INTERNATIONAL ORGANISATION FOR STANDARDISATION
ORGANISATION INTERNATIONALE DE NORMALISATION

ISO/IEC JTC1/SC29/WG11

CODING OF MOVING PICTURES AND AUDIO

ISO/IEC JTC1/SC29/WG11

MPEG2007/N9033

April 2007, San Jose, CA, USA

Title: **WD3.0 of ISO/IEC 14496-20 2nd Edition (1st Ed. + Cor + Amd.)**

Status: **Approved, to be published**

Note: apart from the beginning and URLs, text in blue comes from COR1, and text in purple comes from the AMD1.

Document type: International Standard
Document subtype:
Document stage: (50) Approval
Document language: E

STD Version 2.1c2

ISO/IEC JTC 1/SC 29

Date: 2006-01-23

ISO/IEC FDIS 14496-20:2006(E)

ISO/IEC JTC 1/SC 29/WG 11

Secretariat:

Information technology — Coding of audio-visual objects — Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)

Élément introductif — Élément central — Partie 20: Titre de la partie

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 14496-20 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 14496 consists of the following parts, under the general title *Information technology — Coding of audio-visual objects*:

— *Part 1: Systems*
— *Part 2: Visual*
— *Part 3: Audio*
— *Part 4: Conformance testing*
— *Part 5: Reference software*
— *Part 6: Delivery Multimedia Integration Framework (DMIF)*
— *Part 7: Optimized software for MPEG-4 tools*
— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*
— *Part 9: Reference hardware description*
— *Part 10: Advanced Video Coding (AVC)*
— *Part 11: Scene description and application engine*
— *Part 12: ISO media file format*
— *Part 13: IPMP extensions*
— *Part 14: MP4 file format*
— *Part 15: AVC file format*
— *Part 16: Animation Framework eXtension (AFX)*
— *Part 17: Streaming text format*
— *Part 18: Font compression and streaming*
— *Part 19: Synthesised texture stream*

— *Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)*

— *Part 21: MPEG-J GFX*

Introduction

ISO/IEC 14496-20 specifies syntax and semantics for:
- The Lightweight Application Scene Representation (LASeR), specified in clause 6, which is a binary format for encoding 2D scenes and updates of scenes. The binary format and the scene representation (based on SVG Tiny), are both designed to be suitable for lightweight embedded devices such as mobile phones.
- A Simple Aggregation Format (SAF), specified in clause 7, to efficiently and easily transport LASeR data together with audio and/or video content over various delivery channels. This multiplexing scheme is designed to be simple to implement and to allow efficient demultiplexing on low-end devices.

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this document may involve the use of a patent.

The ISO and IEC take no position concerning the evidence, validity and scope of this patent right.

The holder of this patent right has assured the ISO and IEC that he is willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statement of the holder of this patent right is registered with the ISO and IEC. Information may be obtained from the companies listed in Annex A.

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

Information technology — Coding of audio-visual objects — Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF)

1. Scope

This International Standard defines a scene description format (LASeR) and an aggregation format (SAF) respectively suitable for representing and delivering rich-media services to resource-constrained devices such as mobile phones.
LASeR aims at fulfilling all the requirements of rich-media services at the scene description level. LASeR supports:

- an optimized set of objects inherited from SVG to describe rich-media scenes,
- a small set of key compatible extensions over SVG,
- the ability to encode and transmit a LASeR stream and then reconstruct SVG content,
- dynamic updating of the scene to achieve a reactive, smooth and continuous service,
- simple yet efficient compression to improve delivery and parsing times, as well as storage size, one of the design goals being to allow both for a direct implementation of the SDL as documented, as well as for a decoder compliant with ISO/IEC 23001-1 to decode the LASeR bitstream.
- an efficient interface with audio and visual streams with frame-accurate synchronization,
- use of any font format, including the OpenType industry standard,
- and easy conversion from other popular rich-media formats in order to leverage existing content and developer communities.

Technology selection criteria for LASeR included compression efficiency, but also code and memory footprint and performance. Other aims included: scalability, adaptability to the user context, extensibility of the format, ability to define small profiles, feasibility of a J2ME implementation, error resilience and safety of implementations.

SAF aims at fulfilling all the requirements of rich-media services at the interface between media/scene description and existing transport protocols:

- simple aggregation of any type of stream,
- signaling of MPEG and non-MPEG streams,
- optimized packet headers for bandwidth-limited networks,
- easy mapping to popular streaming formats,
- cache management capability,
- and extensibility.

SAF has been designed to complement LASeR for simple, interactive services, bringing:

- efficient and dynamic packaging to cope with high latency networks,
- media interleaving,
- and synchronization support with a very low overhead.

The present specification defines the usage of SAF for LASeR content. However, LASeR can be used independently from SAF.

2. Normative References

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 13818-1, *Information technology — Generic coding of moving pictures and associated audio information: Systems*

ISO/IEC 14496-1, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-12, *Information technology — Coding of audio-visual objects — Part 12: ISO Base Media File Format*

ISO/IEC 14496-18, *Information technology — Coding of audio-visual objects — Part 18: Font compression and streaming*

ISO/IEC 9899:1990 Information technology - Programming Language C

ISO/IEC 14882: Programming Language C++

ISO/IEC 16262:2002 Information technology — ECMAScript language specification

IETF BCP 13, Media Type Specifications and Registration Procedures, http://www.ietf.org/rfc/rfc4288.txt

RFC 3023, XML Media Types, M. Murata, S. St.Laurent, D. Kohn, January 2001, http://www.ietf.org/rfc/rfc3023.txt

RFC 3986, Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, January 2005, http://www.ietf.org/rfc/rfc3986.txt

RFC 2045, MIME formats and encodings, http://www.ietf.org/rfc/rfc2045.txt

RFC 2326, Real Time Streaming Protocol, http://www.ietf.org/rfc/rfc2326.txt

RFC 2965, HTTP State Management Mechanism, Kristol and Montulli, http://www.ietf.org/rfc/rfc2965.txt

W3C SVG11, Scalable Vector Graphics (SVG) 1.1 Specification [Recommendation], http://www.w3.org/TR/2003/REC-SVG11-20030114/,

W3C SMIL2, Synchronized Multimedia Integration Language (SMIL 2.0) - [Second Edition], J. Ayars, D. Bulterman *et. al.*, 07 January 2005. http://www.w3.org/TR/2005/REC-SMIL2-20050107/

W3C CSS, Cascading Style Sheets, level 2 [Recommendation], http://www.w3.org/TR/1998/REC-CSS2-19980512/

W3C DOM, Document *Object* Model Level 2 Events Specification, Version 1.0, W3C Recommendation *13 November, 2000.* http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113

W3C XML Events, an Events Syntax for XML, W3C Recommandation 14 October 2003. http://www.w3.org/TR/2003/REC-xml-events-20031014

W3C xml:id Version 1.0, W3C Recommendation 9 September 2005, http://www.w3.org/TR/2005/PR-xml-id-20050712/

W3C Xlink, XML Linking Language, W3C Recommendation, 27 June 2001. http://www.w3.org/TR/2001/REC-xlink-20010627/

3. Terms, Definitions and Abbreviations

3.1. Terms and definitions

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

3.1.1. Access unit

An individually accessible portion of data within a media stream. An access unit is the smallest data entity to which timing information can be attributed.

3.1.2. Media time line

the axis on which times are expressed within the transport or system carrying a LASeR or other stream.

3.1.3. Normal play time

indicates the stream absolute position relative to the beginning of the presentation [RFC 2326].

3.1.4. Packet

The smallest data entity managed by SAF consisting of a header and a payload.

3.1.5. Scene segment

a set of access units of a LASeR stream, where only the first access unit contains a LASeRHeader.

3.1.6. Scene time line

the axis on which times are expressed within the SVG/LASeR scene, e.g. begin and end.

3.1.7. Waiting tree

The waiting tree is a separate tree defined in addition to the scene tree. The compositor and renderer have no knowledge of the waiting tree, thus objects in the waiting tree are neither composited nor rendered.

3.2. Abbreviations

List of abbreviated terms.

CSS: Cascading Style Sheets, a W3C standard.

SMIL: Synchronized Multimedia Integration Language, a W3C standard.

SVG: Scalable Vector Graphics, a W3C standard.

4. Document Conventions

This document uses the following styling conventions for various types of information.

Any name of element, attribute, descriptor or command defined in this specification is styled in bold italic, such as ***Add***. Any name of element, attribute, descriptor or command defined in another specification is prefixed with the name of that specification, such as ***SVG animate*** or ***SMIL video***.

XML examples use the following style:

```
<?xml version="1.0" encoding="UTF-8"?>
<svg width="480" height="360" viewBox="0 0 480 360"
     version="1.1" baseProfile="tiny">
        <defs>    …
```

SDL descriptions of binary syntax use the following style:

```
Insert extends LASeRUpdate {
  const bit(UpdateBits) InsertCode;
  uint(idBits) ref;
```

The following is the style used for ECMA Script:

```
function Insert(parentId, field, value) {…
```

5. Architecture

5.1. Overview

LaSER is defined in terms of abstract access units, which may be adapted for transmission over a variety of protocols. LaSER streams may be packaged with some or all of their related media into files of the ISO base media file format family (e.g. MP4) and delivered over reliable protocols. There is also a simple aggregation format (SAF), which aggregates a LaSER stream with some or all of its associated media into stream order. SAF may be delivered over reliable or unreliable protocols. Finally, LASeR streams could be adapted to other delivery protocols such as RTP [RFC 2326] or MPEG-2 transport [ISO/IEC 13818-1]; however, the definitions of these mappings is outside the scope of this specification.

Figure 1 presents the LASeR and SAF architecture.

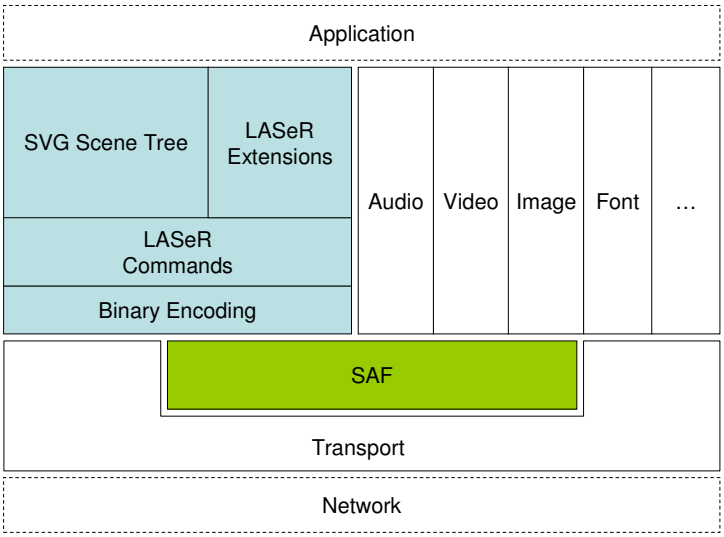


Figure 1 — Architecture of LASeR and SAF

5.2. LASeR Systems Decoder Model

5.2.1. Introduction

The purpose of the LASeR Systems decoder model is to provide an abstract view of the behaviour of the terminal complying with ISO/IEC 14496-20. It may be used by the sender to predict how the receiving terminal will behave in terms of buffer management and synchronization when decoding data received in the form of elementary streams. The LASeR systems decoder model includes a timing model and a buffer model. The LASeR systems decoder model specifies:

1. the conceptual interface for accessing data streams (Delivery Layer),

2. decoding buffers for coded data for each elementary stream,

3. the behavior of elementary stream decoders,

4. composition memory for decoded data from each decoder, and

5. the output behavior of composition memory towards the compositor.

These elements are depicted in [ref].

Each elementary stream is attached to one single decoding buffer.

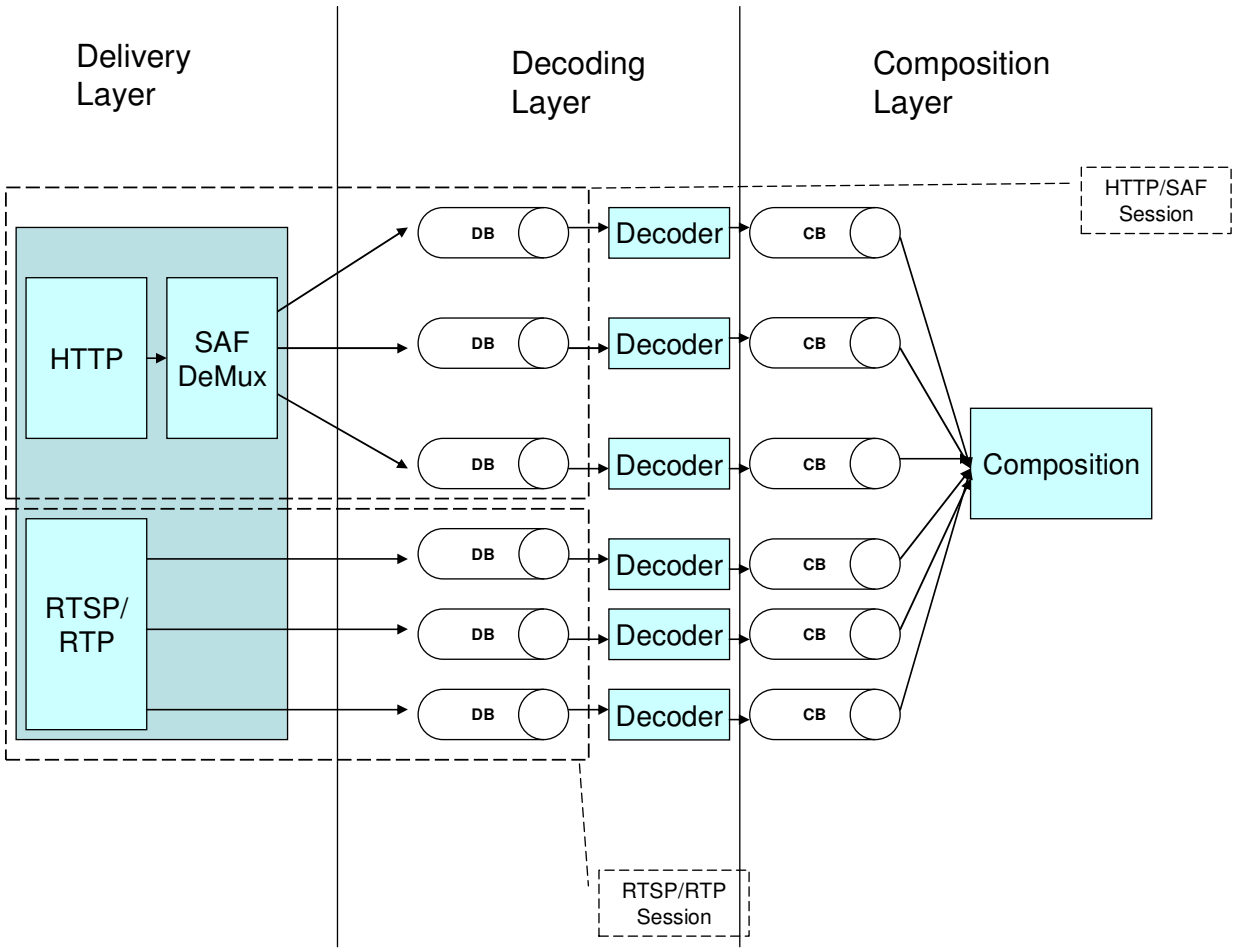


Figure 2 LASeR Systems Decoder Model

The definition in ISO/IEC 14496-1 of Access Unit, Decoding Buffer(DB), elementary stream (ES), Decoder (CU) and Composition Unit apply.

5.2.2. Decoder Model

The decoder model as specified in 14496-1 Section 7.4.1 applies

5.2.2.1. Decoding Buffer

The needed decoding buffer size is known by the sending terminal and conveyed to the receiving terminal as specified in 7.6. The size of the decoding buffer is measured in bytes. The decoding buffer is filled at the rate given by the maximum bit rate for this elementary stream while data is available and with a zero rate

otherwise. The maximum bit rate is conveyed by the sending terminal as a part of the decoder configuration information during the set up phase for each elementary stream (see 7.6).

