteRange` or `bs1:bitstreamSegment` datatype points to a data segment in a bitstream. The `bs1:bitstreamURI` attribute allows specifying the URI of this bitstream.

As specified in 4.4.5, a BSD may contain elements pointing to different bitstreams. In this case, several `bs1:bitstreamURI` attributes may be used to identify the different described bitstreams.

Associated property

The *bitstreamURI* property indicates the absolute URI of the described bitstream and is associated with 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 (for an 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 BS Description document.

NOTE This means that if a `bs1:bitstreamURI` attribute is specified for the BSD root element (either explicitly in the BS Description, or by default in the BS Schema) with a relative value, the resulting bitstreamURI property is obtained by resolving the relative URI against the absolute BSD URI.

NOTE The URI may contain a fragment identifier, in which case it points to a resource fragment.

Syntax

No further constraint beyond *Schema for BSDL-1 Extensions*.

Semantics

Attribute specifying the bitstreamURI property, according to the mechanism specified above.

NOTE See also the URI specification [7], and especially subclause 5.2 and Annex C.1 for the relative URI resolution mechanism.

Example

In the following example, let `file://myDir/myBSD.xml` be the absolute URI of the BSD.

The `bs1:bitstreamURI` attribute of the root element `el0` specifies a relative URI, which is resolved against the absolute URI of the BSD document. The *bitstreamURI* property is therefore `file://myDir/bitstreams/`. The `bs1:bitstreamURI` attribute of `el1` specifies a relative URI, which is resolved against the *bitstreamURI* property of its parent element `el0`. The *bitstreamURI* property of `el1` is

therefore `file://myDir/bitstreams/myBitstream1.bin`. The element `el11` inherits this property. Similarly, the *bitstreamURI* property of `el2` and `el22` is `file://myDir/bitstreams/myBitstream2.bin`.

```
<?xml version="1.0" encoding="UTF-8"?>
<el0
  bs1:bitstreamURI="bitstreams/"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exampleBitstreamURI.xsd">
  <el1 bs1:bitstreamURI="myBitstream1.bin">
    <el11>0 10</el11>
  </el1>
  <el2 bs1:bitstreamURI="myBitstream2.bin">
    <el22>0 10</el22>
  </el2>
</el0>
```

5.3.4 bs1:addressUnit attribute

Introduction

The `bs1:byteRange` and `bs1:bitstreamSegment` datatypes point to a data segment in a bitstream identified by its offset and length. The `bs1:addressUnit` attribute specifies the unit (bit or byte) used for addressing the segment.

Associated property

The *addressUnit* property is associated with 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*.

Syntax

No further constraint beyond *Schema for BSDL-1 Extensions*.

Semantics

The `bs1:addressUnit` attribute specifies the addressing unit (bit or byte) used by `bs1:byteRange` and `bs1:bitstreamSegment` datatypes.

Example

In the example below where the type of `el1` is `byteRange`, `el1` points to the first bits of `myBitstream.bin`.

```
<?xml version="1.0" encoding="UTF-8"?>
<el0 bs1:bitstreamURI="myBitstream1.bin" bs1:addressUnit="bit"
     xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS">
  <el1>0 10</el1>
</el0>
```

5.3.5 bs1:codec attribute

Introduction

As described in 5.2.2, BSDL allows applications to define additional datatypes in a BSDtoBin and/or BintoBSD parser. Such datatypes are identified in a BS Schema or BSD by a URI value in a `bs1:codec` attribute.

Associated property

None.

Syntax

The `bs1:codec` attribute may appear as an attribute declaration in a `xsd:complexType` as long as the type has `xsd:simpleContent`, or as an attribute on any element in a BS Description.

NOTE For BSDtoBin and BintoBSD, the type specified by the `bs1:codec` attribute supersedes the one declared in the schema.

An `xsd:complexType` declaring a `bs1:codec` attribute may extend or restrict any XML Schema type, without regard to the restrictions imposed by subclause 5.2.

Semantics

A BSDtoBin or BintoBSD parser shall not directly process a type that declares a `bs1:codec` attribute. The parser shall delegate such processing to the `codec` specified by the attribute value.

NOTE 1. The recommended usage of `bs1:codec` is that it contains a URI that is interpreted as the namespace of the library, and a fragment identifier equal to the local name of the data type.

NOTE 2. Because a generated BSD should be XML Schema-valid, the lexical value generated by the `codec` should conform to the datatype indicated by the BS Schema.

NOTE 3. The behaviour of a BintoBSD or BSDtoBin processor that does not provide an implementation of any given codec is not specified here.

NOTE 4. A BS Schema may directly provide the implementation of a data type defined by `bs1:codec` using the means specified in B.2.1.

Example

The following example illustrates how to use `bs1:codec` in a BS Schema and in a BS Description. The BS Schema declares a `bs1:codec` attribute for the `myType` type with a default value.

When parsing the BSD, the BSDtoBin parser builds the Post Schema Validated Infoset of the BSD. According to the PSVI, `e11` has a `bs1:codec` attribute with the default value. For `e12`, the `bs1:codec` value specified in the BSD supersedes the default value specified in the BS Schema. Consequently, the BSDtoBin parser shall encode the `e11` element according to the `myUserType1` datatype, and `e12` according to `myUserType2`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  bs2:requiredExtensions="urn:example:myLib#myUserType1"
  id="exampleCodec.xsd">

  <xs:import namespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
    schemaLocation="../Schemas/MPEG-B-BSDL-1.xsd"/>
```

```
  <!-- Example of BS Schema snippet using the bs1:codec attribute -->
  <xs:element name="el0">
    <xs:complexType>
      <xs:choice maxOccurs="unbounded">
        <xs:element name="el1" type="myType"/>
        <xs:element name="el2" type="myType"/>
      </xs:choice>
      <xs:attribute ref="bs1:requiredExtensions"/>
    </xs:complexType>
  </xs:element>

  <xs:complexType name="myType">
    <xs:simpleContent>
     <xs:extension base="xs:int">
       <xs:attribute ref="bs1:codec" default="urn:example:myLib#myUserType1"/>
     </xs:extension>
    </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<el0
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="exampleCodec.xsd"
  bs1:requiredExtensions="urn:example:myLib#myUserType1
    urn:example:myLib#myUserType2">

  <el1>1</el1>
  <el2 bs1:codec="urn:example:myLib#myUserType2">2</el2>
</el0>
```


5.3.6 bs1:requiredExtensions attribute

Introduction

The `bs1:codec` attribute allows additional datatypes to be used in a BS Schema. However, the behaviour of a BSDtoBin parser is not specified when the implementation of such datatypes are not available. The `bs1:requiredExtensions` attribute therefore specifies the list of additional datatypes that are required to run BSDtoBin on a BSD.

NOTE A companion attribute `bs2:requiredExtensions` (see 6.2.10) specifies the list of additional datatypes required to successfully run the BintoBSD parser with a BS Schema that uses such datatypes.

Associated property

None.

Syntax

The `bs1:requiredExtensions` attribute may be used in a BSD and/or in a BS Schema when declared with a default or fixed value.

Semantics

The `bs1:requiredExtensions` attribute indicates the additional datatypes used by the element to which it is attached, and its descendents, if any. Its value is a list of URIs. Each URI indicates either a single datatype or a library of datatypes. In the first case, the URI shall contain a fragment identifier pointing to datatype name itself. In the second case, the URI shall identify the library.

The `bs1:requiredExtensions` provides additional information within a BSD or BS Schema, but does not imply any normative behaviour of the BSDtoBin parser. The behaviour of a BSDtoBin parser is not specified when the implementation of the specified datatypes are not available.

In case of conflict between the values specified in the BSD and the BS Schema, the BSD value supersedes the default value declared in the BS Schema.

Example

In the example shown in subclause 5.3.5, the `bs1:requiredExtensions` attribute indicates that the `myUserType1` and `myUserType2` type-specific codecs of the library `urn:example:myLib` are required to run the BSDtoBin parser on the BSD.

5.3.7 bs1:insertEmPrevByte attribute

Introduction

Some coding formats (for example ISO/IEC 14496-10 Advanced Video Coding) use a mechanism to prevent the unintended emulation of start codes. For this, one or several additional bytes are inserted in the bitstream whenever the sequence of bytes produced by the encoding mechanism would otherwise be equivalent to a start code.

The `bs1:insertEmPrevByte` attribute instructs the BSDtoBin parser to apply such an emulation prevention mechanism. This attribute can also be used to instruct BSDtoBin to cease the insertion of emulation prevention bytes.

NOTE A companion attribute `bs2:removeEmPrevByte` (see 6.2.5) specifies the equivalent mechanism for removing emulation prevention bytes when parsing a bitstream with BintoBSD.

Associated property

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*.

Syntax

The `bs1:insertEmPrevByte` must contain an even number (possibly null) of hexadecimal strings.

Semantics

The *insertEmPrevByte* property indicates that the BSDtoBin parser shall apply the emulation prevention mechanism to the current element. The value of the property shall be an even number of hexadecimal values. For each pair of values, the first specifies the byte sequence that requires the insertion of emulation prevention bytes and the second specifies the byte sequence that shall be written to the output bitstream instead.

When the attribute is empty, no emulation prevention byte is inserted.

Example

According to the BS Schema shown in 4.4.2, any `payload` element in the BSD shown in 4.4.1 has by default a `bs1:insertEmPrevByte` attribute with the following value: "`000000 00000300 000001 00000301 000002 00000302 000003 00000303`". When processing the payload element, the BSDtoBin parser

shall insert a insert a "0x03" byte whenever the output byte string is one of "000000", "000001" or "00000303".

The BSD shown below demonstrates how the insertion mechanism may be switched on and off and provides the same result: the insertion mechanism is switched on for the Bitstream element, then off for seqParameterSet (empty string), and then on again for payload.

