
**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)*

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2008

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

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

Published in Switzerland

Contents

Page

Foreword.....	v
Introduction	vi
1 Scope	1
2 Normative References	1
3 Terms, definitions and abbreviated terms	1
3.1 Terms and definitions.....	1
3.2 Abbreviated terms	2
4 Schema documents	2
4.1 General.....	2
4.2 Use of prefixes in this specification	2
4.3 XML, Schema, XML Schema overview.....	3
4.4 BS Description, BS Schema, BSDL	4
4.4.1 BS Description	4
4.4.2 BS Schema	5
4.4.3 BSDL	7
4.4.4 BSDL parsers: BSDtoBin and BintobSD	8
4.4.5 Advanced use of BS Descriptions	10
4.5 Examples of applications for BSDL	11
4.6 Relation with ISO/IEC 21000-7	11
5 BSDL-1 and BSDtoBin.....	12
5.1 Constraints on BS Descriptions.....	12
5.2 Datatypes in BSDL.....	13
5.2.1 Overview	13
5.2.2 Extension datatypes	13
5.2.3 Facets.....	13
5.2.4 Simple type derivation.....	14
5.2.5 XML Schema built-in datatypes supported by BSDL	14
5.2.6 BSDL built-in datatypes	15
5.3 BSDL-1 attributes.....	18
5.3.1 Overview	18
5.3.2 bs1:ignore attribute	19
5.3.3 bs1:bitstreamURI attribute.....	20
5.3.4 bs1:addressUnit attribute	21
5.3.5 bs1:codec attribute.....	22
5.3.6 bs1:requiredExtensions attribute	23
5.3.7 bs1:insertEmPrevByte attribute	24
5.3.8 bs1:bsdlVersion attribute	25
5.4 bs1:script element	26
5.5 Schema for BSDL-1 Extensions	26
5.6 Bitstream generation with BSDtoBin.....	32
5.6.1 Processing model.....	32
6 BSDL-2 and BintobSD.....	33
6.1 Overview	33
6.1.1 Introduction	33
6.1.2 Annotation mechanisms of XML Schema	33
6.1.3 Facets.....	34
6.1.4 XPath expressions.....	34
6.1.5 bs2:log2() XPath function	35
6.1.6 XPath variables assignment	35

6.2	BSDL-2 attributes	36
6.2.1	bs2:nOccurs	36
6.2.2	bs2:if attribute	36
6.2.3	bs2:ifNext, bs2:ifNextMask and bs2:ifNextSkip attributes	37
6.2.4	bs2:rootElement attribute	40
6.2.5	bs2:removeEmPrevByte attribute	41
6.2.6	bs2:defaultTreeInMemory, bs2:startContext, bs2:stopContext, bs2:partContext, bs2:redefineMarker attributes	41
6.2.7	bs2:layerLength attribute	44
6.2.8	bs2:assignPre and bs2:assignPost attributes	45
6.2.9	bs2:bsdlVersion attribute	46
6.2.10	bs2:requiredExtensions attribute	47
6.3	BSDL-2 facets	47
6.3.1	bs2:length facet	47
6.3.2	bs2:bitLength facet	48
6.3.3	bs2:startCode and bs2:endCode facets	49
6.3.4	bs2:escape and bs2:cdata facets	50
6.4	Other BSDL-2 schema components	51
6.4.1	bs2:ifUnion component	51
6.4.2	bs2:parameter component	53
6.4.3	bs2:xpathScript component	53
6.4.4	bs2:variable component (optional feature)	54
6.5	Schema for Schema for BSDL-2 Extensions	56
6.6	BintoBSD Parser	59
6.6.1	Processing model	59
6.6.2	BSDL-2 validity of BS Schemas	61
Annex A (normative)	ECMAScript implementation of extension datatypes	62
A.1	Overview	62
A.2	BSDL-defined ECMAScript functions	63
Annex B (informative)	Non-normative feature for BSDL	65
B.1	Introduction	65
B.2	Non-normative BSDL-1 attribute	65
B.2.1	bs1i:implementation attribute	65
B.3	Schema for non-normative BSDL-1 Extensions	66
B.4	Non-normative BSDL-2 attributes	66
B.4.1	bs2i:debugMsg, bs2i:debugBool, bs2i:debugInt, bs2i:debugNumber and bs2i:debugStr attributes	66
B.5	Schema for non-normative BSDL-2 Extensions	67
Annex C (informative)	Parsing process for Exp-Golomb codes	68
Bibliography		70

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 `e10` element shall contain an element named `e11` and an element named `e12`
- the content of `e11` shall be an integer
- the content of `e12` shall be a hexadecimal value