5.2.2.2. Decoder model with grouped streams

This decoder model may be enhanced when used for group of multiple elementary streams.

In such case, only one composition buffer for the group of streams is used for composition.

When such streams are grouped, and when the setup of multiple decoding chains are available, it is possible, although not mandatory, not to decode all streams at a time.

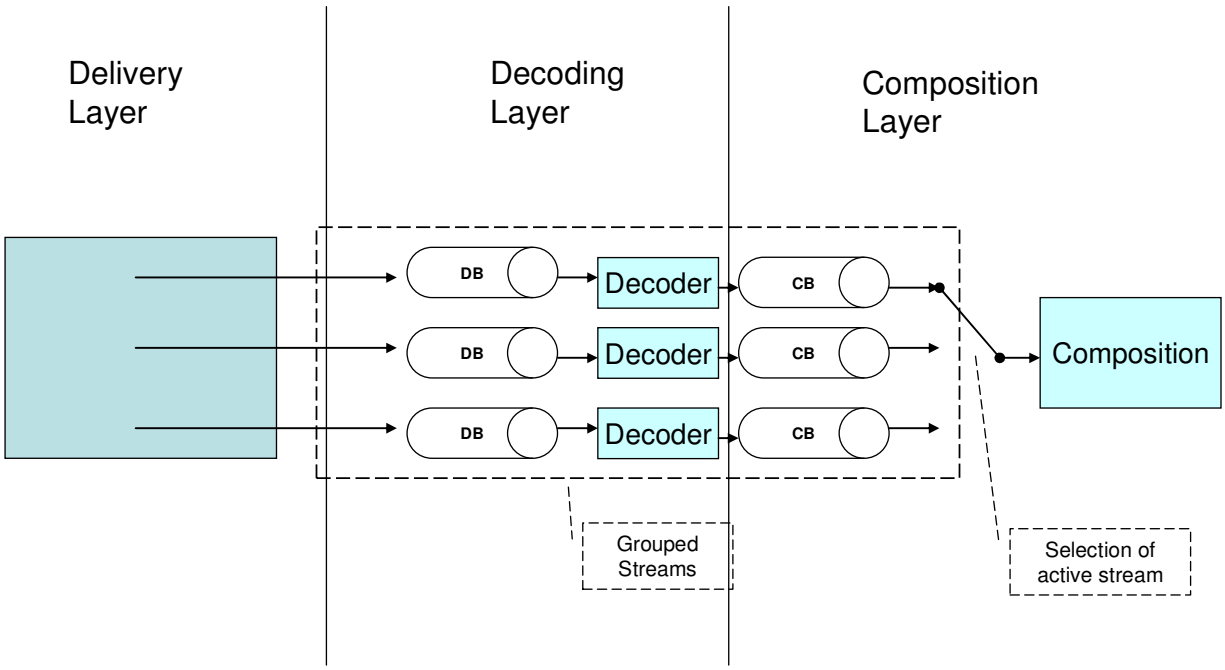


Figure 3: Stream grouping with specified System Decoder Model (multiple decoders)

It is indeed expected that multiple decoders may not be available in lightweight terminals or that some delivery scenarios do not allow for having all streams available at the same time (e.g. in broadcast scenarios, the delivery layer could only tune in to one of the streams). The usage of new information about this grouping enables a smart usage of buffers and decoders.

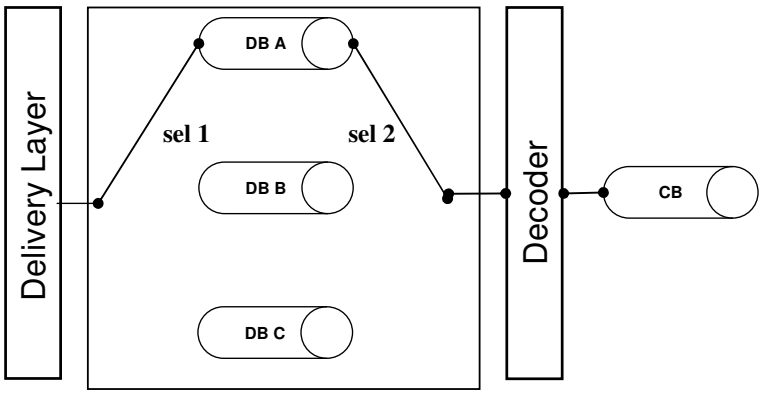


Figure 4: Broadcast example of streams grouping, showing a potential optimization using a single decoder.

When only a subset of the group of streams can be accessed at a time (e.g. broadcast scenario depicted above), the selection of the active stream corresponds to a request for the corresponding streams. Nevertheless, the buffer model for stream grouping does not assume immediate reception of data after such request and therefore the active decoding buffer may continue to be used by the decoder up to the moment at which data is available for the newly available stream. In this case the decoding buffer associated to the newly connected stream can be associated with the decoder. At this point the terminal may discard any remaining access units in the previous decoding buffer.

6. Scene Representation

6.1. Overview

In this document, a multimedia presentation is a collection of a scene description and media (zero, one or more). A media is an individual audiovisual content of the following type: image (still picture), video (moving pictures), audio and by extension, font data. A scene description is constituted of text, graphics, animation, interactivity and spatial, audio and temporal layout.

A scene description specifies four aspects of a presentation:

- how the scene elements (media or graphics) are organised spatially, e.g. the spatial layout of the visual elements;
- how the scene elements (media or graphics) are organised temporally, i.e. if and how they are synchronised, when they start or end;
- how to interact with the elements in the scene (media or graphics), e.g. when a user clicks on an image;
- and if the scene is changing, how the scene changes happen.

A scene description may change by means of animations. The different states of the scene during the whole animation may be deterministic (i.e. known when the animation starts) or not. The former case is illustrated by parametric animations. The latter case is illustrated by, for instance, a server sending modification to the scene on the fly. The sequence of a scene description and its timed modifications is called a scene description stream.

The scene description format specified herein is called **LASeR**. A scene description stream is called a **LASeR Stream**. Modifications to the scenes are called **LASeR Commands**. A command is used to act on elements

or attributes of the scene at a given instant in time. LASeR Commands that need to be executed at the same time are grouped into one **LASeR Access Unit (AU)**.

This specification defines an XML language to describe scenes which can be encoded with the LASeR format defined throughout subclauses 6.5 to 6.8.53. The exact XML syntax for these elements and attributes is described in the schemas provided as electronic attachments to this specification.

This specification also defines a binary format to efficiently represent 2D scene descriptions.

6.2. Relationship with SVG

6.2.1. Scene tree

The scene constructs on which the binary format defined in this specification is based are the elements defined by the W3C in the SVG specification [W3C SVG11] [W3C SVGT12]. Subclause 6.8 explicitly refers to the SVG or SMIL elements and attributes which can be encoded using the binary format defined in this specification. A LASeR scene is an SVG scene possibly with LASeR extensions. These extensions are also defined in this subclause. This specification defines in subclause 6.6.2.4 a set of commands, called LASeR Commands, which can be applied to a LASeR scene.

The API defined in Appendix A of [2] with IDL definitions in Appendix B of the same document can be used to access the LASeR scene tree from programming languages such as ECMA-Script [ISO/IEC 16262], Java [5], C [ISO/IEC 9899:1990] or C++ [ISO/IEC 14882:2002].

6.2.2. Fonts

LASeR supports the encoding of fonts. Fonts may be encoded separately from the scene using ISO/IEC 14496-18, and sent as a media stream together with the scene stream. Fonts may also be transmitted using SVG elements related to font encoded using the `privateAnyXMLElement` SDL construct, as specified in 12.2.1 and 12.2.3.

NOTE 1 to encode SVG scenes with SVG fonts in LASeR, font information shall be extracted from the SVG scene, encoded separately and sent as a media stream. MPEG-4 Part 18 is one option to encode and transmit the font, and more options may be specified in the future.

NOTE 2 when using LASeR to encode an SVG scene which includes SVG Fonts derived from OpenType fonts, a better quality can be achieved by transmitting the original OpenType fonts.

NOTE 3 care should be taken when extracting font information from an SVG scene that the effective target of references into the SVG scene, e.g. from scripts, is not changed. One possible way is to replace the extracted font element with a suitable supported (possibly empty) element.

```
<?xml version="1.0" encoding="UTF-8"?>
<svg width="480" height="360" viewBox="0 0 480 360" version="1.1"
     baseProfile="tiny">
        <defs>
            <font horiz-adv-x="959">
                <font-face font-family="TestComic" .../>
                <missing-glyph horiz-adv-x="1024" d="M128 0V1638H896V0H1…"/>
                <glyph unicode="@" horiz-adv-x="1907"
                       d="M1306 412Q1200 412 1123 443T999 ..."/>
                <glyph unicode="A" horiz-adv-x="1498"
                       d="M1250 -30Q1158 -30 1090 206Q1064 ..."/>
                <glyph unicode="y" horiz-adv-x="1066"
                       d="M1011 892L665 144Q537 -129 469 ..."/>
                <glyph unicode="Ö" horiz-adv-x="1635"
                       d="M802 -61Q520 -61 324 108Q116 ..."/>
                <glyph unicode="ç" horiz-adv-x="1052"
                       d="M770 -196Q770 -320 710 -382T528 ..."/>
            </font>
        </defs>
        <g transform="translate(165, 220)" font-family="TestComic"
                font-size="60" fill="black" stroke="none">
            <line x1="0" y1="0" x2="210" y2="0" stroke-width="1"
                  stroke="#888888"/>
            <text>AyÖ@ç</text>
        </g>
</svg>
```

Example 1 — SVG scene with embedded font information

```
<?xml version="1.0" encoding="UTF-8"?>
<saf:SAFSession xmlns:saf="urn:mpeg:mpeg4:SAF:2005" …>
    <saf:sceneHeader>
        <LASeRHeader …/>
    </saf:sceneHeader>
    <saf:mediaHeader streamType="12" objectTypeIndication="6" streamID="font"/>
    <saf:mediaUnit streamIDref="font" .../>
      <!--this media unit contains the OpenType font -->
    <saf:sceneUnit>
        <lsru:NewScene>
            <svg width="480" height="360" viewBox="0 0 480 360" version="1.1"
                 baseProfile="tiny">
                <defs>
                    <desc>this was a font</desc>
                </defs>
                <g transform="translate(165, 220)" font-family="TestComic"
                   font-size="60" fill="black" stroke="none">
                    <line x1="0" y1="0" x2="210" y2="0" stroke-width="1"
                          stroke="#888888"/>
                    <text>AyÖ@ç</text>
                </g>
            </svg>
        </lsru:NewScene>
    </saf:sceneUnit>
    <saf:endOfSAFSession/>
</saf:SAFSession>
```

Example 2 — LASeR/SAF equivalent of Example 1

(the remainder of this subclause is informative)

Differences between example 1 and 2 are:

- the SVG scene has been wrapped in a NewScene update, then in a SAF layer.

- the font description is removed from the SVG scene, encoded with MPEG-4 part 18 and placed in a SAF mediaUnit. The attributes streamType=”12” and objectTypeIndication=”6” in the SAF mediaHeader with streamID “font” identify the content of the SAF stream.
- the SAF mediaHeader and SAF mediaUnit are connected through the streamID “font”, which is encoded as a number, and is strictly local to SAF.
- connection between font-family=”TestComic” and the font encoded in the SAF mediaUnit happens through the font name which is part of the OpenType encoding.

6.3. Timing Model

This subclause describes the notion of Scene Times, Media Times, and Encoded Scene Times.

Logically, a LASeR scene at any instant could be represented by an XML document, which appears like an SVG document:

```
<svg>
...
   <animate begin="X" ... \>
...
</svg>
```

Times within this logical XML document are uniformly expressed in scene times. Scene times have a zero origin and the timescale is defined in SVG.

Logically XML fragments are sent in access units which have Media Time timestamps (MT). These may not have a known origin, and are expressed on a timescale declared at the transport layer. Note that the equations below do not show the correction for timescale units, for simplicity.

The XML fragment containing the "svg" element in this example is sent in an access unit which is a NewScene. The media timestamp MT(ns) of that access unit is arbitrary, but the defined SceneTime of it is zero; ST(ns) = 0.

The XML fragment which supplies the construct "r" is sent in a later access unit with media timestamp MT(r). The defined scene time of that access unit is ST(r) = MT(r) - MT(ns).

For LASeR commands inside a conditional element, ST(r) is the scene time when the conditional is activated.

A RefreshScene command has an arbitrary media time, as usual, but contains within the access unit the defined SceneTime for that media time. This enables terminals which "tune in" after the NewScene was sent, or for any other reason did not receive the NewScene, to nonetheless establish Scene Times. The encoder could calculate the value of that scenetime by comparing the media timestamp of the RefreshScene MT(rs) with the media timestamp of the preceding NewScene MT(ns), and sending MT(rs) - MT(ns) .

When a scene segment starts with a NewScene, the scene time is reset to 0. In such a scene segment, the scene time of a LASeR access unit is defined as the difference between the media time of that access unit and the media time of the closest previous NewScene.

When a scene segment does not start with a NewScene, the scene time is not reset to 0 and let T_{s0} be the scene time within the initial scene segment upon reception of the first access unit of that new scene segment. In such a scene segment, the scene time of a LASeR access unit is defined as the difference between the media time of that access unit and the media time of the first access unit of that scene segment incremented by T_{s0}. Note: the determination of T_{s0} will vary if there is any variation in delivery times between terminals.

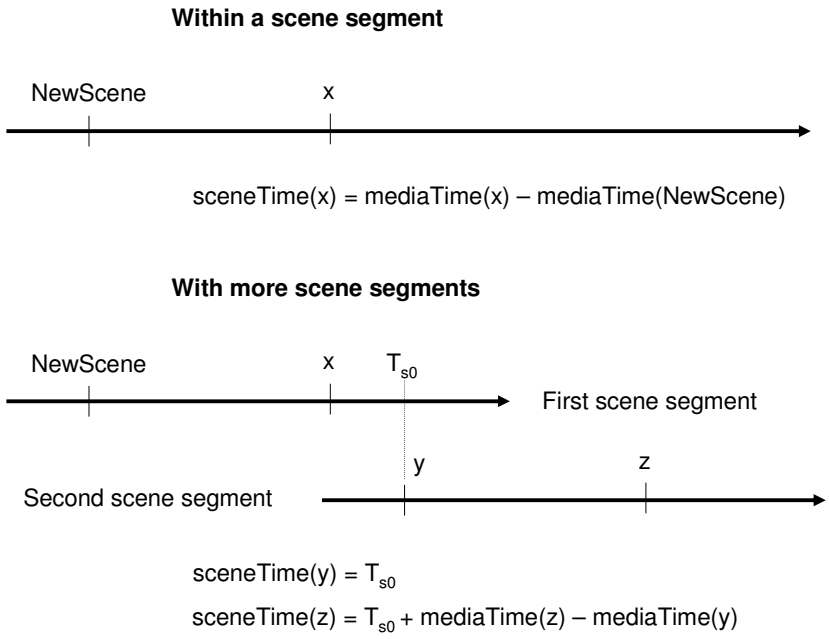


Figure 5 — scene time and scene segments

Time values are encoded in ticks. The number of ticks per seconds for time values relating to the scene time line is defined by the timeResolution and timeEncoding attributes of the LASeRHeader. Attributes “clipBegin” and “clipEnd”, which hold times in a media time line of another stream, are encoded with a predefined resolution of 1000 ticks per seconds.

6.4. Execution Model

An application which shows a presentation comprising a LASeR stream in a way compliant with this specification is called a **LASeR Engine**.

The playback algorithm of a compliant LASeR Engine shall produce the same result as the algorithm described below with the following high-level steps for each execution cycle:

1. Compute the new scene time Ts (begin of execution cycle);
2. Decode any LASeR AU with a scene time below or equal to Ts, and not yet presented in earlier execution cycles;
3. Execute LASeR Commands from LASeR AUs decoded at step 2;
4. Process all events (DOM, SVG or LASeR) according to the DOM event model [3] and resolve all begin and end times that can be resolved according to the SMIL Timing Model, in clause 10 of [SMIL2];
5. Determine active media objects by inspecting begin and end times,
6. For each active media object, present the media access unit with the normal play time equal to clipBegin + (Ts – begin time) and clamp it using clipEnd.
7. Render the audio and visual element of the scene tree according to the SVG rendering model as described in Clause 3 of [W3C SVG11] (end of execution cycle).