```
<?xml version="1.0" encoding="UTF-8"?>
<Bitstream
  xmlns="urn:mpeg:mpegb:example:AVC"
  bs1:bitstreamURI="football_40_frames.avc"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xsi:schemaLocation="urn:mpeg:mpegb:example:AVC SimpleAVC.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  bs1:insertEmPrevByte="000000 00000300 000001
      00000301 000002 00000302 000003 00000303">

  <seqParameterSet bs1:insertEmPrevByte="">
    <startCode>00000001</startCode>
    <forbidden_zero_bit>0</forbidden_zero_bit>
    <nal_ref_idc>3</nal_ref_idc>
    <nal_unit_type>7</nal_unit_type>
    <payload bs1:insertEmPrevByte="000000 00000300 000001 00000301
      000002 00000302 000003 00000303">5 9</payload>
  </seqParameterSet>

  <!-- and so on... -->

</Bitstream>
```

5.3.8 bs1:bsdlVersion attribute

Introduction

The bs1:bsdlVersion allows indicating what version of BSDL a BSD conforms to with respect to BSDL-1 validity. This attribute allows versioning of BSD with respect to potential future versions of BSDL.

NOTE A companion attribute bs2:bsdlVersion (see 6.2.9) specifies what version of BSDL a BS Schema conforms to with respect to BSDL-2 validity.

Associated property

None.

Syntax

No further constraint beyond *Schema for BSDL-1 Extensions*.

Semantics

The bs1:bsdlVersion attribute indicates the version of BSDL that this BSD conforms to and hence the version of BSDtoBin processor required. It is recommended to use this attribute in the BSD root element to allow a BSDtoBin processor check whether it may process the BSD.

The referred version is the value of the version attribute of the xsd:schema component of the *Schema for BSDL-1 Extensions*.

The `bs1:bsdlVersion` provides additional information in a BSD or BS Schema, but does not imply any normative behaviour of the BSDtoBin parser.

Example

The example shown in 4.4.1 indicates that the BSD conforms to the "ISO/IEC 23001-5" version of BSDL.

5.4 bs1:script element

Introduction

As described in subclause 5.3.5, a BS Schema may use additional datatypes. However, interoperability is then limited to those BintoBSD parsers supporting the referenced extension datatypes.

When a BintoBSD parser supports ECMAScript as described in Annex A, it is possible to provide an implementation of the extension types as a script within the BS Schema itself and thus to maintain full interoperability.

The `bs1:script` component acts as a wrapper for embedding the script implementation of the extension datatype.

BSDL does not mandate the support for script implementation for extension datatypes, but Annex A specifies an optional mechanism for ECMAScript implementation of datatypes.

Syntax

The `bs1:script` element shall be added to the `xsd:schema` component via the `xsd:annotation`/`xsd:appinfo` combination. Several `bs1:script` may be used to declare several datatypes, where each `bs1:script` shall contain the implementation of a single extension datatype.

Semantics

The `bs1:script` element provides a wrapper for scripts associated with the BSDtoBin and/or BintoBSD parsing process.

NOTE Other than the optional Annex A, the use of `bs1:script` elements is not normatively defined.

NOTE Unlike BSDL-1 attributes, the `bs1:script` element does not augment the BSD infoset. However, it is specified in the BSDL-1 namespace because the mechanism it introduces is used by both BSDtoBin and BintoBSD.

Example

See Annex A.

5.5 Schema for BSDL-1 Extensions

```
<?xml version="1.0"?>
<!-- Bitstream Syntax Description Language ISO/IEC 23001-5 -->
<!-- Schema for BSDL-1 extensions -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
        targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
        version="ISO/IEC 23001-5" id="MPEG-B-BSDL-1.xsd">

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

```
<include schemaLocation="MPEG-B-UnsignedIntegers.xsd"/>

<!-- ################################### -->
<!--          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 name="insertEmPrevByte" type="bs1:hexBinaryStrings"/>
<attribute name="bsdlVersion" type="string"/>

<!-- 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>

<!-- ################################### -->
<!--       ECMAScript element            -->
<!-- ################################### -->
<element name="script">
  <complexType>
    <simpleContent>
      <extension base="string">
        <attribute name="id" type="ID" use="required"/>
      </extension>
    </simpleContent>
  </complexType>
</element>

<!-- ################################### -->
<!--       BSDL Built-in Datatype        -->
<!-- ################################### -->
<simpleType name="byteRange" id="byteRange">
  <restriction>
    <simpleType>
      <list itemType="nonNegativeInteger"/>
    </simpleType>
    <length value="2" fixed="true"/>
  </restriction>
</simpleType>

<complexType name="bitstreamSegment" id="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" id="stringUTF16NT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16BENT" id="stringUTF16BENT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16LENT" id="stringUTF16LENT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF8NT" id="stringUTF8NT">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16" id="stringUTF16">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16BE" id="stringUTF16BE">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF16LE" id="stringUTF16LE">
  <restriction base="string"/>
</simpleType>
<simpleType name="stringUTF8" id="stringUTF8">
  <restriction base="string"/>
</simpleType>

<simpleType name="longLE" id="longLE">
  <restriction base="long"/>
</simpleType>
<simpleType name="intLE" id="intLE">
  <restriction base="int"/>
</simpleType>
<simpleType name="shortLE" id="shortLE">
  <restriction base="short"/>
</simpleType>
<simpleType name="unsignedLongLE" id="unsignedLongLE">
  <restriction base="unsignedLong"/>
</simpleType>
<simpleType name="unsignedIntLE" id="unsignedIntLE">
  <restriction base="unsignedInt"/>
</simpleType>
<simpleType name="unsignedShortLE" id="unsignedShortLE">
  <restriction base="unsignedShort"/>
</simpleType>

<simpleType name="align32" id="align32">
  <restriction base="hexBinary">
    <length value="4"/>
  </restriction>
</simpleType>
<simpleType name="align16" id="align16">
  <restriction base="hexBinary">
    <length value="2"/>
  </restriction>
</simpleType>
<simpleType name="align8" id="align8">
  <restriction base="hexBinary">
    <length value="1"/>
  </restriction>
</simpleType>
<simpleType name="unsignedExpGolomb" id="unsignedExpGolomb">
```

```
    <restriction base="unsignedInt"/>
  </simpleType>
  <simpleType name="signedExpGolomb" id="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"?>
<!-- Bitstream Syntax Description Language ISO/IEC 23001-5 -->
<!-- Schema for BSDL-1 extensions : Unsigned Integer Datatypes Schema -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        version="ISO/IEC 23001-5" id="MPEG-B-UnsignedIntegers.xsd">

  <annotation>
    <documentation>
      Schema for BSDL-1 extensions: Unsigned Integers
    </documentation>
  </annotation>

  <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>
```

5.6 Bitstream generation with BSDtoBin

5.6.1 Processing model

This subclause describes the recursive method producing a bitstream from its description (Figure 4).

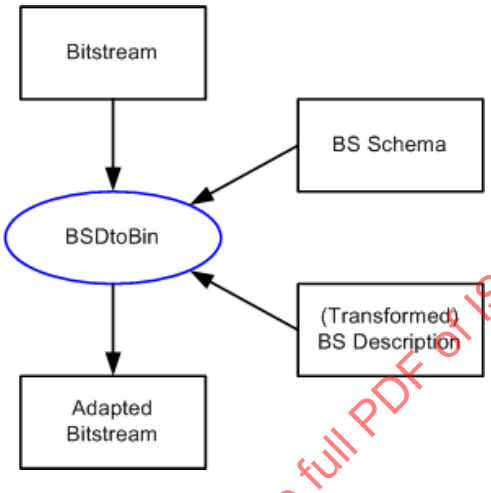


Figure 4 — Architecture of bitstream generation using BSDtoBin

It is reminded that no mixed content is allowed in a bitstream syntax description. Furthermore, any type definition left open in the BS Schema (e.g., by using `xsd:anyType` or `xsd:anySimpleType`) must be superseded in the instance by a relevant type with the `xsi:type` attribute. Lastly, prior to generating a bitstream, the BS Description must be valid with respect to its schema in the XML Schema meaning. Following these rules, a type definition in the schema may be assigned to each simple content element of the instance and thus specify an encoding scheme for the content. Furthermore, when an element of the BS Description is empty and its declaration in the schema specifies a value constraint (fixed or default attribute), the BSDtoBin processes this default value.