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="e10">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="e11" type="xs:int"/>
        <xs:element name="e12" 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"?>
<e10 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="example1.xsd">
  <e11>1</e11>
  <e12>FF</e12>
</e10>
```

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 `e10` element shall contain an element named `e11` and an element named `e12`)
- on the datatypes used (the content of `e11` 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:mpeg21:2003:01-DIA-BSDL1-NS"
  bs1:bitstreamURI="football_40_frames.avc"
  xmlns:bs1="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS"
  xsi:schemaLocation="urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS urn:mpeg:mpeg21:2003:01-DIA-BSDL1-NS.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 BS Description is an XML Document. Additionally, BSDL specifies semantics for parsing a bitstream and generating the corresponding BSD and vice versa.

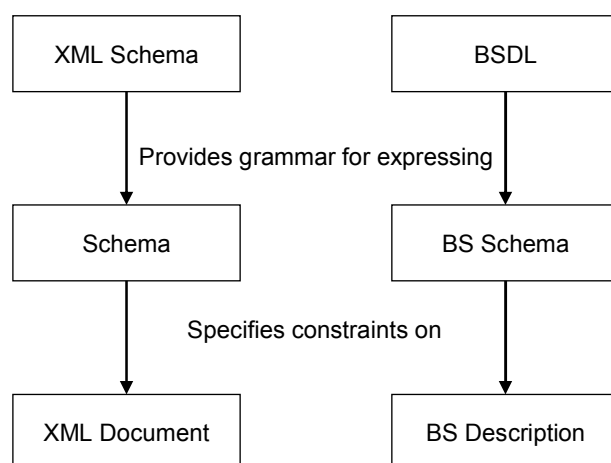


Figure 1 — Relation between XML Schema and BSDL

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.

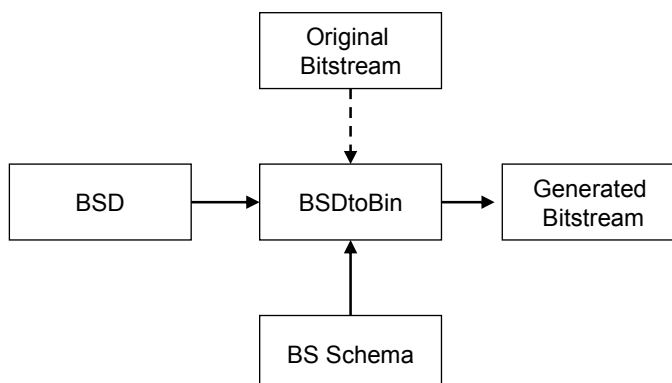


Figure 2 — BSDtoBin parser

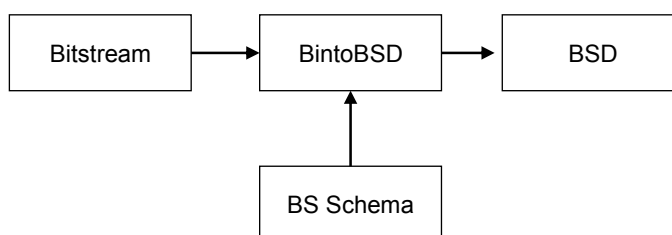


Figure 3 — BintoBSD parser

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 below.

```
<?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
<code>xsd:string</code>	Indefinite		<code>xsd:length</code> <code>bs2:length</code> <code>bs2:cdata</code> <code>bs2:escape</code>
<code>xsd:normalizedString</code>	Indefinite		<code>xsd:length</code> <code>bs2:length</code> <code>bs2:cdata</code> <code>bs2:escape</code>
<code>xsd:float</code>	4 bytes		
<code>xsd:double</code>	8 bytes		
<code>xsd:hexBinary</code>	Indefinite		<code>xsd:length</code> <code>bs2:length</code> <code>bs2:startCode</code> <code>bs2:endCode</code>
<code>xsd:base64Binary</code>	Indefinite		<code>xsd:length</code> <code>bs2:length</code> <code>bs2:startCode</code> <code>bs2:endCode</code>
<code>xsd:long</code>	8 bytes		
<code>xsd:int</code>	4 bytes		
<code>xsd:short</code>	2 bytes		
<code>xsd:byte</code>	1 byte		
<code>xsd:unsignedLong</code>	8 bytes	<code>xsd:maxExclusive</code>	<code>xsd:maxExclusive</code> <code>bs2:bitLength</code>
<code>xsd:unsignedInt</code>	4 bytes	<code>xsd:maxExclusive</code>	<code>xsd:maxExclusive</code>

			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 `e11` element and assigns a property named *addressUnit* to the `e11` element, which is inherited by the `e12` element. According to its enriched PSVI, the *addressUnit* property of `e12` is therefore set to "bit", and BSDtoBin will process `e12` accordingly. Note that from a strict XML point of view, the `bs1:addressUnit` attribute only adds information to `e11`, and is unknown to `e12`.

```
<!-- Description example -->
<e11 bs1:addressUnit="bit">
  <e12>2 5</e12>
</e11>
```

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