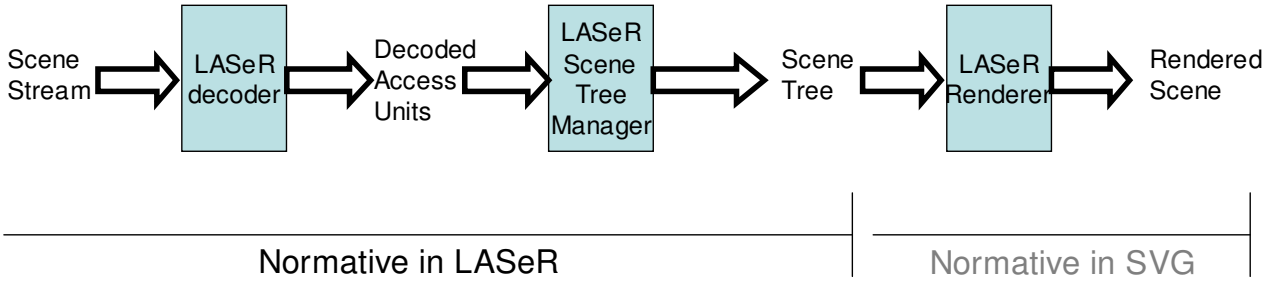


Figure 6 — LASeR engine components and normative parts

6.5. Events

6.5.1. Purposes of events

As in SVG, LASeR defines events following the XML Event specification [W3C XMLEv]. The events defined in LASeR relate to the management of the network session and decoding chains (including decoding buffers). The events defined in the following subclauses can be used by elements in the scene such as script elements being associated, through the listener element, in order to respond to such events.

Note : For instance, in a progressive download scenario, the "buffering" event could be listened by a script in order to trigger a text indicating that content will be played shortly.

```
...
<ev:listener handler="#myscript" event="LASeRBuffering">
<script id="myscript">
   <lsr:Replace ref="#text" attributeName="visibility" value="visible">
</script>
<text id="text" visibility="hidden">Content is being buffered</text>
...
```

In the previous example, the LASeRBuffering event is being listened to by a script "myscript". When the event is launched by the LASeR engine, the visibility attribute of the text element is set to "visible".

These events are launched by the LASeR engine either at the "Network" Layer of the LASeR engine in which case, the scope of these events is the session or at the decoding chain level, in which case the events are at the stream level.

Note : a session is identified by a unique url and streams are identified by a streamID.

6.5.2. Events imported from SVG Tiny

The list of supported events with their properties is given in Table 1.

Event name	Namespace	Description	Bubble	Canc.
“focusin” or “DOMFocusIn”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“focusout” or “DOMFocusOut”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“activate” or “DOMActivate”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes
“click”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes
“mousedown”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes
“mouseup”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes

“mouseover”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes
“mouseout”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	Yes
“mousemove”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“load” (or deprecated “SVGLoad”)	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“resize” (or deprecated “SVGResize”)	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“scroll” (or deprecated “SVGScroll”)	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“zoom” (or deprecated “SVGZoom”)	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“beginEvent”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“endEvent”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“repeatEvent”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	Yes	No
“keyup”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“keydown”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“textInput”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“mouseWheel”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“timer”	http://www.w3.org/2001/xml-events	As defined in section 13.2 of 0.	No	No
“preload”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“loadProgress”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“postLoad”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“connectionConnected”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“connectionClosed”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“connectionError”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“connectionDataSent”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No
“connectionDataReceived”	http://www.w3.org/2000/svg	As defined in section 13.2 of 0.	No	No

Table 1: List of supported events from SVGT1.2

6.5.3. Pseudo-events

Pseudo-events are shortcuts created by combinations of other events. Their definition follows:

Event name	Namespace	Description	Bubble	Canc.
“accessKey(keyCode)”	http://www.w3.org/2000/svg	The key keyCode has been pressed, as defined in section 10.3.1 of [W3C SVGT12]. This pseudo-event is triggered immediately by a listener on keydown placed on the document node.	No	No
“longAccessKey(keyCode)”	urn:mpeg:mpeg4:laser:2005	This pseudo-event is a combination of a listener on keydown and a listener on keyup placed on the document node; this pseudo-event is triggered if after a system-defined time A after the keydown event, no keyup event has happened.	No	No
“repeatKey(keyCode)”	urn:mpeg:mpeg4:laser:2005	This pseudo-event is a combination of a listener on keydown and a listener on keyup placed on the document node; this pseudo-event is triggered after a system-defined time B after the keydown event, repeatedly every system-defined period C, until a keyup event happens.	No	No
“shortAccessKey(keyCode)”	urn:mpeg:mpeg4:laser:2005	This pseudo-event is a combination of a listener on keydown and a listener on keyup placed on the document node; this pseudo-event is triggered if a keyup event happens after a time shorter than A after the keydown event. This is exclusive of longAccessKey(k).	No	No

Table 2: List of pseudo-events

6.5.4. LASeR Events

LASeR defines the following events:

Event name	Namespace	Description	Bubble	Canc.
“pause”	urn:mpeg:mpeg4:laser:2005	Freezes the clock of the timed object they are sent	No	No

		to, and have no effect on non timed objects.		
“play”	urn:mpeg:mpeg4:laser:2005	Starts or resumes the clock of the timed object they are sent to, and have no effect on non timed objects.	No	No
“pausedEvent”	urn:mpeg:mpeg4:laser:2005	Occurs when a Timed Element is paused	Yes	No
“resumedEvent”	urn:mpeg:mpeg4:laser:2005	Occurs when a Timed Element is resumed	Yes	No
“activatedEvent”	urn:mpeg:mpeg4:laser:2005	Occurs when an element is transferred from the waiting tree to the DOM tree.	No	No
“deactivatedEvent”	urn:mpeg:mpeg4:laser:2005	Occurs when an element is transferred from the DOM tree to the waiting tree.	No	No
“screenOrientation0”	urn:mpeg:mpeg4:laser:2005	The screen orientation has changed to typical ‘landscape’ orientation (LASeREvent)	No	No
“screenOrientation90”	urn:mpeg:mpeg4:laser:2005	The screen orientation has changed to typical ‘portrait’ orientation (LASeREvent)	No	No
“screenOrientation180”	urn:mpeg:mpeg4:laser:2005	The screen orientation has changed to inverted ‘landscape’ orientation (LASeREvent)	No	No
“screenOrientation270”	urn:mpeg:mpeg4:laser:2005	The screen orientation has changed to inverted ‘portrait’ orientation (LASeREvent)	No	No
“stop”	urn:mpeg:mpeg4:laser:2005	Upon receiving such an event, a timed element behaves as if the uDOM method endElement() was called. (LASeREvent)	No	No

Table 3: List of LASeR events

Example:

Typical usage for screen orientation events is to have a animation triggered by one of these events, the animation changing the position/rotation of a group of scene elements to match the new screen orientation:

```
<animateTransform begin=”lsr:screenOrientation90” to=”…” xlink:href=”#object1” dur=”1s”/>

<animateTransform begin=”urn:mpeg:mpeg4:laser:2005:screenOrientation90” to=”…” xlink:href=”#object2” dur=”1s”/>

<animateTransform begin=”urn:mpeg:mpeg4:laser:2005:screenOrientation90” to=”…” xlink:href=”#object3” dur=”1s”/>

…

<animateTransform begin=”urn:mpeg:mpeg4:laser:2005:screenOrientation0” to=”…” xlink:href=”#object1” dur=”1s”/>

<animateTransform begin=”urn:mpeg:mpeg4:laser:2005:screenOrientation0” to=”…” xlink:href=”#object2” dur=”1s”/>

<animateTransform begin=”urn:mpeg:mpeg4:laser:2005:screenOrientation0” to=”…” xlink:href=”#object3” dur=”1s”/>

…
```

6.5.5. General IDL definition of LASeR events.

```
interface LASeREvent : events::Event {};
```

All LASeR specific events are prefixed with LASeR.

6.5.6. IDL Events definitions

6.5.6.1. ExternalValueEvent

```
interface ExternalValueEvent : LASeREvent {
       readonly attribute float absoluteValue;
       readonly attribute boolean computableAsFraction;
       readonly attribute float fraction;
};
```

No defined constants

Attributes

- **absoluteValue**: This value represent the status of a resource of any kind, e.g. the remaining battery time.
- **computableAsFraction:** This value indicates whether a fraction can be computed from the absoluteValue.
- **fraction:** This value shall be between 0 and 1 inclusively and represent the status of the resource, e.g. the fraction of remaining battery time over operation time when fully charged.

No defined methods

Example: The following event could be defined:

“batteryState”	urn:example:X	Received by the document at system-dependent intervals and informs about the fraction of battery charge remaining. (ExternalValueEvent)	No	No

and used as follows:

```
<svg viewBox="0 0 1000 1000" baseProfile="tiny" id="root"
   xmlns="http://www.w3.org/2000/svg" version="1.1"
  xmlns:xlink="http://www.w3.org/1999/xlink"
  xmlns:X="urn:example:X">
   <rect width="100" height="100" stroke="blue"
         stroke-width="1">
     <animateColor id="myBatteryAnim" attributeName="fill"
                   begin="indefinite" from="red" to="blue"
                   dur="indefinite">
  </rect>
   <ev:listener observer="root" event="X:batteryState"
                handler="#myBatteryAnim">
</svg>
```

6.6. Encoder Configuration

6.6.1. Overview

The binary encoding has is defined using SDL in 12.

The next subclause describes the syntax for signalling the encoding configuration.

6.6.2. LASeR headers

6.6.2.2. Semantics

The ***LASeRHeader*** specifies the parsing and decoding configuration of a LASeR scene segment.

The ***LASeRUnitHeader*** specifies parameters that may change at the beginning of each LASeR access unit.

LASeR defines a way to partition scenes into incremental scene segments, allowing services to be built of different scene segments, the first scene segment containing a NewScene update, having the ***newSceneIndicator*** bit set, and the other scene segments not starting with a ***NewScene*** update, and being designed as addition to the first scene segment.

LASeR defines an interface to persistent storage. The LASeR engine has the ability to cache selected scene information, using specific LASeR commands, on a best effort basis. The principles behind this caching closely follow the state caching mechanism in HTTP, commonly called cookies [RFC 2965]. Within the LASeR streams, there are two commands that may be used. One command saves, associated with a string name called ***groupID***, the values of some attributes of some nodes. This storage is scoped by the domain-name and path computed from the source using the fields in the LASeR header. The other command restores the attributes (if any) previously saved under the given ***groupID***, as scoped by the domain-name and path.

The stored values are scoped by the domain-name and path. That is, it is possible for both ".acme.com" and ".widget.com" to store data under the same ***groupID*** and for that storage to be distinct; similarly for "/user/laser-expert/" and "/user/laser-novice/" at ".acme.com" to save state under the same ***groupID***, and for those saved states to be distinct.

It is possible that there is state saved under the same ***groupID*** for more than one domain-name/path pair, and that more than one of these match the request-URI. For example, if the request URI has domain "x.y.z.com" and path "/demos/acme", and there is state saved under the same ***groupID*** for domain ".y.z.com" and ".z.com" then both sets of state apply. Under these circumstances, the saved states are ordered primarily by preferring more specific domains (with more components) over less-specific, and then for states with the same domain, and preferring more specific paths (with more components) over less-specific. Once the saved states have been so ordered all the saved states are restored, starting with the least specific (least preferred) and ending with the most specific (most preferred).

For example, if there is saved state under the same ***groupID*** for

1) domain-name ".acme.com", path "/user/laser-expert/";
2) domain-name ".acme.com", path "/user/laser-expert/demo";
3) domain-name "www.acme.com", path "/user/laser-expert/";
4) domain-name "www.acme.com", path "/user/laser-expert/demo"

Then state (1) is restored, then state (2), (3) and (4), in that order. It is possible that these saved states do not overwrite each other (different attributes or nodes), partially overwrite each other (some attributes in common) or completely overwrite each other.

6.6.2.3. Attributes of LASeRHeader

- ***profile***: this value signals the profile of LASeR that the scene segment starting with this LASeRHeader adheres to.
- ***level***: this value signals the level of LASeR which this scene segment starting with this LASeRHeader adheres to.
- ***resolution***: this attribute is a number between -8 and 7 defining the coordinate resolution as $2^{-resolution}$. When reading a coordinate, the encoded value shall be multiplied by the coordinate resolution to obtain the coordinate value expressed in pixels.
  Example:
  When ***resolution*** is 0, 4 or -2, the coordinate resolution is 1, 0.0625 or 4 respectively, and a encoded coordinate value of 100 yields 100, 6.25 and 400 pixels respectively.
- **timeResolution**: this attribute is a 16-bits positive integer defining a resolution for time values (e.g. clock values). When reading a time value from the bit stream, the encoded value shall be divided by this number to obtain a time value in seconds. The default value for timeResolution is 1000.
- ***coordBits***: this attribute defines the number of bits used for encoding coordinates. The default value is 12.

- ***scaleBits_minus_coordBits***: this attribute defines the number of bits above coordBits used for encoding scaling factors. The default value is 0.
- ***colorComponentBits***: this attribute defines the number of bits used for encoding color components.
- ***useFullRequestHost***: this Boolean attribute indicates whether the full domain name of the request-host is used (1) or the first component of the domain name is elided (0). For example, if the source material came from “www.laser.com”, then this differentiates between associating the “service” with “www.laser.com” and “.laser.com”. (Note the definition of local names in the RFC, and the possibility to associate the “service” with locally loaded files, and that the domain name may be either “<hostname>.local” or “.local” in that case.). Together with pathComponents, this attribute defines the “service”.
- ***pathComponents***: this integer attribute indicates how much of the source path is used. If this takes the value 0, then the “service” is not associated with a path, and if it takes the special value 15 (or any value equal to or greater than the number of components in the path) then the entire path is used up to but excluding the final file-name. For example, if the source was “/user/laser-expert/demo/art.mp4” then a value of 4 or greater selects “/user/laser-expert/demo/art.mp4” as the path, the value 2 selects “/user/laser-expert” and the value zero sets no path. Together with useFullRequestHost, this attribute defines the “service”.
- ***newSceneIndicator***: this Boolean indicates if this scene segment starts with a NewScene command or a RefreshScene command. When it is set, the first LASeR command in this scene segment shall be a NewScene or a RefreshScene; otherwise, the first LASeR command in this scene segment shall be neither a NewScene nor a RefreshScene.

  NOTE : the terminal may remove (parts of) the existing scene tree, for resource optimization, when it receives a LASeRHeader with this Boolean set to true.

- ***timeEncoding***: this attribute defines the reference for encoded times: if the value is 0, all times in this segment are relative to the scene time line; if the value is 1, all times in this segment are relative to the beginning of this segment; if the value is 2, all times are encoded in the same way as in the previous segment.
- **pointsCodecType:** this attribute specifies which strategy is used to encode point lists. Possible values are in Table 4.

Table 4 — pointsCodecType values

Point sequence encoding strategy	Code
ExpGolombPointsCodec	0
ISO Reserved	0x1-0x3

- ***reserved***: this ISO reserved attribute has a value of 0.

6.6.2.4. Attributes of LASeRUnitHeader

- ***resetEncodingContext:*** when this Boolean attribute is true, the encoding context defined in subclause 6.6.2.5 shall be reset upon reception of this LASeRUnitHeader.

6.6.2.5. Encoding Context

The encoding context consists of the following sets of associations since either the beginning of the scene or the last LASeRHeader with a ***resetEncodingContext*** attribute set :

- the set of associations between binary indexes and colors,
- the set of associations between binary IDs and font names as used in font-family attributes,
- the set of associations between binary indexes and name spaces,
- the set of associations between binary indexes and private XML tags.