BSDtoBin parses the description following the navigation path of an XML document. Each non-empty, simple content element produces a segment of data in the output bitstream or a list of parameters in the case of an `xsd:list`. A complex content element produces a segment of data, which is itself the result of the contributions of its child elements.

For each element, attributes other than within the XML Schema Instance or BSDL namespaces are ignored. Furthermore, it is reminded that the tag name itself has no impact on the bitstream generation. For a simple content, the BSDtoBin parser reads the text content of the element encodes it according to its datatype and appends it to the output bitstream. For a complex content, the child elements are recursively processed in the same order they appear in the instance.

The encoding scheme of each simple type is implicit in its definition. In the case of a `bs1:byteRange`, BSDtoBin reads the resource indicated by the bitstream URI of the element and copies the segment of data indicated by the offset and length to the output bitstream. In the case of `bs1:bitstreamSegment`, BSDtoBin copies the relevant data segment only if the current element has no child element.

6 BSDL-2 and BintoBSD

6.1 Overview

6.1.1 Introduction

BSDL-2 introduces language extensions over XML Schema that are required by BintoBSD to parse the bitstream and generate the BS Description.

While BSDL-1 extensions consist in type definitions and attribute declarations that can be used in BS Schemas to carry specific semantics in the context of bitstream generation, BSDL-2 extensions introduce new language constructs over XML Schema, such as conditional statements.

These language extensions consist in:

- a set of BSDL-2 attributes
- a set of BSDL-2 components, including in particular:
  - a set of BSDL-2 facets

The syntax of these extensions is partially defined in the *Schema for Schema for BSDL-2 extensions*. This schema document is similar to the *Schema for Schema* of XML Schema in that it specifies the syntax of the language elements themselves. Additional constraints on the syntax are specified for each BSDL-2 attribute or component.

These constraints are said static in the sense they can and should be checked on the BS Schema before and independently from the BS Description generation process. The input bitstream is thus not required for checking the constraints.

The BSDL-2 extensions are added to the BS Schemas as XML Schema annotations and thus do not intervene in the schema validation. From a schema validation point of view and unlike the *Schema for BSDL-1 Extensions*, the *Schema for Schema for BSDL-2 extensions* does not need to be imported by a BS Schema.

6.1.2 Annotation mechanisms of XML Schema

XML Schema provides two ways of adding application-specific annotations. Firstly, all schema components can contain an `xsd:annotation` component, which itself can contain one or several `xsd:appinfo` components, intended as a placeholder for application-specific information. Secondly, all XML Schema components allow attributes with non-schema namespace, which leaves the possibility to add new attributes to some schema components by qualifying them with a specific namespace.

BSDL-2 uses this annotation mechanism for adding BSDL-2 language extensions to XML Schema:

- *BSDL-2 components* are new schema components similar to XML Schema components. They are added to the schema via `xsd:annotation` and `xsd:appinfo`.
- *BSDL-2 attributes* are new attributes characterising an XML Schema component.

The two types of extensions are qualified with the BSDL-2 namespace. Their syntax is specified in the *Schema for Schema for BSDL-2 extensions* given in subclause 6.4.3. Note that in XML Schema, application-specific information does not intervene in the XML validation, which means an XML Schema validator will not validate the content of the `xsd:appinfo` schema component against its schema. The same applies to attributes with non-schema namespace. It is therefore up to the BSDL parsers to check that the attributes and schema components with BSDL-2 namespace follow the syntax specified in the *Schema for schema for BSDL-2 extensions*.

Since an `xsd:annotation` component may contain several `xsd:appinfo`, it is possible to add BSDL-2 components to a schema component by including them in the same `xsd:appinfo` or in several `xsd:appinfo` in the same `xsd:annotation`. There is no difference in semantics: in both cases, the BSDL-2 components are considered by BintoBSD as child elements of the annotated schema component.

6.1.3 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 own facets, they are declared as BSDL-2 components added to the `xsd:restriction` component via the annotation mechanism, i.e., the `xsd:annotation`/`xsd:appinfo` combination.

Like the XML Schema facets, a BSDL-2 facet constrains an XML Schema or BSDL-1 datatype and specifies how BintoBSD must parse this datatype.

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 the bitstream.

The subclause 6.3 lists the BSDL-2 facets. For each facet, it specifies the XML Schema and BSDL-1 datatypes it applies to.

6.1.4 XPath expressions

One of the key features of BSDL-2 is the use of variables, the value of which should be read from the bitstream. This is required for example when a bitstream contains a segment of data, the length of which is given by another field previously found in the bitstream. In order to parse this segment, the BintoBSD Parser needs to retrieve this field and evaluate its value. It is therefore necessary to locate a downstream parameter (i.e., already read from the bitstream) and evaluate its value.

While parsing a bitstream, the BintoBSD parser progressively instantiates the BS Description. As soon as a field is parsed, the corresponding XML element is instantiated and added to the Description. A data field which length or number of occurrences depend on a field found earlier in the bitstream can thus be read by locating and evaluating this parameter in the partially instantiated BS Description.

For this, BSDL declares variables by using XPath expressions, as specified in XPath 1.0 [8]. XPath defines four basic types (see clause 1 of the recommendation):

- *node-set* (an unordered collection of nodes without duplicates)
- *boolean* (true or false)
- *number* (a floating-point number)
- *string* (a sequence of UCS characters)

Both BSDL-2 attributes and components may use such variables. For example, the `bs2:nOccurs` attribute specifies the number of occurrences of a particle and the `bs2:length` facet specifies the length of a datatype such as `xsd:hexBinary`.

Depending on the BSDL-2 attribute or component, a number may be cast into an integer. XPath expressions are evaluated relatively to a reference node, called *context node*. For each BSDL-2 attribute or component using an XPath expression, the text specifies:

— the context node

— the expected type of the XPath expression result

An XPath expression may use qualified names. In this case, the namespace binding context is the context of the schema component where this expression is used.

NOTE Prefixes are only local abbreviations for a namespace. Therefore, prefixes used in the BS Schema do not need to be the same used in the instantiated BS Description.

6.1.5 bs2:log2() XPath function

XPath specifies a Core Function Library. In addition, BSDL-2 specifies a new function named `bs2:log2()` as follows: this function returns the logarithm in base 2 of the input argument. The input argument and output value are XPath *numbers*, *i.e.* floating-point values. 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.

6.1.6 XPath variables assignment

XPath expressions may use variables: they are referenced by their qualified name preceded by the "`$`" character. However, XPath in itself does not specify how to assign a value to a variable.

BSDL specifies several ways of assigning a value to an XPath variable.

— The `bs2:parameter` component (see 6.4.2) assigns a constant value to a variable.

— The `bs2:assignPost` attribute (see 6.2.8) assigns the value of a simple content element to a variable.

— The `bs2:assignPre` attribute (see 6.2.8) assigns one or several values read directly upstream to variables.

— The `bs2:variable` component (see 6.4.4) assigns the result of an XPath expression to a variable.

NOTE Variables are declared and assigned in a same step.

All variables declared using these features shall have global scope. That is, they may be referenced from any XPath expression within a BS Schema. However, the variable may have an undefined value, if it is dereferenced before the BintoBSD parser has assigned a value to it.

NOTE 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.

XPath variables are declared and referenced with a qualified name.

When declaring an XPath variable with one of the BSDL-2 components and attributes listed above, the prefix is resolved against the local namespace binding context of the declaring element (`bs2:parameter` or `bs2:variable`) or the element containing the attribute `bs2:assignPost` or `bs2:assignPre`.

When referencing an XPath variable, the prefix is resolved against the local namespace binding context of the element where the XPath expression is used.

NOTE It is therefore possible to declare a variable in another namespace than the target namespace of the schema.

6.2 BSDL-2 attributes

6.2.1 bs2:nOccurs

Introduction

The `bs2:nOccurs` attribute specifies the number of occurrences of a particle with a dynamic value, in contrast to `minOccurs` and `maxOccurs` which use constant values. The number of occurrences is provided as an XPath expression that is evaluated against the partially instantiated BSD.