Binary IDs shall not be referred to across scene segments.

6.6.2.6. String IDs Encoding

The element StringIDTable is an optional first child of the LASeRUnit in LASeRML. This StringID table informs the encoder to encode the given IDs as a string ID. When hasStringIDs is set, this table can be omitted as all IDs shall be encoded as strings.

```
<StringIDTable strings="<oneString> <oneString> …"/>
```

An ID that must be encoded as a string ID must be declared at the latest in the StringIDTable of the LASeRUnit where it is first used (by definition or by reference). When the SceneUnit has the RAP attribute set, it shall contain a table containing all already known string IDs.

6.7. LASeR Scene Commands

6.7.1. Overview

Scene Commands are a declarative way (as opposed to programmatic as in a script) of specifying changes to the scene. The following commands are defined:

- ***NewScene***: to create a new scene.
- ***RefreshScene***: to repeat the current state of the scene, for use as a random access point into the LASeR stream or as a means to recover from packet loss.
- ***Insert***: to insert any element in a group, a point in a sequence.
- ***Delete***: to delete any element by id or from a group by index, a point in a sequence.
- ***Replace***: to replace an element by another element (by id or from a group by index), or to replace the value of any attribute of any element.
- ***Add***: similar to replace, but with the notion of adding to the value rather than replacing it.
- ***Save***, ***Restore*** and ***Clean***: to save, reload or remove persistent scene information in the form of the value of a list of attributes. Other commands have no influence on persistent scene information.
- ***SendEvent***: to send an event to any element in the scene.
- ***Activate***: to transfer an element from the waiting tree to the scene tree
- ***Deactivate***: to transfer an element from the scene tree to the waiting tree
- ***ReleaseResource***: to instruct the LASeR engine that a resource (typically media stream) will no longer be used in the scene and may therefore be reclaimed.

The following restrictions apply to all commands:

- Commands shall refer to existing elements and attributes.
- The following attributes cannot be updated: attributeName, id, type, xml:space, preserveAspectRatio, the x and y attributes of the text element and the following attributes of the animation elements: by, from, to, values and fill. This constraint can be worked around by updating the whole element.
- Indexed commands can only be applied to attributes with multiple values or lists of children

Commands not following these restrictions shall be ignored.

In the definition of the commands, two pseudo-attributes are used: **children** refers to the list of children of an element, and **textContent** refers to the text content of an element.

LASeR Commands modify a scene tree regardless of their representation, transport or encoding and regardless of the origin of the scene tree on which they apply. Commands and initial scene may be represented separately in different documents, encoded and delivered in different streams.

A separate DOM tree is defined in addition to the scene tree: the waiting tree. The compositor and renderer have no knowledge of the waiting tree, thus objects in the waiting tree are neither composited nor rendered. The LASeR Commands are extended to search for elements first in the scene tree, then in the waiting tree if the elements are not found in the scene tree. The commands Activate and Deactivate are defined to operate on the waiting tree.

Note – Elements placed in the waiting tree can be updated but are not accessible from the scene tree. A use element pointing to an element placed into the waiting tree behaves as if the referred element did not exist, i.e. nothing is rendered. Deactivating / activating a subtree with listeners may imply unregistering / registering the listeners observing elements not in the waiting tree.

Example:

```
<!-- Module 1 -->
<g id="module1">
      <rect x="0" y="10" width="176" height="20" fill="green"/>
      <text x="88" y="25" text-anchor="middle">Press FIRE</text>
      <ev:listener event="accessKey(FIRE)" handler="#go1"/>
      <ev:listener event="foo.click" handler="#go3"/>
      <animate xlink:href="#foo" …/>
   ...
</g>

<!-- Module 2 -->
<g id="module2">
      <rect x="0" y="10" width="176" height="20" fill="lime"/>
      <text x="88" y="25" text-anchor="middle">Press FIRE</text>
      <ev:listener event="accessKey(FIRE)" handler="#go2"/>
      <ev:listener event="foo.click" handler="#go4"/>
      <animate xlink:href="#foo" …/>
   ...
</g>

<g id=”foo”>
   ...
</g>

<!-- Modules aggregation and control -->
<lsr:conditional begin="0;accessKey(1)">
      <lsr:Deactivate ref="#module2"/>
      <lsr:Activate ref="#module1"/>
</lsr:conditional>
<lsr:conditional begin="accessKey(2)">
      <lsr:Deactivate ref="#module1"/>
      <lsr:Activate ref="#module2"/>
</lsr:conditional>
```

This simple example shows 2 independent modules. Since they use both the FIRE key, they cannot be both “active” at the same time. The two conditionals at the end implement very simply an exclusive activation of the two modules.

6.7.2. Add

6.7.2.7. Semantics

The ***Add*** command is similar to ***Attribute Replacement*** defined in subclause 6.7.8.27, while adding to the designated value instead of replacing it. It thus adds a new value to a specific attribute of an element. The ***ref*** attribute specifies the parent element, the ***attributeName*** attribute specifies which attribute of the parent element is added to, and the ***value*** attribute specifies the added value. For a string, addition is concatenation. For numeric types, addition is done component by component.

The ***Add*** command may also add the value of another attribute of another element to the target attribute. The value to add is then defined by the element id ***operandElementId*** and the attribute name ***operandAttributeName***. The attribute which is the source of the added value shall be of the same type as the target attribute. The only exception is if the target attribute is a string: if the added value belongs to the following types in Table 5, it is then converted to a string and added to the target attribute.

Table 5 — Attribute Values convertible to a string with Add

Values	Format for string conversion
integer	"%d"
float	"%.4f"
point	x and y as "%.4f %.4f"
time	hours, minutes and seconds as "%02d:%02d:%02d"

Examples:

- The script s1 adds 3 to the font-size of the text element with ID "txt1". Resulting font-size is 15.
- The script s2 adds " bar" to the text content of the text element with ID "txt2". Resulting text content is "foo bar".
- The script s3 adds the value of textContent of the text element with ID "txt3" (i.e. "service?3") to the xlink:href attribute of the a element with ID "a1". Resulting xlink:href is "http://www.example.org/service?3"

```
<text id="txt1" font-size="12" …>
<script id="s1">
   <lsr:Add ref="txt1" attributeName="font-size" value="3"/>
</script>

<text id="txt2">foo</text>
<script id="s2">
   <lsr:Add ref="txt2" attributeName="textContent" value=" bar"/>
</script>

<text id="txt3">service?3</text>
<a id="a1" xlink:href="http://www.example.org/">…</a>
<script id="s3">
   <lsr:Add    ref="a1"    attributeName="xlink:href"    operandElementId="txt3"
       operandAttribute="textContent"/>
</script>
```

The value designated by ***operandElementId*** and ***operandAttributeName*** is the presentation value. The following attributes cannot be added to: ***children***, ***d***, ***mpath***, ***begin***, ***end***.

6.7.2.8. Attributes

- ***ref***: the id of the element on which the addition will be applied.
- ***value***: the (constant) added value.
- ***operandElementId***: the id of the element from which the added value is taken
- ***operandAttributeName***: the name of the field from which the added value is taken
- ***attributeName***: the name of the field on which the addition will be applied.

6.7.2.9. DOM Formulation

This subclause is informative.

```
//
// partial implementation of incremental update
// adds to 'transform', numeric fields or strings
//
function AddToValue(parentId, field, value) {
  var parent = document.getElementById(parentId);
  if (parent != null) {
    if (field == 'transform') {
      // manipulate as matrices
      var m = new Matrix();
      m.initFromString(parent.getAttribute(field));
      var p = new Matrix();
      p.initFromString(value);
      m.concatRight(p);
      parent.setAttribute(field, m.toString());
    } else {
      var prevval = parseFloat(parent.getAttribute(field));
      var numval = parseFloat(value);
      if (prevval != 'NaN' && numval != 'NaN') {
        // if both are numbers, add
        parent.setAttribute(field, prevval + numval);
      } else {
        // if either is not a number, concatenate as strings
        parent.setAttribute(field, parent.getAttribute(field) + value);
      }
    }
  } else {
    alert("parent not found in add to field");
  }
}
```

6.7.3. Clean

6.7.3.10. Semantics

The ***Clean*** command erases the storing area identified by the attribute ***groupID*** with the most specific matching defined in 6.6.2.2. The element information stored in the corresponding memory area is not available anymore.

6.7.3.11. Attributes

- ***groupID***: this string attribute defines the group ID as defined in 6.6.2.2.

6.7.4. Delete

6.7.4.12. Semantics

The ***Delete*** command deletes an element by id (use of ***ref*** attribute only) or at a specific position in a parent element (use of ***ref*** and ***index*** attributes).

The element deletion specified with a ***ref*** (no use of the ***index*** attribute) deletes the element.

The element deletion specified with a ***ref***, an ***attributeName*** and an ***index*** deletes the index-th child of the referenced element. The behaviour is undefined if the attribute mentioned in attributeName is not a list.

When deleting an element all its descendant nodes and attributes are removed from the scene graph.

6.7.4.13. Attributes

- ***ref***: this attribute defines the id of the element that shall be deleted or modified.
- ***index***: this attribute defines the zero-based index of the element to delete in the list of children.
- ***attributeName***: when index is specified, this attribute defines the name of the attribute in which the deletion happens, by default “children”. When index is not specified, the whole attribute is removed.

6.7.4.14. DOM Formulation

This subclause is informative.

```
//
// auxiliary function for delete node by id
//
function DeleteNodeInternal(elem, id, poundId) {
   if (elem) {
      var i;
      if (elem.hasChildNodes()) {
         for (i=0; i<elem.childNodes.length; i++) {
            var child = elem.childNodes.item(i);
            if (child.nodeType == 1) {
               if (child.getAttribute("id") == id) {
                     elem.removeChild(child);
                     return;
               }
               DeleteNodeInternal(child, id, poundId);
            }
         }
      }
   }
}

//
// implementation of remove child indexed
//
function DeleteChildIndexed(parentId, index) {
  var parent = document.getElementById(parentId);
  if (parent != null) {
    var child = getElementChild(parent, index);
    if (child != null) parent.removeChild(child);
    else alert("child not found");
  } else {
    alert("parent not found in delete");
  }
```

```
}

//
// implementation of delete node by id
//
function DeleteNode(nodeId) {
  // remove all instances of nodeId wherever they are
  // including use
  DeleteNodeInternal(document, nodeId, "#"+nodeId);
}
```

6.7.5. Insert

6.7.5.15. Semantics

The ***Insert*** command inserts an element or value in a list.

Examples (fragments):

```
    <NewScene>
        <svg id="root" width="333" height="250">
            <g>…</g>
        </svg>
    </NewScene>

    <Insert ref="root">
        <g id="Dictionary" visibility="hidden"/>
    </Insert>

    <Insert ref="Dictionary" attributeName="children">
        <polyline id="Shape4" stroke="0.0 0.0 0.019607844"
            points="-166.5 359.9 984.6 356.65 983.65 358.5"/>
    </Insert>

    <Insert ref="Shape4" attributeName="points" value="0.65 8.5" index="0">
```

In the above sample, the first ***Insert*** adds the g with id Dictionary after the single g object already present in the root svg. The ***attributeName*** attribute has a default value of “children”. The second ***Insert*** adds a polyline at the end of the currently empty Dictionary ***g***. The third ***Insert*** adds one point at the beginning of the ***points*** attribute of the Shape4 ***SVG polyline***.

6.7.5.16. Attributes

- ***ref***: this attribute defines the id of the insertion point. In the absence of a *ref* specification, the default insertion point is the root ***SVG svg*** element.
- ***index***: this attribute defines the index at which to insert the child. In the absence of an index, the child is inserted at the end of the children list. In the case of indexed inserts to the begin/end attributes of all timed elements, the index is ignored, which means the insertion will be done at the end of the list. Indexed inserts to the attributes of type path, namely d and mpath, are not allowed.
- ***attributeName***: this attribute defines the name of the attribute (which must be a list) in which the insertion happens, by default “children”.
- ***value***: the (constant) inserted value.

6.7.5.17. Children

Any element.

6.7.5.18. DOM Formulation

This subclause is informative.

```
//
// auxiliary function: get the nth child of type element
// this is necessary because scene updates only apply to elements
// and yet there are many other types of nodes in the tree
// (text, comment, ...)
//
function getElementChild(element, i) {
  if (i < 0) return getLastElementChild(element);
  var k;
  for (k = 0; k < element.childNodes.length; k++) {
    var el = element.childNodes.item(k);
    if (el != null && el.nodeType == 1) {
      if (i == 0) return el;
      else i--;
    }
  }
  return null;
}

//
// auxiliary function: similar to getElementChild for the last child
//
function getLastElementChild(element) {
  var k;
  for (k = element.childNodes.length - 1; k >= 0; k--) {
    var el = element.childNodes.item(k);
    if (el != null && el.nodeType == 1) {
      return el;
    }
  }
  return null;
}

//
// implementation of insert child indexed
//
function InsertChildIndexed(parentId, child, index) {
  var parent = document.getElementById(parentId);
  if (parent != null && child != null) {
    parent.insertBefore(child, getElementChild(parent, index));
  } else {
    alert("parent or child not found in insert at");
  }
}
```

6.7.6. NewScene

6.7.6.19. Semantics

The ***NewScene*** command inserts a new scene in the LASeR engine. Any currently playing scene is stopped, its resources reclaimed and it is replaced by the scene contained in the ***NewScene*** command. The scene time is reset to 0.

NOTE it is not possible to reuse an element from the previous scene in the new scene, even by an id reference.

6.7.6.20. Attributes

No attributes

6.7.6.21. Children

A single ***SVG svg*** element, which constitutes the initial state of a new scene.

6.7.6.22. DOM Formulation

This subclause is informative.

```
function NewScene(svg) {
  document.svgDocument.root = svg;
}
```

6.7.7. RefreshScene

6.7.7.23. Semantics

The **RefreshScene** command provides a functionally identical copy of the current scene, to a LASeR engine that may have lost some information since the previous NewScene or RefreshScene command, or which has not yet seen a NewScene or RefreshScene command. For LASeR engines in other states (i.e. those that have a scene for which they have received all information) this command shall be ignored. For LASeR engines that interpret this command, it is functionally identical to NewScene except that the scene time is reset to the given value instead of 0. After this command has passed, it should not be possible to differentiate the state of a LASeR engine that did not need to interpret it, and skipped it, from one that did interpret it; the scene graph and scene time in the two LASeR engines should identical.

A LASeR Access Unit carrying a RefreshScene command shall be indicated as a random access point at the transport protocol level. A terminal may skip such an access unit.

6.7.7.24. Attributes

- **time**: the current scene time to set if this command is interpreted. This attribute is expressed as a number of ticks, using the LASeR time resolution.

6.7.7.25. Children

A single **SVG svg** element, which constitutes the initial state of the scene.

6.7.7.26. DOM Formulation

This subclause is informative.

```
function NewScene(svg, time) {
  document.svgDocument.root = svg;
  document.time = time;
}
```

6.7.8. Replace

6.7.8.27. Semantics

The ***Replace*** command has two variants: Element Replacement and Attribute Replacement.

The ***Element Replacement*** replaces an existing element and its replacement with a new element. The ***ref*** attribute specifies the element to be replaced and the element given as child of the **Replace** is used as replacement. The element replacement replaces the element and all its instances, if it was referenced by a ***SVG use***.

The ***Attribute Replacement*** command replaces a specific attribute of an element with a new value. The ***ref*** attribute specifies the parent element, the ***attributeName*** attribute specifies which attribute of the parent element is replaced, and the ***value*** attribute or the content of the Replace element specifies the new value. If ***index*** is specified, then the replacement is done on the value with rank ***index*** in the list.

When replacing begin and end attributes, the pseudo-event executionTime can be used to signify the time at which the command is executed. After execution, this pseudo-event is replaced by its time value.

The ***Replace*** command may also replace the target value with the value of another attribute of another element. The replacing value is then defined by the element id ***operandElementId*** and the attribute name ***operandAttributeName***. The attribute which is the source of the replacing value shall be of the same type as the target attribute. The only exception is if the target attribute is a string: if the added value belongs to the following types in Table 5, it is then converted to a string and added to the target attribute.

NOTE The following attributes cannot be updated: id, by, from, to, values, type, xml:space and fill (from animation elements). This constraint can be worked around by updating the whole element.

The following restrictions apply:

- attributeName may point at a private XML attribute, in which case index is not allowed.
- Indexed replacements to the begin/end attributes of all timed elements are not allowed.
- Indexed replacements to the attributes of type path, namely d and mpath, are not allowed.
- The values ***id***, ***begin***, ***end***, ***d*** and ***mpath*** are not allowed for operandAttributeName.

The value designated by ***operandElementId*** and ***operandAttributeName*** is the presentation value as defined in the SVG specification.

Attribute replacement with attributeName=”textContent” and a value of index replaces the index-th component of the text content of the target element, if it exists. Without an index value, the text content is flattened and replaced as a whole, as in text editing. In the absence of text content, this command is ignored.

6.7.8.28. Attributes

- ***ref***: this attribute defines the id of the element on which the replacement will be applied.
- ***index***: this attribute defines the position of the replacement
- ***value***: the (constant) replaced value.
- ***operandElementId***: this attribute defines the id of the element from which the added value is taken
- ***operandAttributeName***: this attribute defines the name of the field from which the added value is taken
- ***attributeName***: this attribute defines the name of the replaced attribute. Acceptable values are the name of any attribute of a LASeR element or the name of any private XML attribute used in the scene.

6.7.8.29. Children

Any element or text content

6.7.8.30. DOM Formulation

This subclause is informative.

```
//
// implementation of replace node by id
//
function Replace(nodeId, newNode) {
```

```
  // replace all instances of nodeId wherever they are
  // including use
  if (document.getElementById(newNodeId) == null) {
    alert("replacement element not found");
  } else {
    ReplaceInternal(document, nodeId, "#"+nodeId, newNode);
  }
}


//
// auxiliary function for replace node by id
//
function ReplaceInternal(elem, id, poundId, newNode) {
   if (elem) {
      var i;
      if (elem.hasChildNodes()) {
         for (i=0; i<elem.childNodes.length; i++) {
            var child = elem.childNodes.item(i);
            if (child.nodeType == 1) { // element node
               if (child.getAttribute("id") == id) {
                     elem.replaceChild(newNode, child);
                     return;
               }
               if (child.nodeName == "use") {
                  if (child.getAttributeNS("http://www.w3.org/1999/xlink, href")
                       == poundId) {
                     elem.replaceChild(newNode, child);
                     return;
                  }
               }
               ReplaceInternal(child, id, poundId, newNode);
            }
         }
      }
   }
}

//
// implementation of replace field
//
function ReplaceField(parentId, field, value) {
  var parent = document.getElementById(parentId);
  if (parent != null) {
    parent.setAttribute(field, value);
  } else {
    alert("parent not found in replace field");
  }
}

//
//implementation of replace child indexed
//
function ReplaceChildIndexed(parentId, child, index) {
  var parent = document.getElementById(parentId);
  if (parent != null) {
    if (child != null) {
      var replaced = getElementChild(parent, index);
      if (replaced != null) parent.replaceChild(child, replaced);
      else alert("nothing found to replace");
```

```
    } else alert("child not found");
  } else {
    alert("parent not found in replace indexed");
  }
}
```

6.7.9. Restore

6.7.9.31. Semantics

The ***Restore*** command restores attributes that have been stored by the ***Save*** command. The retrieved values will replace the current attributes in the scene graph. If any saved and restored attribute type does not match, the whole restore command is ignored.

6.7.9.32. Attributes

- ***groupID***: this string attribute defines the group ID as defined in 6.6.2.2.

6.7.10. Save

6.7.10.33. Semantics

The ***Save*** command stores in memory a selection of attributes from elements contained in the current scene graph. The saved value is the decoded (or DOM) value.

The ***useFullRequestHost*** and ***pathComponents*** attributes defined in the LASeRHeader specify a unique memory area where the element information will be stored. The ***groupID*** allows for the same element information to be saved in different areas at different times.

Only attributes of elements identified by string IDs can be saved and restored reliably across scenes. Attributes referencing elements by ID can only be saved and restored reliably across scenes if they hold a string ID.

6.7.10.34. Attributes

- ***groupID***: this string attribute defines the group as defined in 6.6.2.2.
- ***elements***: this attribute defines a list of element ids.
- ***attributes***: this attribute defines a list of attribute names, one per element id in the previous list, each pair (element id, attribute name) defining one of the attributes to be saved. The following attributes are not allowed in the ***attributes*** list: ***children***, ***begin***, ***end***.

6.7.11. SendEvent

6.7.11.35. Semantics

The ***SendEvent*** command is used to send an event to an object, for example sending the ***activate*** event to a script to trigger its execution.

6.7.11.36. Attributes

- ***ref***: this attribute defines the id of the element to which the event will be send.
- ***event***: this attribute defines the type of the event that is sent. The list of supported events is defined in 6.5.
- ***pointvalue***: this attribute defines the value of the event in the case of a mouse event
- ***keyCode***: this attribute defines the value of the event in the case of a accesskey or longaccesskey event. Key codes are defined in 6.5.

- ***intvalue***: this attribute defines the value of the event in the case of e.g. an error event
- ***stringvalue***: this attribute defines the value of the event in the case of a text event
- ***intvalue2***: this attribute defines the second int value of the event.

The above attributes carry information from the interface of the events, as specified in the next table:

Name of event	Attribute of event	Attribute of SendEvent
TimeEvent	detail	intvalue
UIEvent	detail	intvalue
WheelEvent	wheelDelta	intvalue
MouseEvent	screenX	*not carried*
	screenY	*not carried*
	clientX	pointvalue
	clientY	pointvalue
	button	intvalue
TextEvent	data	stringvalue
KeyEvent	keyIdentifier	stringvalue when unknown keyIdentifier, intvalue when a LASeR key code is defined
ProgressEvent	lengthComputable	*not carried, the LASeR engine shall set this value to true if the value total is present*
	loaded	intvalue
	total	invalue2

6.7.12 Activate

6.7.12.1 Semantics

The Activate command reverses the effect of the Deactivate command, i.e. it restores the target element from the waiting tree to the scene tree. From the DOM point of view, the effect on the element is that the lsr:ghost element is replaced by the element, e.g. using a DOM replaceChild.
Note – The lsr:ghost element may have been transferred to the waiting tree as part of a Activate command applied to one of its ancestors.

6.7.12.2 Attributes

- ***ref***: the id of the element which shall be restored from the waiting tree to the scene tree.

6.7.13 Deactivate

6.7.13.1 Semantics

The Deactivate command takes an element and places it into the waiting tree, so that it can be restored later to the scene tree by a Activate command. From the DOM point of view, the effect on the element is:

- the element is replaced by an lsr:ghost element, e.g. using a DOM replaceChild, regardless of whether the element is in the scene tree or in the waiting tree. Note: the element could already be in the waiting tree if one of its ancestors has been placed in the waiting tree.
- the element is placed in the waiting tree.

Note – The lsr:ghost element is never encoded nor transmitted nor rendered.

6.7.13.2 Attributes

- ***ref***: the id of the element which shall be placed in the waiting tree.

6.7.14 ReleaseResource

6.7.14.1 Semantics

The ReleaseResource command instructs the LASeR engine that the indicated resource will no longer be used in the scene. All associated LASeR engine resources (memory, I/O) may be reclaimed. All scene elements that may be referencing this resource behave as if they pointed at a non available resource.

6.7.14.2 Attributes

- ***ref***: this attribute defines the resource which may be discarded. It may be a stream ID or any other URI.

6.8. Scene Description Elements

6.8.1. Conventions

For each element defined in other specifications (SVG or SMIL), the reference to the SVG or SMIL element definition is given. When semantics are identical, only that reference is provided. The list of possible attributes for an element is given in the summary table in subclause 6.8.53. Only changes and extensions are defined in the element specification within subclause 6.8.

6.8.2. General information

All elements have two optional attributes for assigning unique identifiers to elements: the ***SVG id*** attribute is defined in [W3C SVG11] and the ***XML id*** attribute is defined in [xml:id]. Only one of this attribute shall be used at a time on an element.

Some of the SVG attributes are derived from [W3C CSS]. They can be specified on any parent of the elements that are going to use their value. This behavior is called *property inheritance* as defined in subclause 6.2 of [W3C CSS].

SVG defines the notion of locatable elements. Locatable elements have a transform attribute. LASeR extends the attributes allowed on those elements for efficient updating through LASeR Commands Replace and Add. The extension consists in adding the ***scale, rotation*** and ***translation*** attributes. The respective encoding of ***SVG transform*** and these attributes are:

- ***SVG transform*** holds a full matrix and is exclusive of the other attributes.
- ***scale*** holds a (sx, sy) scaling factor which is applied before rotation or translation if present; sx is a positive number, only sy is allowed to be negative.
- ***rotation*** holds a rotation angle which is applied after a possible scale and before translation.
- ***translation*** holds a (tx, ty) translation vector which is applied after scale and rotation if present.

When updating a matrix using LASeR Commands applying to scale or rotate only, the LASeR engine shall decompose the matrix as a sequence of scale then rotate then translate in this order. If this recovery is unsuccessful, the LASeR Commands applying independently to scale, rotation and translation shall be ignored. For instance, when replacing the scale on an object with an existing matrix, the LASeR engine needs to isolate the current scale factors from the rest of the matrix. It does that by decomposing the matrix in the order given above.

6.8.3. SVG a

The ***SVG a*** element is specified in subclause 17.1 of [W3C SVG11].

The expression “linked content” is used in the definition of the target attribute in [W3C SVG11]. For LASeR content, if the incoming data starts with a NewScene, “linked content” is the scene described by the incoming

data; if the incoming data starts with any other command than a NewScene or RefreshScene, “linked content” is the current scene modified by the commands contained in the incoming data.

When an a element is activated, the incoming data replaces the previously incoming LASeR Commands and media.

6.8.4. SVG animate

6.8.4.37. Semantics

The ***SVG animate*** element is specified in subclause 19.2.10 of [W3C SVG11].

When an animate element is not active and it receives an ExternalValueEvent, then the element sets the animation target attribute value as if the animation progress was at the fraction contained in the ExternalValueEvent.

When an animate element is not active and it receives a ProgressEvent which has lengthComputable=true, then the element sets the animation target attribute value as if the animation progress was at the fraction loaded over total.

6.8.4.38. Attributes

- enabled: this Boolean attribute specifies whether the element is animating its target or not. This attribute does not influence the activation or deactivation of the element by events, nor the sending of events, so has no influence on the SMIL Timing model. This attribute is not animatable and not inheritable. The default value for this attribute is ‘true’.

6.8.5. SVG animateColor

6.8.5.39. Semantics

The ***SVG animateColor*** element is described in subclause 19.2.13 of [W3C SVG11].

When an animateColor element is not active and it receives an ExternalValueEvent, then the element sets the animation target attribute value as if the animation progress was at the fraction contained in the ExternalValueEvent.

When an animateColor element is not active and it receives a ProgressEvent which has lengthComputable=true, then the element sets the animation target attribute value as if the animation progress was at the fraction loaded over total.

6.8.5.40. Attributes

- enabled: this Boolean attribute specifies whether the element is animating its target or not. This attribute does not influence the activation or deactivation of the element by events, nor the sending of events, so has no influence on the SMIL Timing model. This attribute is not animatable and not inheritable. The default value for this attribute is ‘true’.

6.8.6. SVG animateMotion

6.8.6.41. Semantics

The ***SVG animateMotion*** element is described in subclause 19.2.12 of [W3C SVG11].

When an animateMotion element is not active and it receives an ExternalValueEvent, then the element sets the animation target attribute value as if the animation progress was at the fraction contained in the ExternalValueEvent.

When an animateMotion element is not active and it receives a ProgressEvent which has lengthComputable=true, then the element sets the animation target attribute value as if the animation progress was at the fraction loaded over total.

6.8.6.42. Attributes

- enabled: this Boolean attribute specifies whether the element is animating its target or not. This attribute does not influence the activation or deactivation of the element by events, nor the sending of events, so has no influence on the SMIL Timing model. This attribute is not animatable and not inheritable. The default value for this attribute is 'true'.

6.8.7. SVG animateTransform

6.8.7.43. Semantics

The ***SVG animateTransform*** element is described in subclause 19.2.14 of [W3C SVG11].

When an animateTransform element is not active and it receives an ExternalValueEvent, then the element sets the animation target attribute value as if the animation progress was at the fraction contained in the ExternalValueEvent.

When an animateTransform element is not active and it receives a ProgressEvent which has lengthComputable=true, then the element sets the animation target attribute value as if the animation progress was at the fraction loaded over total.

6.8.7.44. Attributes

- enabled: this Boolean attribute specifies whether the element is animating its target or not. This attribute does not influence the activation or deactivation of the element by events, nor the sending of events, so has no influence on the SMIL Timing model. This attribute is not animatable and not inheritable. The default value for this attribute is 'true'.

6.8.8. SMIL audio

6.8.8.45. Semantics

The ***SMIL audio*** element is defined in subclause 7.3.1 of [SMIL2]**Erreur ! Source du renvoi introuvable.**.

6.8.8.46. Attributes

- clipBegin: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The value of clipBegin is taken into account when the media element is activated. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.

- clipEnd: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The value of clipEnd is taken into account when the media element is activated. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.

- syncReference: this attribute holds a reference to the stream whose clock acts as a clock reference for the stream referred to by this element. This attribute is not animatable and not inheritable.

- syncBehavior: this attribute is defined in subclause 6.3.1 of [SMIL2]. This attribute is not animatable and not inheritable.

- syncTolerance: this attribute is defined in subclause 6.3.1 of [SMIL2]. This attribute is not animatable and not inheritable.

6.8.9. SVG circle

The ***SVG circle*** element is defined in subclause 9.3 of [W3C SVG11].

6.8.10. LASeR conditional

6.8.10.47. Semantics

The ***LASeR conditional*** element allows sets of scene commands to be inserted in the scene, for later execution upon activation by time or through the ***XML Events listener*** element.

Upon activation, a conditional element emits a beginEvent and the expression “condID.begin” where condID is the ID of a conditional element can be used in begin/end attributess.

The children of a conditional element are not accessible through DOM or LASeR Commands.

6.8.10.48. Attributes

- **begin**: this attribute specifies the time at which the conditional element is triggered. This attribute is not animatable and not inheritable. The default value is indefinite.

- **enabled**: this Boolean attribute specifies whether the element is activatable or not. This attribute is not animatable and not inheritable. The default value is true.

6.8.11. LASeR cursorManager

6.8.11.49. Semantics

The LASeR cursorManager element may point to any LASeR element to confer it the semantics of virtual pointer. The virtual pointer may be moved through LASeR Commands or other interaction by moving the object associated with the virtual pointer. When moved in and out of mouse-sensitive elements, the same events shall be generated as if a pointing device such as a mouse had been moved in its stead. For example:

- when the virtual pointer is moved in a mouse-sensitive region, a mouseover is generated,
- when the virtual pointer is moved out of a mouse-sensitive region, a mouseout is generated,
- when the virtual pointer is moved, a mousemove is generated,
- when the virtual pointer is located on a mouse-sensitive region and a activate event is received by the virtual pointer, the activate event is translated to a click event sent to the region.

LASeR elements implementing the interaction moving the object associated with the virtual pointer shall be placed as children of the cursor element.