Syntax

The `bs2:nOccurs` attribute is allowed for the particle components, *i.e.*:

— `xsd:element` and `xsd:group` components, other than immediate descendants of `xsd:schema`; and

— the model groups (`xsd:all`, `xsd:choice` and `xsd:sequence`).

Semantics

The `bs2:nOccurs` attribute specifies the number of occurrences of a particle. Its value is an XPath expression that shall be resolved against the partially instantiated BS Description with the context node set to the parent element of the particle being instantiated and cast into an integer.

NOTE Since `bs2:nOccurs` is ignored by XML Schema, the values of `xsd:minOccurs` and `xsd:maxOccurs` of the same component should be set so as to allow the full range of possible values. For example, writing `xsd:minOccurs="0"` and `xsd:maxOccurs="unbounded"` for a particle with a `bs2:nOccurs` attribute ensures that the generated BS Description will be schema-valid with respect to the number of occurrences of the particle.

Example

The following example is taken from a BS Schema describing an AVC bitstream. The element `slice_group_id` occurs multiple times, according to the value of `pic_size_in_map_units_minus1` plus 1.

```
<xsd:sequence bs2:if="slice_group_map_type = 6">
  <xsd:element name="pic_size_in_map_units_minus1" type="bs1:unsignedExpGolomb"/>
  <xsd:element name="slice_group_id" type="bs1:unsignedExpGolomb"
    maxOccurs="unbounded" bs2:nOccurs="avc:pic_size_in_map_units_minus1 + 1"/>
</xsd:sequence>
```

6.2.2 bs2:if attribute

Introduction

The `bs2:if` attributes specifies a test to determine whether a particle is to be instantiated. The test is described by an XPath expression that is evaluated against the partially instantiated BSD.

Syntax

The `bs2:if` attribute is allowed for the particle components, *i.e.*:

— `xsd:element` and `xsd:group` components, other than immediate descendants of `xsd:schema`; and

— the model groups (`xsd:all`, `xsd:choice` and `xsd:sequence`).

Semantics

The `bs2:if` attribute specifies a conditional expression on the occurrence of a particle component. Its value is an XPath expression that shall be resolved against the partially instantiated BS Description with the context node as the parent element of the particle being instantiated. If the equivalent Boolean value of the expression is `true`, then the particle shall be instantiated. Otherwise, the particle shall not be instantiated. The expression shall be reevaluated at each potential occurrence of the same particle.

NOTE Since `bs2:if` is ignored by XML Schema, the values of `xsd:minOccurs` and `xsd:maxOccurs` of the same component should be set so as to allow the full range of possible values. For example, writing `xsd:minOccurs="0"` and `xsd:maxOccurs="1"` for a particle with a `bs2:if` attribute ensures that the generated BS Description will be schema-valid with respect to the number of occurrences (0 or 1) of the particle.

Example

The sequence in the preceding example is instantiated only if the numeric value of `slice_group_map_type` is equal to 6. `slice_group_map_type` is an XPath expression that selects a child of the context node (which is the parent of the sequence).

6.2.3 bs2:ifNext, bs2:ifNextMask and bs2:ifNextSkip attributes

Introduction

The `bs2:ifNext`, `bs2:ifNextMask` and `bs2:ifNextSkip` attributes specify another test to determine whether a particle is to be instantiated. Whereas `bs2:if` specifies a test that relies on downstream parameters (anything that has already been parsed and instantiated in the BSD), the attributes here test upstream bytes. This part of the bitstream has not yet been parsed, and its content is therefore not yet described in the BSD.

Syntax

The `bs2:ifNext`, `bs2:ifNextMask` and `bs2:ifNextSkip` attributes are allowed for the particle components, global elements and groups, i.e.:

- `xsd:element` and `xsd:group` components; and
- the model groups (`xsd:all`, `xsd:choice` and `xsd:sequence`).

The `bs2:ifNext` attribute contains one or two strings conforming to the syntax specified in the *Schema for Schema for BSDL-2 Extensions*.

The `bs2:ifNext` value shall be interpreted as either one or two big-endian byte-strings separated by whitespace. The format of each string shall be one of:

- hexadecimal format, optionally preceded by "`0x`";
- binary format preceded by "`0b`"; or
- ASCII format, delimited at both ends by either single (‘) or double (“) quotes. Allowed characters are digits and upper and lower-case letters.

In the latter case, the character string is converted into a byte-string as follows: the left most character is the most significant byte and each ASCII character is decoded according to its hexadecimal code. For example, the character string "aA" is converted into the byte-string 0x6141.

The following constrains apply on the use of attributes:

- `bs2:ifNextMask` or `bs2:ifNextSkip` shall not be used without the `bs2:ifNext` attribute

— When `bs2:ifNext` contains two strings:

  — They shall represent the same number of bytes.

  — The byte-value of the first string shall be less than or equal to the byte-value of the second string.

— The `bs2:ifNextMask` and `bs2:ifNext` byte-strings shall represent the same number of bytes.

The `bs2:ifNextMask` attribute contains a hexadecimal value, which length in bytes must be equal to the length of the value(s) of `bs2:ifNext`.

In XML Schema, the `xsd:element` component refers to three cases:

— global element declaration: the `xsd:element` component has a `name` attribute and is immediately within `xsd:schema`

— local element declaration: the `xsd:element` component has a `name` attribute and is not immediately within `xsd:schema`

— element reference: the `xsd:element` has a `ref` attribute, which references an element declared globally

The `bs2:ifNext`, `bs2:ifNextMask` and `bs2:lookAhead` can be used for any of the three cases. However, they shall not be simultaneously used in an element reference and in the corresponding global element declaration.

The same rule applies to `xsd:group`.

Semantics

A BintoBSD parser shall instantiate a component having a `bs2:ifNext` attribute according to the following process:

1) Mark the current position in the bitstream.

2) If the component has a `bs2:ifNextSkip` attribute, consume the number of bytes indicated by the value of `bs2:ifNextSkip`.

3) Let *X* equal the value of the subsequent *n* bytes of the bitstream, where *n* is equal to the length in bytes of the `bs2:ifNext` string(s).

4) If `bs2:ifNextMask` is present, let *X* equal the bit-wise AND of *X* and the value of `bs2:ifNextMask`.

5) Reset the bitstream to the position marked at step (1).

6) The component shall be instantiated if *X* is greater than or equal to the value of the first string in the `bs2:ifNext` attribute, and either `bs2:ifNext` does not have a second string, or *X* is less than or equal to the value of the second string.

The additional run-time constraints applies:

— If a component with a `bs2:ifNext` attribute is encountered when the bitstream is not aligned with a byte boundary, the BintoBSD parser shall signal an error.

NOTE When the stream contains emulation prevention bytes, as indicated by the `bs2:removeEmPrevByte` attribute (see 6.2.5), BintoBSD shall test the bytes read from the raw bitstream, *i.e. before* removing the emulation prevention byte.

Example

The following example illustrates the use of the `bs2:nOccurs`, `bs2:if` and `bs2:ifNext` attributes. The use of `bs2:nOccurs` and `bs2:if` attributes is similar to `xsd:minOccurs` and `xsd:maxOccurs`, i.e., it may apply to `xsd:element` and `xsd:group` when they are not at the schema top level, `xsd:sequence`, `xsd:choice` and `xsd:all`. On the other hand, the `bs2:ifNext` may also be used in a global element declaration. In the example below, `bs2:ifNext` is used in local element declarations (`elt1` and `elt2`) and in global element declaration (see `elt3`), but cannot be used simultaneously in the global and local declaration of the same element.

In the example below, the behaviour of the BintoBSD parser is the following:

The BintoBSD starts by instantiating a `rootElt`. Since its content model is an `xsd:choice`, it then tries to instantiate `elt1`. Since the default value of `xsd:minOccurs` and `xsd:maxOccurs` is 1, it tries to instantiate one occurrence of it. For this, it first evaluates the boolean XPath expression given by the `bs2:if` attribute and compares the next bytes in the stream to the value FF01 given by the `bs2:ifNext` attribute. If both conditions are true, then an `elt1` is instantiated.

Otherwise, BintoBSD tries to instantiate `elt2` an indefinite number of occurrences since the `xsd:maxOccurs` value is set to `unbounded`. For this, it tests whether the next bytes in the stream correspond to the sequence FF02. If yes, then an `elt2` is parsed, and BintoBSD successively instantiates further occurrences of the `elt2` as long as the next bytes correspond to the FF02 sequence. In other terms, the test on the next bytes is performed for each occurrence of `elt2`.