Example:

```
<lsr:cursorManager id=”cursorManager1” xlink:href="#vp">
  <!-- pointer movement -->
  <ev:listener event="accessKey(UP)" handler="#s1"/>
  <lsr:conditional id="s1">
    <lsr:Add ref="g" attribute="translation" pointvalue="0 -5"/>
  </lsr:conditional>
  <ev:listener event="accessKey(DOWN)" handler="#s2"/>
  <lsr:conditional id="s2">
    <lsr:Add ref="g" attribute="translation" pointvalue="0 5"/>
  </lsr:conditional>
  <ev:listener event="accessKey(LEFT)" handler="#s3"/>
```

```
  <lsr:conditional id="s3">
    <lsr:Add ref="g" attribute="translation" pointvalue="-5 0"/>
  </lsr:conditional>
  <ev:listener event="accessKey(RIGHT)" handler="#s4"/>
  <lsr:conditional id="s4">
    <lsr:Add ref="g" attribute="translation" pointvalue="5 0"/>
  </lsr:conditional>
</lsr:cursorManager>
<!-- activate event to be translated to a click -->
<ev:listener event="accessKey(FIRE)" handler="#cursorManager1"/>
<!-- mouse sensitive shape -->
<polygon …>
   <ev:listener event="click" handler="#someConditional"/>
</polygon>
<g id="vp" fill="black">
  <!-- crosshair -->
  <line x1="-5" y1="0" x2="5" y2="0"/>
  <line x1="0" y1="-5" x2="0" y2="5"/>
</g>
```

6.8.11.50. Attributes

Attributes of lsr:cursorManager are x, y and xlink:href as defined on the cursor element of SVG 1.1 [W3C SVG11]

6.8.12. SVG defs

The ***SVG defs*** element is specified in subclause 5.3 of [W3C SVG11].

6.8.13. SVG desc

The ***SVG desc*** element is specified in subclause 5.4 of [W3C SVG11].

6.8.14. SVG ellipse

The ***SVG ellipse*** element is defined in subclause 9.4 of [W3C SVG11].

6.8.15. SVG foreignObject

The ***SVG foreignObject*** element is described in subclause 23.3 of [W3C SVG11].

6.8.16. SVG g

The ***SVG g*** element is described in subclause 5.2 of [W3C SVG11].

6.8.17. SVG image

6.8.17.51. Semantics

The ***SVG image*** element is specified in subclause 5.7 of [W3C SVG11].

6.8.17.52. Attributes

- **transformBehavior**: the values and semantics of this attribute are the same as those of the transformBehavior attribute of the ***SVG video*** element defined in subclause 6.8.40. This attribute is not animatable and not inheritable.

6.8.18. SVG line

The ***SVG line*** element is defined in subclause 9.5 of [W3C SVG11].

6.8.19. SVG linearGradient

The ***SVG linearGradient*** is defined in subclause 13.2.2 of [W3C SVG11].

6.8.20. XML Events listener

6.8.20.53. Semantics

The **XML Events listener** element is defined in subclause 3.1 of the XML Events Specification [W3C XML Events]0. The present specification restricts its usage in a way compatible with the (informative) SVG Tiny 1.2 specification [W3C SVGT12] and also adds compatible extensions.

6.8.20.54. Attributes

- **event:** as specified in the XML Events Specification. The event shall be one of list of supported events as defined in 6.5.

- **phase:** as specified in XML Events Specification with the restriction that the capture phase is not supported. The only allowed value for this attribute is 'default'.

- **handler**: as specified in XML Events Specification with the restrictions that supported handlers depend on the event. If the handler attribute is not present, the default handler is the parent of the listener element.

  The following table summarizes LASeR specific handlers and their associated default action:

Event	Handler	Default Action
activate	a	follow the hyperlink
activate	conditional	executes the conditional content
pause	any element with timing attributes	freezes the time line of this element
play	any element with timing attributes	starts or resumes the time line of this element

  The following table summarizes SVG specific handlers and their associated default action, as defined in the SVG specification:

Event	Handler	Default Action
activate	any editable element	starts the editing
activate	any element with timing attributes	activates the element

  For other couple (event, handler) not defined in this specification, the default action is undefined.

- **enabled:** this Boolean attribute specifies whether the listener is currently active or not. This attribute is not animatable and not inheritable. The default value for this attribute is 'true'.

6.8.20.55. Children

None.

6.8.21. SVG metadata

The ***SVG metadata*** element is defined in subclause 21 of [W3C SVG11].

6.8.22. SVG mpath

The ***SVG mpath*** element is defined in subclause 19.2.12 of [W3C SVG11].

6.8.23. SVG path

The ***SVG path*** element is defined in subclause 8 of [W3C SVG11].

6.8.24. SVG polygon

The ***SVG polygon*** element is defined in subclause 9.7 of [W3C SVG11].

6.8.25. SVG polyline

The ***SVG polyline*** element is defined in subclause 9.6 of [W3C SVG11].

6.8.26. SVG radialGradient

The ***SVG radialGradient*** is defined in subclause 13.2.3 of [W3C SVG11].

6.8.27. SVG rect

The ***SVG rect*** element is defined in subclause 9.2 of [W3C SVG11].

6.8.28. LASeR rectClip

6.8.28.56. Semantics

The **LASeR rectClip** element acts as a clipping element, limiting the rendering of its children to a device-pixel-aligned rectangle, in addition to the semantics of the **SVG g** element. The rectangle is not sensitive to rotation or skew of the local coordinate system, and if scaled, its size is rounded to the nearest pixel. If the rectClip element carries a transform attribute, the corresponding transformation is applied to the rectClip children but not to the clipping rectangle itself.

6.8.28.57. Attributes

- **size**: a pair of coordinates defining the width and height of the clipping rectangle. All children are clipped by this axis-aligned rectangle centered on the origin of the local coordinate system and of size (size.x, size.y). The rectangle is not sensitive to rotation or scale of the local coordinate system. This attribute is animatable but not inheritable.
- **width**: the width of the clipping rectangle. This can also be accessed through the x component of the size field. This attribute is animatable but not inheritable.
- **height**: the height of the clipping rectangle. This can also be accessed through the y component of the size field. This attribute is animatable but not inheritable.
- **x**: the horizontal coordinate of the center of the clipping rectangle. This attribute is animatable but not inheritable.
- **y**: the vertical coordinate of the center of the clipping rectangle. This attribute is animatable but not inheritable.

6.8.29. SVG script

The ***SVG script*** element is specified in subclause 18.2 of [W3C SVG11].

6.8.30. LASeR selector

6.8.30.58. Semantics

The semantics of the **LASeR selector** element is to act as a selection element, rendering zero or one of its children, in addition to the semantics of the **SVG g** element.

In the following, N is the number of children of the selector element. The *choice* attribute determines the actual rendering mode:

1. *choice* >= 0 & *choice* < N: only the child of index *choice* is displayed, i.e. is "on" while the other children are "off". The "off" children are neither composed nor rendered. animate* elements and conditionals in the tree below "off" children are inactive.
2. *choice* == none | *choice* >= N: nothing is displayed. All children are "off", i.e. neither composed nor rendered. animate* elements and conditionals in the tree below "off" children are inactive.
3. *in all other cases*: all the children are displayed at (0,0) of the local coordinate system without clipping.

In case 1. and 2., when a child changes from "off" to "on", its children with begin attributes get activated; when a child changes from "on" to "off", its children with end attributes get stopped.

6.8.30.59. Attributes

- **choice:** the rendering mode selector. It is either one of the value 'none' or 'all' or an integer representing the 0-based index of the child to be displayed. This attribute is animatable but not inheritable.

6.8.31. SVG set

The ***SVG set*** element is described in subclause 19.2.11 of [W3C SVG11].

6.8.32. LASeR simpleLayout

6.8.32.60. Semantics

The semantics of the **LASeR simpleLayout** element is to act as a simple layout tool, by spacing its children by a specified amount, thus creating rows or columns of children, in addition to the semantics of the **SVG g** element.

6.8.32.61. Attributes

- **delta:** a pair of coordinates defining the spacing between children. The first child is displayed at (0,0) of the local coordinate system, and the n-th child is displayed at ((n-1)*size.x,(n-1)*size.y) of the local coordinate system. One of the two coordinates shall be 0. This creates a row or column of objects.This attribute is animatable but not inheritable.

NOTE – If a child is hidden, the effect will be that a gap is visible between the previous and following child.

6.8.33. SVG stop

The ***SVG stop*** element is defined in subclause 13.2.4 of [W3C SVG11].

6.8.34. SVG svg

6.8.34.62. Semantics

The ***SVG svg*** element is described in subclause 5.1 of [W3C SVG11].

6.8.34.63. Attributes

- **syncBehaviorDefault**: this attribute is defined in subclause 6.3.1 of [SMIL2].
- **syncToleranceDefault**: this attribute is defined in subclause 6.3.1 of [SMIL2].
- **lsr:fullscreen:** the attribute 'lsr:fullscreen' is defined on the <svg> element to hint that the LASeR scene should be rendered on the entire screen. The attribute can take two values "true" or "false", with false (normal rendering) being the default value. With the attribute set to true the LASeR UE should negotiate the rendering area with its parent UE and get as large part of the screen as possible for the LASeR rendering area. The attribute is neither animatable nor inheritable.

6.8.35. SVG switch

The ***SVG switch*** element is specified in subclause 5.8.2 of [W3C SVG11].

This specification defines four extension strings in order to allow the use of the switch element to control the initial layout of a scene according to the screen orientation:

- urn:mpeg:mpeg4:LASeR:2005:screenOrientation0 for typical 'landscape' orientation
- urn:mpeg:mpeg4:LASeR:2005:screenOrientation90 for typical 'portrait' orientation
- urn:mpeg:mpeg4:LASeR:2005:screenOrientation180
- urn:mpeg:mpeg4:LASeR:2005:screenOrientation270

An example of usage is:

```
<switch>
   <g requiredExtensions="urn:mpeg:mpeg4:LASeR:2005:screenOrientation90">
      … layout for portrait …
   </g>
   <g requiredExtensions="urn:mpeg:mpeg4:LASeR: 2005:screenOrientation0">
      … layout for landscape…
   </g>
</switch>
```

6.8.36. SVG text

6.8.36.64. Semantics

The ***SVG text*** element is defined in subclause 10.4 of [W3C SVG11], with the ***SVG text-decoration*** attribute from subclause 10.12 of [W3C SVG11].

6.8.36.65. Attributes

- Additional values for the text-decoration attributes are defined, taken from [4]: "LASeR_OUTLINE", "LASeR_EMBOSS", "LASeR_ENGRAVE", "LASeR_LEFTDROPSHADOW", "LASeR_RIGHTDROPSHADOW".

NOTE – Authors who want their content to degrade gracefully on pure SVG client should make sure that either: the default value for text-decoration is appropriate or that the requiredExtensions attribute is used.

- SVG font-size: the size of the font represents the height value of the EM-box of a font in the local coordinate system.
- SVG editable: If set to "false" (default) the contents of the text element are not editable in place through the LASeR engine. If set to "true", the LASeR engine must provide a way for the user to edit the content of the text element and all contained subelements which are not hidden (with visibility="hidden") or disabled (through the switch element or display="none").

- SVG display-align: this attribute is defined in subclause 10.11.5 of [W3C SVGT12], i.e. it governs alignment in the direction orthogonal to the main direction of the text.

6.8.37. SVG title

The ***SVG title*** element is defined in subclause 5.4 of [W3C SVG11].

6.8.38. SVG tspan

The ***SVG tspan*** element is defined in subclause 10.5 of [W3C SVG11].

6.8.39. SVG use

The ***SVG use*** element is described in subclause 5.6 of [W3C SVG11].

6.8.40. SMIL video

6.8.40.66. Semantics

The ***SMIL video*** element is defined in subclause 7.3.1 of [SMIL2].

6.8.40.67. Attributes

- **clipBegin**: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The value of clipBegin is taken into account when the media element is activated. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.
- **clipEnd**: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.
- **lsr:fullscreen:** This attribute can take the values true and false (the default). When set, the video shall be rendered alone and should fill the full screen area. This means no other visual element in the scene shall be rendered and in every other respect, the processing is as normal, e.g. events must still be dispatched to the appropriate elements and audio elements are rendered. The preserveAspectRatio attribute should be respected. If the preserveAspectRatio attribute indicates uniform scaling, then uniform scaling shall be applied. The viewport-fill-opacity attribute should be ignored. The fullscreen attribute is not animatable and not inheritable. This attribute shall not be set on more than one video element at any time in a scene.

  Note – The full screen area is the largest part of the whole screen that can be used. The presence of this attribute indicates that the content author wishes to have the video presented as large as possible, and with as few as possible other visual elements (e.g. from other applications or from the OS) on the screen. If the video does not fill the rendering area, the viewport fill color is suggested but not required for the unpainted areas.
- **syncReference**: this attribute holds a reference to the stream whose clock acts as a clock reference for the stream referred to by this element. This attribute is not animatable and not inheritable.
- **syncBehavior**: this attribute is defined in subclause 6.3.1 of [SMIL2]. This attribute is not animatable and not inheritable.
- **syncTolerance**: this attribute is defined in subclause 6.3.1 of [SMIL2]. This attribute is not animatable and not inheritable.
- **transformBehavior**: the following values are added to the SVG list of possible values for transformBehavior: “pinned90”, “pinned180” and “pinned270”. The semantics are defined in Table 6. This attribute is not animatable and not inheritable.

Table 6 — Extended values for transformBehavior

Value	Semantics
pinned	Video at the native resolution of the media is painted centered on the local coordinate system origin. The pixels are aligned to the device pixel grid and no resampling will be done.
pinned_90	Video at the native resolution of the media is then painted centered on the local coordinate system origin with a rotation of 90° counter-clockwise. The pixels are aligned to the device pixel grid and no resampling will be done.
pinned_180	Video at the native resolution of the media is then painted centered on the local coordinate system origin with a rotation of 180° counter-clockwise. The pixels are aligned to the device pixel grid and no resampling will be done.
pinned_270	Video at the native resolution of the media is then painted centered on the local coordinate system origin with a rotation of 270° counter-clockwise. The pixels are aligned to the device pixel grid and no resampling will be done.

6.8.41. AnimateScroll

6.8.41.1 Semantics

The purpose of this element is to allow to:

- scroll a set of objects of unknown size, inside a clipping rectangle.
- define the speed of the scrolling in terms of clip size, not in terms of the objects size
- define the bounds of scrolling precisely, according to the actual size of the objects
- define the beginning and ending conditions (screen full or empty of the objects)
- define automatic and manual scrolling (continuous scrolling, page advance on action…), and allow a combination of automatic and manual
- define scroll stops (for manual scrolling) by page (size of viewport) or by object markers

The animateScroll element is designed in a manner similar to SMIL/SVG animate* elements. It works in combination with a clipping region (lsr:rectClip element). Scrolling can be automatic or manual (e.g. triggered by key events). The scrolling effect is obtained by a translation applied to all children of the lsr:rectClip element. Multiple animateScroll and setScroll elements acting on the same objects combine in the same way as multiple animateTransform elements with additive=”sum”.

Each frame where it is active, animateScroll does:

- if the element or one of its attributes has changed, recompute the element size
- depending on the parameters, compute the new position of the scrolled objects

The direction attribute of animateScroll is not sensitive to rotation.

Example 1: Plain vanilla

```
<lsr:rectClip id="t1" size="120 20">
  <text…>Que j’aime à faire apprendre ce nombre utile au sage. Immortel
Archimède, toi de qui Syracuse garde encore la mémoire.</text>
</lsr:rectClip>
<animateScroll xlink:href="#t1" direction="left" speed="2"
               delayAtStart="1" delayAtEnd="1" begin="3"
               repeatDur="indefinite"/>
```

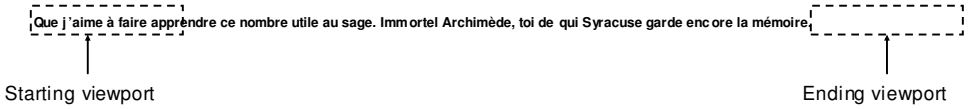


Figure 7: Start and end states of Example 1

Example 2: Start and end conditions

```
<lsr:rectClip id="t1" size="120 20">
  <text…>Que j'aime à faire apprendre ce nombre utile au sage. Immortel
Archimède, toi de qui Syracuse garde encore la mémoire.</text>
</lsr:rectClip>
<animateScroll xlink:href="#t1" direction="left" speed="2"
   delayAtStart="1" delayAtEnd="1" begin="3"
   repeatDur="indefinite" from="100 0" to="100 0"/>
```

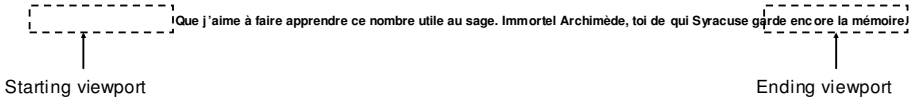


Figure 8: Start and end states of Example 2

The units of the from and to attributes are percents of the clip size. When the direction is “left”, from=”100 0” means that the object is initially 100% of clip with to the right of the clip, which means right outside of the clip. to=”100 0” means that the scrolled objects are, at the end of the animation, one clip width further to the left than the default leading edge of the clipping zone in the scrolling direction.

Example 3: Manual mode

```
<lsr:rectClip id="t1" size="120 20">
  <text…>Que j'aime à faire apprendre ce nombre utile au sage. <tspan
id="t2"/>Immortel Archimède, toi de qui Syracuse garde encore la mémoire.</text>
</lsr:rectClip>
<setScroll xlink:href="#t1" increment="100 0" begin="accessKey(RIGHT)"/>
<setScroll xlink:href="#t1" increment="-100 0" begin="accessKey(LEFT)"/>
<setScroll xlink:href="#t1" to="#t2" begin="accessKey(2)"/>
```

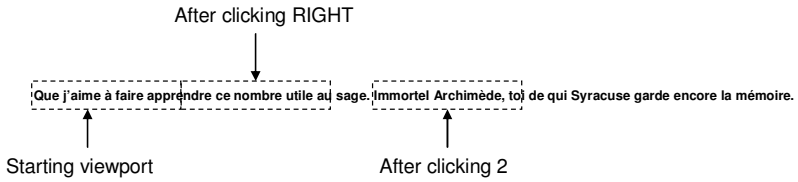


Figure 9: Start, interactive and end states of Example 3

Example 4: Mixed mode

```
<lsr:rectClip id="t1" size="120 20">
  <text…>Que j'aime à faire apprendre ce nombre utile au sage. <tspan
id="t2">Immortel Archimède,</tspan> toi de qui Syracuse garde encore la
mémoire.</text>
</lsr:rectClip>
<animateScroll xlink:href="#t1" begin="0;accessKey(0)" repeatDur="indefinite"
              direction="left" speed="3" delayAtStart="1" delayAtEnd="1"
```

```
                end="accessKey(2);accessKey(RIGHT);accessKey(LEFT)"/>
<setScroll xlink:href="#t1" increment="100 0" begin="accessKey(RIGHT)"/>
<setScroll xlink:href="#t1" increment="-100 0" begin="accessKey(LEFT)"/>
<setScroll xlink:href="#t1" to="#t2" begin="accessKey(2)"/>
```

The above sample has one scrolling to the left in 3 seconds per clip width with 1s waiting at each end, and looping indefinitely, unless the user presses a key. Pressing RIGHT or LEFT move the text by one page (one clip width) left or right. Pressing 2 goes to “Immortel”. Pressing 0 resumes the automatic scrolling.

6.8.41.2 Attributes

- xlink:href: this attribute specifies the ID of an lsr:rectClip element containing the scrolled objects. In the absence of this attribute, this element does not have any effect.
  Animatable: no. Inheritable: no.
- by: this attribute specifies the relative translation and consists in 2 values, expressed in percents of the clip size. The default value is 0 0. In case of repeated scrolling, this functions as an animateTransform with accumulate=”sum”.
  Animatable: no. Inheritable: no.
- from: this attribute specifies the starting position for the scrolling and consists in either 2 values, expressed in percents of the clip size, or in an element ID. The default value is 0 0, for the leading edge of the clipping zone in the scrolling direction. When from is an ID, it shall be the ID of a child of the scrolled object; the scrolling shall start with the leading edge of the object matching the leading edge of the clipping rectangle, according to the scrolling direction. from defines the minimum position of the clip.
  Animatable: no. Inheritable: no.
- to: this attribute specifies the end of scrolling position and consists in either 2 values, expressed in percents of the clip size, or in an element ID. The default value is 0 0, for the leading edge of the clipping zone in the scrolling direction. When to is an ID, it shall be the ID of a child of the scrolled object; the scrolling shall end with the trailing edge of the object matching the trailing edge of the clipping rectangle, according to the scrolling direction. to defines the maximum position of the clip.
  Animatable: no. Inheritable: no.
- delayAtStart: this attribute specifies the delay between the start of a cycle and the beginning of the movement. The default value is 0 for no delay.
  Animatable: no. Inheritable: no.
- delayAtEnd: this attribute specifies the delay between the end of the movement and the end of the cycle (and the restart of the cycle if the animation is looping). The default value is 0 for no delay.
  Animatable: no. Inheritable: no.
- direction: this enumeration specifies the scrolling direction: up, down, left, right. For example, “up” means that the automatic scrolling move the scrolled objects toward the negative Ys. The default value is left.
  Animatable: no. Inheritable: no.
- speed: this attribute expresses the time to scroll one page (or size of clipping zone) of text. It is exclusive from dur/end. The default value is 0 for no scrolling at all.
  Animatable: no. Inheritable: no.
- begin: this attribute specifies the beginning of the animation, in a similar manner as for the other animate* elements. The default value is indefinite.
  Animatable: no. Inheritable: no.
- dur: this attribute specifies the duration of the animation, including possible values of delayAtStart and delayAtEnd. The value of dur, if specified, shall be greater than delayAtStart+delayAtEnd. The default value is 0 for no scrolling.
  Animatable: no. Inheritable: no.
- end: the difference between the value of begin and end specifies the duration of the animation, including possible values of delayAtStart and delayAtEnd. The value of end, if specified, shall be greater than begin+delayAtStart+delayAtEnd. The default value is indefinite.
  Animatable: no. Inheritable: no.

- restart: this attribute specifies the restartability of the animation, in a similar manner as for the other animate* elements. The default value is always.
  Animatable: no. Inheritable: no.
- repeatCount: this attribute specifies the repeatability of the animation, in a similar manner as for the other animate* elements. The default value is 0 for no repetition.
  Animatable: no. Inheritable: no.
- repeatDur: this attribute specifies the repeatability of the animation, in a similar manner as for the other animate* elements. The default value is 0 for no repetition.
  Animatable: no. Inheritable: no.
- fill: this attribute specifies whether the text returns to its initial position before the animation (remove) or keeps its last position (freeze). The default value is freeze.
  Animatable: no. Inheritable: no.

6.8.42. SMIL2 animation

6.8.41.3 Semantics

The ***SMIL2 animation*** element is specified in section 7.3.1 of [W3C SMIL2]. Neither local IDs nor global IDs are shared between the main scene containing the animation element and the subscene referred by the animation element.

6.8.41.4 Attributes

- **clipBegin**: this attribute is defined in subclause 7.5.1 of [W3C SMIL2]. The value represents a normal play time. The value of clipBegin is taken into account when the media element is activated. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.
- **clipEnd**: this attribute is defined in subclause 7.5.1 of [W3C SMIL2]. The value represents a normal play time. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.

6.8.43. SVGT1.1 font

The ***SVGT1.1 font*** element is specified in section 20.3 of [W3C SVG11]

6.8.44. SVGT1.1 font-face

The ***SVGT1.1 font-face*** element is specified in section 20.8.3 of [W3C SVG11]

6.8.45. SVGT1.1 font-face-src

The ***SVGT1.1 font-face-src*** element is specified in section 20.8.3 of [W3C SVG11]

6.8.46. SVGT1.1 font-face-uri

The ***SVGT1.1 font-face-uri*** element is specified in section 20.8.3 of [W3C SVG11]

6.8.47. SVGT1.1 font-face-name

The ***SVGT1.1 font-face-name*** element is specified in section 20.8.3 of [W3C SVG11]

6.8.48. SVGT1.1 glyph

The ***SVGT1.1 glyph*** element is specified in section 20.4 of [W3C SVG11]

6.8.49. SVGT1.1 missing-glyph

The ***SVGT1.1 missing-glyph*** element is specified in section 20.5 of [W3C SVG11]

6.8.50. SVGT1.1 hkern

The ***SVGT1.1 hkern*** element is specified in section 20.7 of [W3C SVG11]

6.8.51. LASeR setScroll

6.8.51.68. Semantics

The setScroll element is a simplified animateScroll. Examples are provided in the animateScroll subclause.

6.8.51.69. Attributes

- xlink:href: this attribute specifies the ID of an lsr:rectClip element containing the scrolled objects. In the absence of this attribute, this element does not have any effect.
  Animatable: no. Inheritable: no.
- increment: this attribute specifies the relative translation and consists in 2 values, expressed in percents of the clip size. The scrolling increment is applied as many times as the element has been activated. The default value is 0 0.
  Animatable: no. Inheritable: no.
- to: this attribute specifies an absolute scrolling position and consists in either 2 values, expressed in percents of the clip size, or in an element ID. The default value is 0 0, for the leading edge of the clipping zone in the scrolling direction. When to is an ID, it shall be the ID of a child of the scrolled object; the scrolled objects shall be placed such that the leading edge of the scrolled objects match the leading edge of the clipping rectangle, according to the scrolling direction.
  Animatable: no. Inheritable: no.
- direction: this enumeration specifies the scrolling direction: up, down, left, right. For example, “up” means that the automatic scrolling moves the scrolled objects toward the negative Ys. The default value is left.
  Animatable: no. Inheritable: no.
- begin: this attribute specifies the beginning of the animation, in a similar manner as for the other animate* elements. The default value is indefinite.
  Animatable: no. Inheritable: no.

6.8.52. LASeR streamSource

6.8.52.70. Semantics

The purpose of streamSource is to first make available at the scene level information about the state of media chains (e.g. buffering, availability for rendering …) and secondly to give hints for potential resource optimization like, for instance, usage of a single hardware decoder, smooth transition during rendering without silence or black screens, ads display during channel switching.

The ‘streamSource’ element manages a set of streams, it does not actually have any display/rendering functionality. Instead, it keeps available an up-to-date pixel or audio buffer that can be used, and thus displayed by one or more video elements. As such, streamSource makes a pixel/audio buffer available to video/audio elements.

If any of the elements (in the source attribute) in the streamSource element is an audiovisual element containing both audio and video, two buffers (for audio and video respectively) shall be made available for all elements referring to the streamSource.

In such case,

- the pixel buffer associated to the streamSource when an audio only stream is active will represent a black buffer, the size of which will be defined by the width and height elements. Children of the streamSource element may define the default buffer in that case.
- the audio buffer associated to the streamSource when an video only stream is active will render as silence. Children of the streamSource element may define a default audio clip in that case.

The 'streamSource' element is able to trigger events such as the availability of the desired stream.

6.8.52.71. Attributes

- sources : is an array of references to stream sources
- sourceIndex : is the current active video source. On loading of the scene or when changing the sourceIndex, the LASeR engine shall connect to the session defined by the url corresponding to sourceIndex, this is source[i]
- width, height : default pixel buffer size
- mode : is one of "replace", "useOld", "keepOld", "playList".
    - The value "replace" indicates that the composition buffer associated to the stream should be immediately replaced when sourceIndex is changed.
    - The value "useOld" means that the previous stream will be decoded till the stream pointed by sourceIndex is ready which means that data will start to be available for composition for the stream pointed by sourceIndex. This implies that decoding and rendering of the previous stream will occur up to the moment at which the first composition buffer is made available.
    - With "keepOld" the same behaviour as from "useOld" is expected, furthermore the previous video stream source is kept open.
    - The value "playList" indicates that the different sources will be played in sequence. In this case, it is possible to append a "repeat" and/or "shuffle" modes such as "playList, repeat, shuffle". This modes repeat and shuffle allow respectively to cycle through sources and to shuffle the sources.

6.8.53. LASeR updates

6.8.53.72. Semantics

The LASeR updates element provides a link to a source of scene updates. The value of the xlink:href attribute defines the location of the updates to be applied to the current scene. The updates element is a timed element providing stream playback control.

6.8.53.73. Attributes

All SMIL timing attributes defined in [W3C SVGT12] section 16.2.7 are defined for this element, except the "fill" attribute.

- **clipBegin**: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The value of clipBegin is taken into account when the media element is activated. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.
- **clipEnd**: this attribute is defined in subclause 7.5.1 of [SMIL2]. The value represents a normal play time. The play time of some streams cannot be controlled, and under these circumstances, this attribute has no effect. This attribute is not animatable and not inheritable.
- **xlink:href**: this attribute specifies the location of the stream of updates. In the absence of this attribute, this element does not have any effect. This attribute is not animatable and not inheritable.

- **security:** this attribute controls the type of allowed updates in the referenced stream. The referenced stream may contain a complete scene (for example, LASeR starting with a NewScene command) or a scene segment (for example, LASeR starting with any other command than NewScene or RefreshScene). Allowed values for the security attribute are *any, complete* or *segment.* The value *any* does not restrict the type of updates in the referenced stream: this is the default value. The value *complete* restricts the updates in the referenced stream to be a complete scene: the current scene may be discarded upon activation of the hyperlink. The value *segment* restricts the updates in the referenced stream to be a scene segment: any command replacing or refreshing the entire scene (for example LASeR NewScene/RefreshScene) shall be ignored. This attribute is not animatable and not inheritable.
- **syncReference**: this attribute holds a reference to the stream whose clock acts as a clock reference for the stream referred to by this element. This attribute is not animatable and not inheritable.
- **flow:** this attribute controls the behavior of the referenced stream of updates. Allowed values are *parent* and *new*. The name of the attribute refers to the "flow of information", which comprises incoming commands and media. The value *parent* means that the referenced stream shall replace the flow of information from which this a element originated. The value *new* means that the referenced stream shall be processed in parallel with existing flows of information. Note – the LASeR engine may manage transport connections according to this information. This attribute is not animatable and not inheritable.

6.9. Summary of Possible Attributes per Element

Note on Table 7: In LASeR, to simplify the decoding, all elements have the same content model in the binary format. This binary content model allows all elements as well as extensions and private data. However, failure to comply with the SVG content model will result in the document being in error, as defined in [W3C SVG11]. Any extra attribute in Table 7, not defined in SVG or in this specification, is a place holder for future extension. This table can be regenerated from the validation schema which is normative.