Otherwise (neither `elt1` nor `elt2` could be instantiated), BintoBSD tries to parse `elt3`. The number of occurrences is given by the XPath expression of the `bs2:nOccurs` attribute which is evaluated in the instantiated DOM tree. Note that `xsd:maxOccurs` is set to `unbounded` so that the output description is still valid with respect to the schema. BintoBSD then instantiates `elt3` the corresponding number of times. Even though the number of occurrences is fixed by `bs2:nOccurs`, the test on the next bytes (that should correspond to the `bs2:ifNext` value FF03) is performed for each occurrence. If the test is false before the number of occurrences has been reached, then the bitstream does not comply with the schema and BintoBSD fails.

If a `bs2:if` or `bs2:ifNext` test is specified for a particle, then its number of occurrences should be set accordingly. In an `xsd:sequence`, the `xsd:minOccurs` should be set to zero in case the test is false. In an `xsd:choice` on the other hand, the optional occurrence of the particle is already implicit in the content model and does not need to be specified by `xsd:minOccurs`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            elementFormDefault="qualified"
            targetNamespace="http://www.example.com/myNamespace"
            xmlns:pref=" http://www.example.com/myNamespace "
            xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
            xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
            bs2:rootElement="pref:rootElt">

  <xsd:element name="rootElt">
    <xsd:complexType >
      <xsd:choice>
        <xsd:element name="elt1" minOccurs="0"
                     bs2:if="someBooleanXPathExpressionHere" bs2:ifNext="FF01">
          <!-- type definition here -->
        </xsd:element>
        <xsd:element ref="pref:elt2" minOccurs="0" maxOccurs="unbounded"
                     bs2:ifNext="FF02"/>
        <xsd:element ref="pref:elt3" minOccurs="0" maxOccurs="unbounded"
                                bs2:nOccurs="someIntegerXPathExpressionHere"/>
      </xsd:choice>
```

```
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="elt2">
    <!-- type definition here -->
  </xsd:element>

  <xsd:element name="elt3" bs2:ifNext="FF03">
    <!-- type definition here -->
  </xsd:element>

</xsd:schema>
```

6.2.4 bs2:rootElement attribute

Introduction

In XML Schema, several elements may be declared globally, and there is no way to specify which of these shall be the root element of the document. In this case, the BintoBSD parser cannot determine which element to instantiate as the document root without additional information. The `bs2:rootElement` attribute attached to an `xsd:schema` component provides this information, specifying which global element the BintoBSD parser is to instantiate as the root.

Syntax

The `bs2:rootElement` attribute is allowed for the `xsd:schema` component.

The attribute value shall be equal to the qualified name of an element declared in the same schema document.

Semantics

The value of the `bs2:rootElement` attribute shall indicate the root element for the BSD. That is, BintoBSD shall begin the instantiation of the BSD with the element indicated by `bs2:rootElement`.

Example

The following example illustrates the use of the `bs2:rootElement` attribute. XML Schema allows several global elements, i.e., declared at the root of the schema. While this has no impact on BSDL-1, this introduces an ambiguity for BSDL-2 since the BintoBSD Parser then does not know what is the root element to read from the bitstream. In case this indication is not provided by the application, the `bs2:rootElement` attribute should be added to `xsd:schema` to indicate the qualified name of the root element to be parsed. In the example below, the BintoBSD parser will start by parsing the `pref:elt1` element.

```
<?xml version="1.0" ?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  targetNamespace="http://www.example.com/myNamespace"
  xmlns:pref="http://www.example.com/myNamespace"
  xmlns:bs2="urn:mpeg:mpeg21:dia:2003:01-DIA-BSDL2-NS"
  bs2:rootElement="pref:elt1">

  <!-- Root element declaration -->
  <xsd:element name="elt1">
    <!-- and so on -->
  </xsd:element>

  <!-- Another global element declaration -->
  <xsd:element name="elt2">
```

```
    <!-- and so on -->
  </xsd:element>
</xsd:schema>
```

6.2.5 bs2:removeEmPrevByte attribute

Introduction

`bs2:removeEmPrevByte` specifies how to detect and remove any bytes that have been previously inserted to prevent the emulation of start codes. It is the equivalent attribute of `bs1:insertEmPrevByte` for the BintoBSD parser (see 5.3.7).

Syntax

The `bs2:removeEmPrevByte` attribute is allowed for the `xsd:schema` component.

It must contain an even number of hexadecimal strings.

Semantics

The `bs2:removeEmPrevByte` attribute specifies the removal of emulation prevention bytes.

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.

Example

In the example BS Schema shown in 4.4.2, the `bs2:removeEmPrevByte` specifies that whenever the "`000003`" byte-string is found while parsing the input bitstream, the last byte shall be identified as an emulation prevention byte and discarded.

6.2.6 bs2:defaultTreeInMemory, bs2:startContext, bs2:stopContext, bs2:partContext, bs2:redefineMarker attributes

Introduction

When parsing the input bitstream, the BintoBSD parser progressively instantiates the BSD. When XPath expressions are used (e.g. in `bs2:if` or `bs2:nOccurs` attributes), they are evaluated against this partially instantiated BSD. In practice, this requires the BintoBSD parser to store the instantiated BSD in memory during the parsing process, which may have a significant impact on its performance when the BSD grows very large.

However, in some cases, it is sufficient to store only selected parts of the instantiated BSD which are required by subsequent XPath expressions. In other cases, XPath expressions do not use location paths, (references to elements or attributes) and thus the BSD is not required within the parsing process itself.

The attributes specified in this subclause can be used to identify the parts of a BSD that must be stored in memory so they can be used in the evaluation of subsequent XPath expressions.

Syntax

The `bs2:startContext`, `bs2:partContext` and `bs2:redefineMarker` attributes are allowed for the `xsd:element` component.

The `bs2:stopContext` attribute is allowed for the `xsd:element` and `xsd:choice` components.

The `bs2:defaultTreeInMemory` attribute is allowed for the `xsd:schema` component.

Semantics

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.

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.

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.

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.

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.

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.

Example

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`.

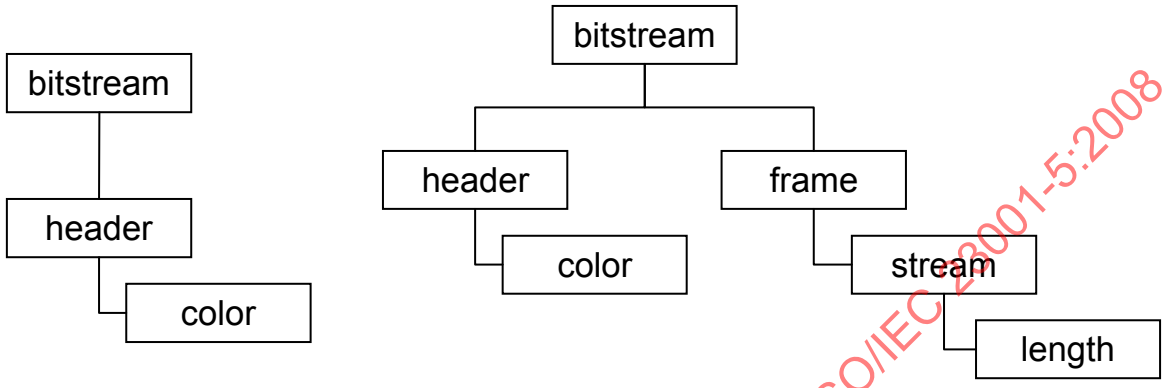
```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema targetNamespace="example"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:ex="example"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
  elementFormDefault="qualified" bs2:rootElement="ex:bitstream"
  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 5.

(a), the internal representation of the description is given after having parsed a frame; in Figure 5

(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.



a) After having parsed a frame. **b) Before a payload element is parsed.**

Figure 5 — Internal representation of the description by using context-related attributes.

6.2.7 bs2:layerLength attribute

Introduction

Some coding formats define a hierarchical structure of data segments, where each data segment contains other data segments and may provide information about its own length (in bytes). The `bs2:layerLength` attribute enables BintoBSD to parse the bitstream in a hierarchical and selective way using this length information.

For example, the ISO Base media file format (ISO/IEC 14496-12) defines a hierarchy of boxes where each box has a header containing a four-byte identifier of the box and a four-byte field indicating the box size. This size is sometimes required for parsing the content of the box, e.g. when the box contains an indefinite number of parameters (i.e. that should be parsed until the end of the box is reached).

The `bs2:layerLength` parameter allows the length (in bytes) of a complex type to be specified.

Syntax

The `bs2:layerLength` attribute is allowed for the `xsd:complexType` component.

Semantics

The `bs2:layerLength` attribute shall be interpreted as an XPath expression to be evaluated against the partially instantiated BSD, with the context node set to the element being instantiated. The total length in bytes of the descendants of the complex type shall be equal to the equivalent integer value of the result of the expression.

Example

The following example implements a box from the ISO File format (ISO/IEC 14496-12). The total size of the box is given by the size parameter. When the BintoBSD processes the `xsd:complexType` component, it

evaluates the XPath expression contained in `bs2:layerLength` and creates a new layer for the input bitstream with the indicated size. BintoBSD then parses the content of this complex type as if it was parsing a new bitstream with the given size. In other words, the `payload` element is read until the end of this bitstream layer.

```
<xs:element name="box">
  <xs:complexType bs2:assignPre="size 0 32" bs2:layerLength="$size">
    <xs:sequence>
      <xs:element name="size" type="xs:unsignedInt"/>
      <xs:element name="type" type="string4"/>
      <xs:element name="payload" type="bs1:byteRange"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

6.2.8 bs2:assignPre and bs2:assignPost attributes

Introduction

XPath expressions may contain literals (*i.e.* constants), location paths (*i.e.* reference to an element or attribute contained within the partially instantiated BSD) and variables (identified by a preceding '$' character). XPath does not provide a means for assigning a value to a variable, with must instead be done by the host language (in this case, BSDL).

The `bs2:assignPre` and `bs2:assignPost` attributes specify how to assign a value to a variable that can be used later in an XPath expression during the parsing process.

The `bs2:assignPre` attribute allows upstream data (i.e. that has not yet been parsed) to be assigned to a variable. This is particularly useful when a conditional statement in `bs2:if` tests the value of a field that has not yet been parsed and is thus not present in the partially instantiated BSD.

The `bs2:assignPost` attribute assigns the value of the most recently instantiated element to a variable.

In many cases, it is possible to avoid the use of location paths in XPath expressions by using XPath variables instead that have been assigned by `bs2:assignPre` and/or `bs2:assignPost`. As explained in 6.2.6, XPath location paths require storing at least parts of the BSD for the expression evaluation. The use of XPath variables therefore allows the memory consumption of BintoBSD to be reduced.

Syntax

The `bs2:assignPre` attribute is allowed for the following components:

- `xsd:complexType`
- `xsd:element` and `xsd:group` components not immediately within `xsd:schema`,
- `xsd:all`, `xsd:choice` and `xsd:sequence`.

It contains one or several triplets of string tokens where for each triplet the first token must be a non-colonized name and the following two unsigned integers.

The `bs2:assignPost` attribute is allowed for the `xsd:element` component.

Semantics

The `bs2:assignPre` attribute assigns the lexical value of one or several bitstream fields indicated by their offset and length to the indicated variables. It contains one or several triplets of string tokens. The first token indicates the qualified name of the variable. The second and third tokens respectively indicate the offset and

length in bits localizing the bitstream field. The offset is relative to the current cursor position. The bitstream field is parsed as an unsigned integer value.

The variable assignment is performed prior the processing of the particle or complex type and prior the evaluation of potential `bs2:if` or `bs2:nOccurs` attributes.

The `bs2:assignPost` attribute assigns the lexical value of the bitstream field that has been read to the indicated variable. The attribute value shall specify the qualified name of the variable that is to be assigned

The variable assignment is performed after the processing of the element.

Example

In the BS Schema shown in 6.2.7, the `bs2:assignPre` attribute specifies how to read a parameter from the bitstream and assign the value to the variable. When parsing the `box` element, the BintoBSD parser first reads the next 32 bits upstream as an unsigned integer (but without moving the parsing cursor) and assigns the value to the `size` variable. This variable is then used when evaluating the XPath expression contained in the `bs2:layerLength` attribute.

The following BS Schema snippet provides an alternative implementation of the schema shown in 6.2.1 using `bs2:assignPost`.

```
<xsd:sequence bs2:if="slice_group_map_type = 6">
  <xsd:element name="pic_size_in_map_units_minus1" type="bs1:unsignedExpGolomb"
               bs2:assignPost="n"/>
  <xsd:element name="slice_group_id" type="bs1:unsignedExpGolomb"
    maxOccurs="unbounded" bs2:nOccurs="$n + 1"/>
</xsd:sequence>
```

6.2.9 bs2:bsdlVersion attribute

Introduction

This attribute is the companion of `bs1:bsdlVersion` (5.3.8) and indicates the BSDL version with respect to BSDL-2 validity.

Syntax

The `bs2:bsdlVersion` attribute is allowed for the `xsd:schema` component.

Semantics

The `bs2:bsdlVersion` attribute indicates the version of BSDL that the BS Schema conforms to and hence the version of BintoBSD processor required. It is recommended to use this attribute in the `xsd:schema` component of the BS Schema to allow a BintoBSD processor check whether it may process the BS Schema to parse a bitstream.

The referred version is the value of the version attribute of the `xsd:schema` component of the Schema for Schema for BSDL-2 Extensions.

NOTE A BS Schema may require different versions of BSDL with respect to BSDtoBin ad BintoBSD. For example, a BS Schema may use BSDL-2 Extensions introduced in a recent version and require the corresponding version of BintoBSD whereas the BSDL-1 Extensions still refer to a previous version and only require the corresponding version of BSDtoBin. In other words, the versioning of a BS Schema may differ between BSDL-1 and BSDL-2. This is why two different attributes `bs1:bsdlVersion` and `bs2:bsdlVersion` are specified.

Example

The example shown in 4.4.2 indicates that the BS Schema conforms to this version of BSDL (ISO/IEC 23001-5).

6.2.10 bs2:requiredExtensions attribute

Introduction

The `bs2:requiredExtensions` attribute is the companion of `bs1:requiredExtensions`. It indicates what extension datatypes are required to run successfully the BintoBSD parser using the BS Schema.

NOTE Some extension datatypes may be required for BintoBSD, but not for BSDtoBin. This is why `bs1:requiredExtensions` and `bs2:requiredExtensions` are both required.

Syntax

The `bs2: requiredExtensions` attribute is allowed for the `xsd:schema` component.

Semantics

The `bs2:requiredExtensions` attribute indicates the user-defined datatypes required by this BS Schema. 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.

The `bs2:requiredExtensions` provides additional information in a BS Schema, but does not imply any normative behaviour of the BintoBSD parser. The behaviour of a BintoBSD parser is not specified when the implementation of the referenced datatypes is not available.

Example

See example in 5.3.5.

6.3 BSDL-2 facets

6.3.1 bs2:length facet

Introduction

The `bs2:length` facet has the same functionality as `xsd:length` (i.e. constraining the length of a datatype), but its value is resolved dynamically via an XPath expression.

Syntax

The `bs2:length` facet may be used to restrict any of the following datatypes:

- `xsd:string`, `xsd:normalizedString`,
- `xsd:hexBinary`, `xsd:base64Binary`,
- `bs1:byteRange`,
- `bs1:stringUTF16NT`, `bs1:stringUTF16BENT`, `bs1:stringUTF16LENT`, `bs1:stringUTF8NT`, `bs1:stringUTF16`, `bs1:stringUTF16BE`, `bs1:stringUTF16LE`, `bs1:stringUTF8`

or the `xsd:list` component.

The additional constraint applies on the use of `bs2:length`:

- A `bs2:length` facet shall not occur on a type that also declares a `xsd:length`, `bs2:startCode` or `bs2:endCode` facet on the same type or any of its base types.

Semantics

The `bs2:length` facet shall be interpreted as an XPath expression, resolved against the partially instantiated BSD, with the context node set to the element being instantiated. The equivalent integer value of the expression shall be used as the length of the data type.

The unit of length depends on the datatype:

— For string data types, the unit of length is characters;

— For the all other types (`xsd:hexBinary`, `xsd:base64Binary` and `bs1:byteRange`) the unit of length is bytes;

— For derivation by list (`xsd:list`), the unit of length is items.

NOTE While the number of characters and the number of bytes may be equal in some cases, this is not always so and depends on the character encoding of the datatype.

NOTE The semantics of `bs2:length` is similar to that of `xsd:length`. The difference is that the latter provides a static integer value, whereas the former provides an expression that is resolved to an integer dynamically.

Example

The following example is another way of implementing a file format box (see also 6.2.7). The `bs2:length` facet specifies the size of the `bs1:byteRange` datatype as the value of the `size` element.

```
<xs:element name="box">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="size" type="xs:unsignedInt"/>
      <xs:element name="type" type="string4"/>
      <xs:element name="payload" type="payloadType"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:simpleType name="payloadType">
  <xs:restriction base="bs1:byteRange">
    <xs:annotation>
      <xs:appinfo>
        <bs2:length value="../size - 8"/>
      </xs:appinfo>
    </xs:annotation>
  </xs:restriction>
</xs:simpleType>
```


6.3.2 bs2:bitLength facet

Introduction

Whereas the `bs2:length` facet constrains the size in bytes of datatypes such as `bs1:byteRange` or `xsd:hexBinary`, the `bs2:bitLength` facet constrains the size in bits of an unsigned integer. In other words, it can describe an unsigned integer that is encoded on a dynamic number of bits.

Syntax

The `bs2:bitLength` facet may be used to restrict any of the following datatypes:

— `xsd:unsignedLong`, `xsd:unsignedInt`, `xsd:unsignedShort` and `xsd:unsignedByte`

Semantics

The `bs2:bitLength` facet shall be interpreted as an XPath expression, resolved against the partially instantiated BSD, with the context node set to the element being instantiated. The equivalent integer value of the expression shall be used as the length in bits of the data type being restricted.

When instantiating an element that is restricted by `bs2:bitLength`, a BintoBSD parser shall add an `xsi:type` attribute to the element, with a value corresponding to one of `bs1:b1` ... `bs1:b32`, according to the number of bits assigned to the element. The behaviour of BintoBSD is not specified if the resulting value is greater than 32.

NOTE BSDtoBin is agnostic about the BSDL-2 extensions and therefore ignores the `bs2:bitLength` facet. This is why the number of encoding bits has to be specified with the `xsi:type` attribute and `bs1:b1` ... `bs1:b32`.

Example

The following example describes the `frame_num` field of an AVC bitstream. The length of this field is computed from `log2_max_frame_num_minus4`, which is in a prior part of the bitstream.

```
<xsd:simpleType name="frameNumType">
  <xsd:restriction base="xsd:unsignedLong">
    <xsd:annotation><xsd:appinfo>
      <bs2:bitLength value="$sliceSPS/avc:log2_max_frame_num_minus4 + 4"/>
    </xsd:appinfo></xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```

6.3.3 bs2:startCode and bs2:endCode facets

Introduction

In some coding formats, a data segment is read until a start code a found. A start code consists of one or more byte-sequences that indicate the start of a new data segment. For example in ISO/IEC 1449-10 (Advanced Video Coding), the content of a NALUnit is read until the start code `0x00000001` is seen (which indicates the start of the subsequent NAL Unit).

Similarly, an end code indicates the end of the current data segment; in this case, the byte sequence is included in the data segment being parsed.

Syntax

The `bs2:startCode` and `bs2:endCode` facets may be used to restrict any of the following datatypes:

— `xsd:hexBinary`, `xsd:base64Binary`

— `bs1:byteRange`

The value of either facet shall conform to the syntax specified in the *Schema for Schema for BSDL-2 Extensions*.

Additionally, the value of the `bs2:startCode` and `bs2:endCode` facets shall conform to the same syntactical constraints as the value of the `bs2:ifNext` attribute (see subclause 6.2.3).

NOTE A `startCode` or `endCode` may be specified using hexadecimal, binary or string format, and may contain a single value, or two values that are interpreted as a start code range (respectively end code) as for the `bs2:ifNext` attribute (see subclause 6.2.3).

Several `bs2:startCode` and `bs2:endCode` may be used to constrain a single datatype.

The additional constraint applies on the use of `bs2:startCode` and `bs2:endCode`:

— A `bs2:startCode` or `bs2:endCode` facet shall not occur on a type that also declares a `bs2:length or xsd:length` facet on the same type or any of its base types.

Semantics

The `bs2:startCode` facet shall be interpreted as a sequence of bytes (or a range of sequences) that denotes the start of the succeeding element.

The `bs2:endCode` facet shall be interpreted as a sequence of bytes (or a range of sequences) that denotes the end of the current element.

A BintoBSD parser shall append data to the current element until any of the declared `bs2:startCode` or `bs2:endCode` sequences are read from the bitstream. In the latter case, the sequence shall also be appended to the current element.

NOTE When the stream contains emulation prevention bytes, as indicated by the `bs2:removeEmPrevByte` attribute (see 6.2.5), BintoBSD shall test the bytes read from the raw bitstream, i.e. before *removing* the emulation prevention byte.

Example

In the following example, the Raw Byte Sequence Payload (RBSP) Type from AVC contains all of the bytes up to but excluding the start code of the following type, which has the value `00000001`.

```
<xsd:simpleType name="rbspType">
  <xsd:restriction base="bs1:byteRange">
    <xsd:annotation><xsd:appinfo>
      <bs2:startCode value="00000001"/>
    </xsd:appinfo></xsd:annotation>
  </xsd:restriction>
</xsd:simpleType>
```


6.3.4 bs2:escape and bs2:cdata facets

Introduction

As is the case for any XML document, a BSD has some constraints with respect to allowed characters. In particular, an XML element cannot contain characters such as '<' or '>' since these characters are interpreted as mark-up characters. This may be an issue when a bitstream contains an XML document. For example, the xml box of ISO/IEC 14496-12 (ISO Base Media File Format) contains an XML document. If BintoBSD parses this box as a string, this will appear as mark-up characters in the resulting BSD, which may not then be well-formed. To address this issue, the `bs2:cdata` indicates that the parsed string should be embedded as CDATA (character data) in the resulting BSD.

Another potential issue is the presence of control characters (such as carriage return or line feed) in a parsed string. These characters are platform specific and may be altered when the BSD is edited or transformed. To avoid this, the `bs2:escape` indicates that ISO control characters need to escaped by a character reference in the instantiated BSD. A character is considered to be an ISO control character if its code is in the range '`\u0000`' through '`\u001F`' or in the range '`\u007F`' through '`\u009F`'. For example, the sequence carriage return/line feed will be instantiated as "`
`". Note that, escaped or not, characters must be part of the allowed character set of XML. For example, the NULL character (`\u0000`) is forbidden in XML.

Syntax

The bs2:escape and bs2:cdata facets are applicable to the following datatypes:

— xsd:string, xsd:normalizedString

— bs1:stringUTF16NT, bs1:stringUTF16BENT, bs1:stringUTF16LENT, bs1:stringUTF8NT, bs1:stringUTF16, bs1:stringUTF16BE, bs1:stringUTF16LE, bs1:stringUTF8

Additionally, these facets shall not be used together for the same datatype.

Semantics

The bs2:escape facet indicates BintoBSD that control characters shall be escaped as XML character references in the output lexical value.

The bs2:cdata facet indicates BintoBSD that the output lexical value shall be marked as character data, i.e., encapsulated in "<![CDATA[" and "]]>".

NOTE These facets address two different constraints: bs2:escape escapes the control characters which encoding may be platform-specific. The bs2:cdata facet escapes strings that may contain XML mark-up characters.

Example

For example, with the bs2:escape facet, a line feed character read in the bitstream should be output as "
".

With the bs2:cdata facet, BintoBSD will instantiate the xml box of a file format as in the BSD snippet below. The XML document that was contained in the xml box is instantiated as CDATA string in the BSD, so that the BSD is well-formed.

```
<xml>
  <size>11960</size>
  <type>xml </type>
  <version>0</version>
  <flags>000000</flags>
  <boxData><![CDATA[<?xml version="1.0" encoding="UTF-8"?>
                    <!--and so on..-->]]</boxData>
</xml>
```

6.4 Other BSDL-2 schema components

6.4.1 bs2:ifUnion component

Introduction

In some coding formats, the datatype of an element depends on another parameter. For example in ISO/IEC 14496-2, the sprite-enable parameter is one or two bits long depending on the value of the video_object_layer_verid parameter.

XML Schema allows a choice between several datatypes to be specified via the xsd:union component. The bs2:ifUnion component provides conditional statements for each of the possible datatypes so that BintoBSD can determine which type should be instantiated.

Syntax

The bs2:ifUnion component shall be placed as the child of a xsd:annotation/xsd:appinfo combination under an xsd:union component.

The number of bs2:ifUnion schema components used within an xsd:union component shall be smaller than or equal to the number of member types of the xsd:union.

Semantics

The `value` attribute of the `bs2:ifUnion` component shall be evaluated as an XPath expression with the context node set to the element being instantiated.

A BintoBSD parser shall follow this process in deciding which type to instantiate from an `xsd:union` component:

1) Let the member types be numbered from 1, beginning with types specified by the `memberTypes` attribute, followed by anonymous type children, if any,

2) Let *i* = 1.

3) If *i* > the number of `bs2:ifUnion` descendents of the `xsd:union`, then the element type shall be the *i*th member type. End.

4) The `value` expression of the *i*th `bs2:ifUnion` shall be evaluated as specified. If the equivalent Boolean value is `true`, then the element type shall be the *i*th member type. End.

5) i = i + 1.

6) If *i* > the number of member types, then the element type shall be the *first* member type. End.

7) Goto (3).

Unless an anonymous type is instantiated, a BintoBSD parser shall place an `xsi:type` attribute on the instantiated element, with a value equal to the qualified name of the instantiated type.

Example

The Video Object Layer of ISO/IEC 14496-2 uses a parameter named `sprite_enable`, which is one or two bits, depending on the value of another parameter `video_object_layer_verid` (one bit long if `video_object_layer_verid = 1`, two bit long otherwise).

This may be implemented by a single element `sprite_enable` declared in the BS Schema with a type that is defined as a `xsd:union` between `bs1:b1` and `bs1:b2` (built-in types of 1 and 2 bit long unsigned integer), and the test is defined as the XPath expression `mp4:video_object_layer_verid = 1`.

Note that this example could also be implemented without `xsd:union`, but with a `bs2:bitLength` facet constraining `xsd:unsignedByte`.

```
<xsd:element name="sprite_enable">
  <xsd:simpleType>
    <xsd:union memberTypes="bs1:b1 bs1:b2">
      <xsd:annotation>
        <xsd:appinfo>
          <bs2:ifUnion value="mp4:video object layer verid = 1"/>
        </xsd:appinfo>
      </xsd:annotation>
    </xsd:union>
  </xsd:simpleType>
</xsd:element>
```

6.4.2 bs2:parameter component

Introduction

The `bs2:parameter` component assigns a constant value to an XPath variable that can be used later in XPath expressions (see also 6.2.8).

This element is similar to the parameter element in XSLT.

Syntax

The `bs2:parameter` component shall be placed as the child of a `xsd:annotation/xsd:appinfo` combination under an `xsd:schema` component.

Semantics

The `bs2:parameter` component declares an XPath variable and assigns a constant value to it. This value may be overridden at run-time by a value provided by some other means (e.g., by command line). The name attribute indicates the qualified name of the variable.

Example

The following BS Schema snippet shows how to assign a constant value (here, the string "0") to the XPath variable `$myParam`. This variable may be referenced in subsequent XPath expressions. Note that as with other XPath expressions, the type of the variable is dynamic: it is dependent on the context in which it is used.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS">

  <xs:annotation>
    <xs:appinfo>
      <bs2:parameter name="myParam">0</bs2:parameter>
    </xs:appinfo>
  </xs:annotation>

</xs:schema>
```


6.4.3 bs2:xpathScript component

Introduction

XPath allows for the use of functions that are not defined by its core library, but (as with variables) provides no syntax for their declaration. A BS Schema may therefore use such functions in XPath expressions, but without a means for function declaration, such a BS Schema is no longer interoperable.

If a BintoBSD parser supports ECMAScript, it is possible to provide an implementation of extension functions as a script included in the BS Schema itself, preserving full interoperability.

The `bs2:xpathScript` component acts as a wrapper for embedding the script implementation of the extension function.

NOTE BSDL does not mandate the support for script implementation for XPath extension functions, but Annex A specifies an optional mechanism for ECMAScript implementation.

Syntax

The `bs2:xpathScript` component shall be placed as the child of a `xsd:annotation/xsd:appinfo` combination under the `xsd:schema` component.

Semantics

The `bs2:xpathScript` element provides a wrapper for scripts associated to extension XPath functions for the BintoBSD parsing process.

NOTE Other than the optional Annex A, the use of `bs1:script` elements is not normatively defined.

Extension functions declared within an `bs2:xpathScript` component shall inherit the target namespace of the schema in which the `bs2:xpathScript` component is instantiated.

Extension function references within an XPath expression must have a prefix in order to distinguish them from the XPath core functions library. The prefix is resolved against the local namespace binding context in the BS Schema where this expression is used.

Example

The example below shows how to specify an XPath extension function in a BS Schema using the `bs2:xpathScript` component. Note that the `bs2:log2()` function is actually natively defined by BSDL-2 (see 6.1.5), and would not need to be defined as a script.

The `log2()` function is defined in the `bs2:xpathScript` component and is referenced in the XPath expression used in the `bs2:bitLength` facet via its qualified name `p:log2()`.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" targetNamespace="urn:example:myNS"
  xmlns:p="urn:example:myNS" xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS">

  <xs:annotation>
    <xs:appinfo>
      <bs2:xpathScript><![CDATA[
          function log2(operand) {
            return Math.log(operand)/Math.log(2);
          }
        ]]></bs2:xpathScript>
    </xs:appinfo>
  </xs:annotation>

  <xs:simpleType name="myType">
    <xs:restriction base="xs:unsignedInt">
      <xs:annotation>
        <xs:appinfo>
          <bs2:bitLength value="ceiling(p:log2($someVar))"/>
        </xs:appinfo>
      </xs:annotation>
    </xs:restriction>
  </xs:simpleType>
</xs:schema>
```


6.4.4 bs2:variable component (optional feature)

Introduction

This component allows a BS Schema to declare variables that may be subsequently referenced from within XPath expressions. In particular, 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 This component is an optional feature, i.e. a BintoBSD parser is not required to implement it. However, when it does, it shall conform to this subclause.

Syntax

The `bs2:variable` component shall be placed as the child of a `xsd:annotation/xsd:appinfo` combination under an `xsd:element` component.

Semantics

The value of a variable shall be one of the following:

— If the `position` attribute is present, a Node-set;

— If the `value` attribute is present, the result of evaluating the `value` expression; or

— The current instantiation of the element node on which the variable is declared, otherwise.

The `name` attribute indicates the qualified name of the variable.

The `value` attribute shall contain an XPath expression which shall be evaluated against the partially instantiated BSD, relative to the current instantiation of the enclosing element. The result of the XPath expression shall be stored as the partial-value of the variable being declared.

If the `value` attribute is not present, then the partial-value of the variable shall be the element node being instantiated.

The `position` attribute shall contain an XPath expression which shall be evaluated against the partially instantiated BSD, relative to the current instantiation of the enclosing element. The XPath expression shall resolve to a strictly positive integer.

NOTE In other words, the first position within a variable node-set is 1.

When the position attribute is present, the value of the variable shall be a node-set. The partial-value indicated by the `value` attribute shall be stored in the position indicated by the result of the `position` expression.

If the partial-value is a node-set with more than one item, then the items in the partial-value are stored within the `variable` node-set in consecutively increasing positions, beginning with the result of the `position` expression.

If the `position` attribute is not present, then the value of the variable shall be the partial-value indicated by the `value` expression.

Example

The example below shows how to declare an XPath variable in a BS Schema using the `bs2:variable` component. This variable is then referenced in an XPath expression.

```
<element name="elementA" type="elementAType">
  <annotation><appinfo>
    <bs2:variable name="aVariable" position="2" value="./text() * 2"/>
  </appinfo></annotation>
</element>

<element name="elementB">
  <complexType>
    <sequence>
      <element name="elementC" type="elementCType" bs2:if="$aVariable[2] = 3"/>
    </sequence>
  </complexType>
</element>
```

6.5 Schema for Schema for BSDL-2 Extensions

```
<?xml version="1.0"?>
<!-- Bitstream Syntax Description Language ISO/IEC 23001-5 -->
<!-- Schema for Schema for BSDL-2 extensions -->
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:bs2="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
        targetNamespace="urn:mpeg:mpeg21:2003:01-DIA-BSDL2-NS"
        version="ISO/IEC 23001-5" id="MPEG-B-BSDL-2.xsd">

  <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            -->
  <!-- ################################### -->
  <attribute name="nOccurs" type="bs2:xPathExpression"/>
  <attribute name="if" type="bs2:xPathExpression"/>
  <attribute name="ifNext" type="bs2:oneOrTwoValues"/>
  <attribute name="ifNextMask" type="hexBinary"/>
  <attribute name="ifNextSkip" type="unsignedShort"/>
  <attribute name="rootElement" type="QName"/>
  <attribute name="removeEmPrevByte" type="bs2:hexBinaryStrings"/>
  <attribute name="defaultTreeInMemory" type="boolean" default="true"/>
  <attribute name="startContext" type="bs2:xPathExpression"/>
  <attribute name="partContext" type="boolean" default="false"/>
  <attribute name="redefineMarker" type="bs2:xPathExpression"/>
  <attribute name="stopContext" type="bs2:xPathExpression"/>
  <attribute name="layerLength" type="bs2:xPathExpression"/>
  <attribute name="assignPre" type="NMTOKENS"/>
  <attribute name="assignPost" type="QName"/>
  <attribute name="bsdlVersion" type="string"/>
  <attribute name="requiredExtensions" type="bs2:anyURIList"/>

  <!-- ################################### -->
  <!--            BSDL Facets              -->
  <!-- ################################### -->
  <element name="length" type="bs2:numFacet"/>
  <element name="bitLength" type="bs2:numFacet"/>
  <element name="startCode" type="bs2:startCodeFacet"/>
  <element name="endCode" type="bs2:startCodeFacet"/>
  <element name="escape" type="bs2:booleanFacet"/>
  <element name="cdata" type="bs2:booleanFacet"/>

  <!-- ################################### -->
  <!--       BSDL Schema Components        -->
  <!-- ################################### -->
  <element name="ifUnion">
    <complexType>
      <complexContent>
```