Table 7 — Summary of Possible Children and Attributes per Element

Element name	Attributes
a	audio-level color color-rendering display display-align externalResourcesRequired fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage target text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type
animate	accumulate additive attributeName begin by calcMode class dur enabled end fill from id keySplines keyTimes max min repeatCount repeatDur restart to values xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space
animateColor	accumulate additive attributeName begin by calcMode class dur enabled end fill from id keySplines keyTimes max min repeatCount repeatDur restart to values xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space
animateMotion	accumulate additive attributeName begin by calcMode class dur enabled end fill from id keyPoints keySplines keyTimes max min path repeatCount repeatDur restart rotate to values xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space
lsr:animateScroll	id class xml:base xml:lang xml:space xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show by from to delayAtStart delayAtEnd speed direction begin dur end fill restart repeatCount repeatDur

Element name	Attributes
animateTransform	accumulate additive attributeName begin by calcMode class dur enabled end fill from id keySplines keyTimes max min repeatCount repeatDur restart to type values xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space
animation	id class xml:base xml:lang xml:space requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts audio-level display image-rendering pointer-events shape-rendering text-rendering viewport-fill viewport-fill-opacity visibility lsr:rotation lsr:scale lsr:translation transform xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable fill focushiglight width height x y externalResourcesRequired begin end dur min max restart repeatCount repeatDur syncBehavior syncTolerance syncMaster pereserveAspectRatio type lsr:syncReference lsr:clipBegin lsr:clipEnd initialVisibility
audio	audio-level begin class dur end externalResourcesRequired id lsr:syncReference repeatCount repeatDur requiredExtensions requiredFeatures requiredFormats syncBehavior syncTolerance systemLanguage type xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space type
circle	audio-level class color color-rendering cx cy display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-weight font-variant id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events r requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
lsr:conditional	id class begin enabled externalResourcesRequired xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show xml:base xml:lang xml:space
lsr:cursorManager	id class xml:base xml:lang xml:space xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show x y
defs	audio-level class color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style font-variant font-weight id image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
desc	class id xml:base xml:lang xml:space
ellipse	audio-level class color color-rendering cx cy display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats rx ry shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
font	id class xml:base xml:lang xml:space externalResourcesRequired horiz-adv-x horiz-origin-x
font-face	id class xml:base xml:lang xml:space accent-height alphabetic ascent bbox cap-height descent externalResourcesRequired font-family font-stretch font-style font-variant font-weight hanging ideographic mathematical overline-position overline-thickness panose-1 slope stemh stemv strikethrough-position strikethrough-thickness underline-position underline-thickness unicode-range units-per-em widths x-height
font-face-src	id class xml:base xml:lang xml:space
font-face-uri	id class xml:base xml:lang xml:space externalResourcesRequired xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show

Element name	Attributes
foreignObject	audio-level class color color-rendering display display-align externalResourcesRequired fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight height id image-rendering line-increment pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility width x xml:base xml:lang xml:space y
g	id class xml:base xml:lang xml:space audio-level color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style text-decoration font-weight font-variant image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering viewport-fill viewport-fill-opacity vector-effect visibility lsr:rotation lsr:scale lsr:translation transform requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable focusHighlight externalResourcesRequired
glyph	id class xml:base xml:lang xml:space arabic-form d glyph-name horiz-adv-x lang unicode externalResourceRequired xlink:href xlink:title xlink:type xlink:role xlink:arcrole xlink:actuate xlink:show
g	id class xml:base xml:lang xml:space audio-level color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style text-decoration font-weight font-variant image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering viewport-fill viewport-fill-opacity vector-effect visibility lsr:rotation lsr:scale lsr:translation transform requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable focusHighlight externalResourcesRequired
hkern	id class xml:base xml:lang xml:space g1 g2 k u1 u2
image	class display externalResourcesRequired nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable height id lsr:rotation lsr:scale lsr:translation opacity pointer-events requiredExtensions requiredFeatures requiredFormats systemLanguage transform transformBehavior type visibility width x xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space y type preserveAspectRatio viewport-fill viewport-fill-opacity
line	audio-level class color color-rendering display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility x1 x2 xml:base xml:lang xml:space y1 y2
linearGradient	audio-level class color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style font-variant font-weight gradient-units id image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility x1 x2 xml:base xml:lang xml:space y1 y2
ev:listener	id enabled event handler observer phase propagate defaultAction target
metadata	class id xml:base xml:lang xml:space
missing-glyph	id class xml:base xml:lang xml:space d horiz-adv x
mpath	class id xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space

Element name	Attributes
path	audio-level class color color-rendering d display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pathLength pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
polygon	audio-level class color color-rendering display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events points requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
polyline	audio-level class color color-rendering display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events points requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
radialGradient	audio-level class color color-rendering cx cy display display-align fill fill-opacity fill-rule font-family font-size font-style font-variant font-weight gradient-units id image-rendering line-increment pointer-events r shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
rect	audio-level class color color-rendering display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight height id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats rx ry shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility width x xml:base xml:lang xml:space y
lsr:rectClip	id class xml:base xml:lang xml:space audio-level color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style text-decoration font-weight font-variant image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering viewport-fill viewport-fill-opacity vector-effect visibility lsr:rotation lsr:scale lsr:translation transform requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable focusHighlight externalResourcesRequired size width height x y
script	begin class enabled externalResourcesRequired id type xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space type

Element name	Attributes
lsr:selector	id class xml:base xml:lang xml:space audio-level color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style text-decoration font-weight font-variant image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering viewport-fill viewport-fill-opacity vector-effect visibility lsr:rotation lsr:scale lsr:translation transform requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable focusHighlight externalResourcesRequired choice
set	attributeName begin class dur enabled end fill id max min repeatCount repeatDur restart to xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space
lsr:setScroll	id class xml:base xml:lang xml:space begin increment to direction xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type
lsr:simpleLayout	id class xml:base xml:lang xml:space audio-level color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style text-decoration font-weight font-variant image-rendering line-increment pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering viewport-fill viewport-fill-opacity vector-effect visibility lsr:rotation lsr:scale lsr:translation transform requiredFeatures requiredExtensions systemLanguage requiredFormats requiredFonts nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable focusHighlight externalResourcesRequired size
stop	audio-level class color color-rendering display display-align fill fill-opacity fill-rule font-family font-size font-style font-variant font-weight id image-rendering line-increment offset pointer-events shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
streamSource	id class xml:base xml:lang xml:space sources sourceIndex width height mode
svg	audio-level baseProfile class color color-rendering contentScriptType display display-align externalResourcesRequired fill fill-opacity fill-rule font-family font-size font-style font-variant font-weight height id image-rendering line-increment playbackOrder pointer-events preserveAspectRatio shape-rendering snapshotTime solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width syncBehaviorDefault syncToleranceDefault text-anchor text-rendering timeLineBegin vector-effect version viewBox viewport-fill viewport-fill-opacity visibility width xml:base xml:lang xml:space zoomAndPan
switch	audio-level class color color-rendering display display-align externalResourcesRequired fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
text	audio-level color color-rendering display display-align editable fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight image-rendering line-increment lsr:rotation lsr:scale lsr:translation pointer-events requiredExtensions requiredFeatures requiredFormats rotate shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility x y

Element name	Attributes
title	class id xml:base xml:lang xml:space
tspan	audio-level class color color-rendering display display-align fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering vector-effect viewport-fill viewport-fill-opacity visibility xml:base xml:lang xml:space
use	audio-level class color color-rendering display display-align externalResourcesRequired fill fill-opacity fill-rule nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable font-family font-size font-style font-variant font-weight id image-rendering line-increment lsr:rotation lsr:scale lsr:translation overflow pointer-events requiredExtensions requiredFeatures requiredFormats shape-rendering solid-color solid-opacity stop-color stop-opacity stroke stroke-dasharray stroke-dashoffset stroke-linecap stroke-linejoin stroke-miterlimit stroke-opacity stroke-width systemLanguage text-anchor text-rendering transform vector-effect viewport-fill viewport-fill-opacity visibility x xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space y
updates	externalResourcesRequired requiredExtensions requiredFeatures requiredFormats systemLanguage xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type begin class dur end id lsr:syncReference repeatCount repeatDur syncBehavior syncTolerance type xml:base xml:lang xml:space clipBegin clipEnd security flow
video	audio-level begin display dur end externalResourcesRequired fullscreen nav-right nav-next nav-up nav-up-right nav-up-left nav-prev nav-down nav-down-right nav-down-left nav-left focusable height lsr:rotation lsr:scale lsr:syncReference lsr:translation overlay pointer-events repeatCount repeatDur requiredExtensions requiredFeatures requiredFormats syncBehavior syncTolerance systemLanguage transform transformBehavior type visibility width x xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type y type

6.10. Additions to the uDOM API

6.10.1. Parsing of an encoded XML element

6.10.1.74. Syntax

```
Node parseLASeRBinaryElement(in sequence<octet> data, in Document contextDoc);
```

6.10.1.75. Semantics

Given a chunk of binary data and a Document object, parse the binary data as a LASeR element and return a Node representing it. If the binary data is not well-formed, this method must return a null value."

The format of the sequence<octet> data parameter is: one LASeRHeader followed by a byte-aligned encoded object of class updatable_elements as defined in clause 12.

6.10.2. Trait access extensions

The following table adds to the table in A.8.12 of [SVGT1.2]. It contains trait access rules for DIMS extensions.

Attribute	Trait Getter	Trait Setter	Def.Val.	Description
fullscreen	getTraitNS[true | false]	setTraitNS[true | false]	false	Available on <video> and <svg> elements
x	getFloatTraitNS	setFloatTraitNS	0.0f	Origin x of the <rectClip>

y	getFloatTraitNS	setFloatTraitNS	0.0f	Origin y of the <rectClip>
width	getFloatTraitNS	setFloatTraitNS	0.0f	Width of the clipping region defined by <rectClip>
height	getFloatTraitNS	setFloatTraitNS	0.0f	Height of the clipping region defined by <rectClip>

6.10.3. Description of getFloatTraitNS and setFloatTraitNS methods

float getFloatTraitNS(in DOMString namespaceURI, in DOMString name) raises(DOMException);

- Same as getFloatTrait, but for namespaced traits. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters:

`namespaceURI` - the namespaceURI of the trait to retrieve.
`name` - the name of the trait to retrieve.

Return Value:

the trait value as float.

Exceptions:

`DOMException` - with error code NOT_SUPPORTED_ERR if the requested trait is not supported on this element or null.
`DOMException` - with error code TYPE_MISMATCH_ERR if requested trait's computed value cannot be converted to a float.

void setFloatTraitNS(in DOMString namespaceURI, in DOMString name, in float value) raises(DOMException);

Same as setFloatTrait, but for namespaced traits. Parameter name must be a non-qualified trait name, i.e. without prefix.

Parameters:

`namespaceURI` - the namespaceURI of the trait to be set.
`name` - the name of the trait to be set.
`value` - the value of the trait to be set as float.

Exceptions:

`DOMException` - with error code NOT_SUPPORTED_ERR if the requested trait is not supported on this element or null.
`DOMException` - with error code TYPE_MISMATCH_ERR if the requested trait's value cannot be specified as a float (for e.g. NaN)

`DOMException` - with error code INVALID_ACCESS_ERR if the input value is an invalid value for the given trait or null.

7. Simple Aggregation Format (SAF)

7.1. Overview

The Simple Aggregation Format (SAF) defines the binary representation of a compound data stream composed of different data elementary streams (ES) such as LASeR scene description, video, audio, image, font, metadata streams. Data from these various data elementary streams results in one SAF stream by multiplexing them for simple, efficient and synchronous delivery.

To efficiently carry elementary data streams synchronously as one logical SAF stream, a basic entity to be carried is defined as a SAF Access Unit (SAF AU), encapsulated into a basic entity for synchronization defined as a SAF Packet, (SAF packet).

An XML syntax (normative) providing a readable representation of the SAF Binary Syntax (normative) is defined and the schema for this syntax is provided in electronic attachement. The top element, SAFSession, is only existing to provide an XML root element, and does not have a binary equivalent. A SAF session is the logical entity grouping elementary streams used in the presentation.

7.2. Time and terminal model specification

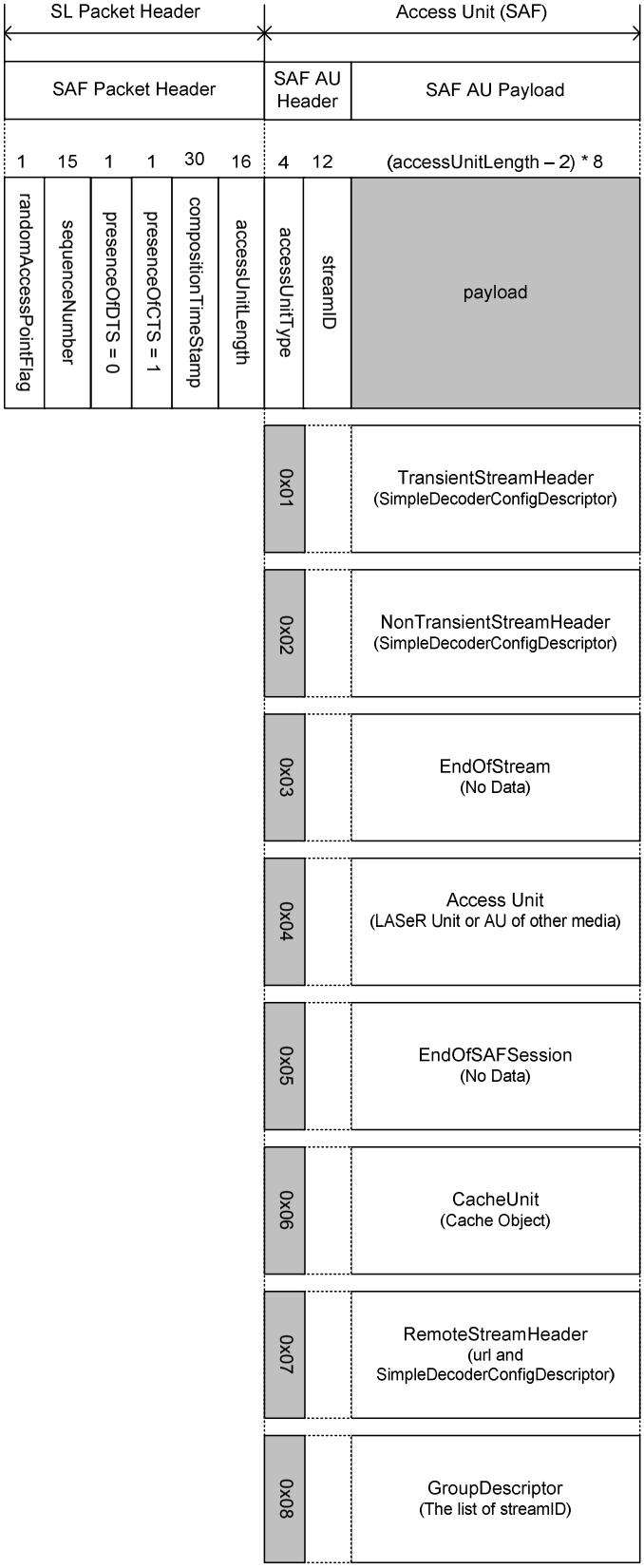
The timing model relies on clock references and time stamps to synchronize audio-visual data conveyed by SAF streams. The concept of a clock is used to convey the notion of time to a receiving terminal. Time stamps are used to indicate the precise time instants at which the receiving terminal decodes the SAF Packet.

Each SAF Packet has an associated nominal composition time, the time at which it must be available for composition. The decoded data contained in a SAF Packet is not guaranteed to be available for composition before this time. Some SAF Packets may have a composition time stamp set to 0; in that case the SAF Packet is decoded and executed as soon as it is received. Otherwise the SAF Packets are decoded and executed at their nominal composition time and in the receiving order. When a SAF Packet is received “late” according to the scene time, the SAF Packet is processed as soon as possible.

7.3. SAF Packet

The SAF Packet consists of a SAF packet header and a SAF packet payload. The SAF packet header carries the coded representation of the time stamps and associated information.

Here is a presentation of the architecture of the SAF Packet in



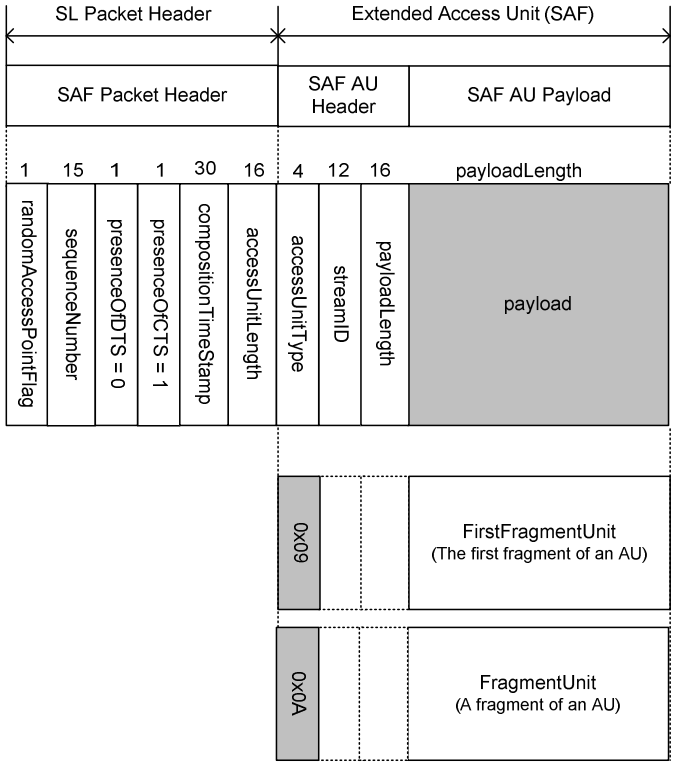
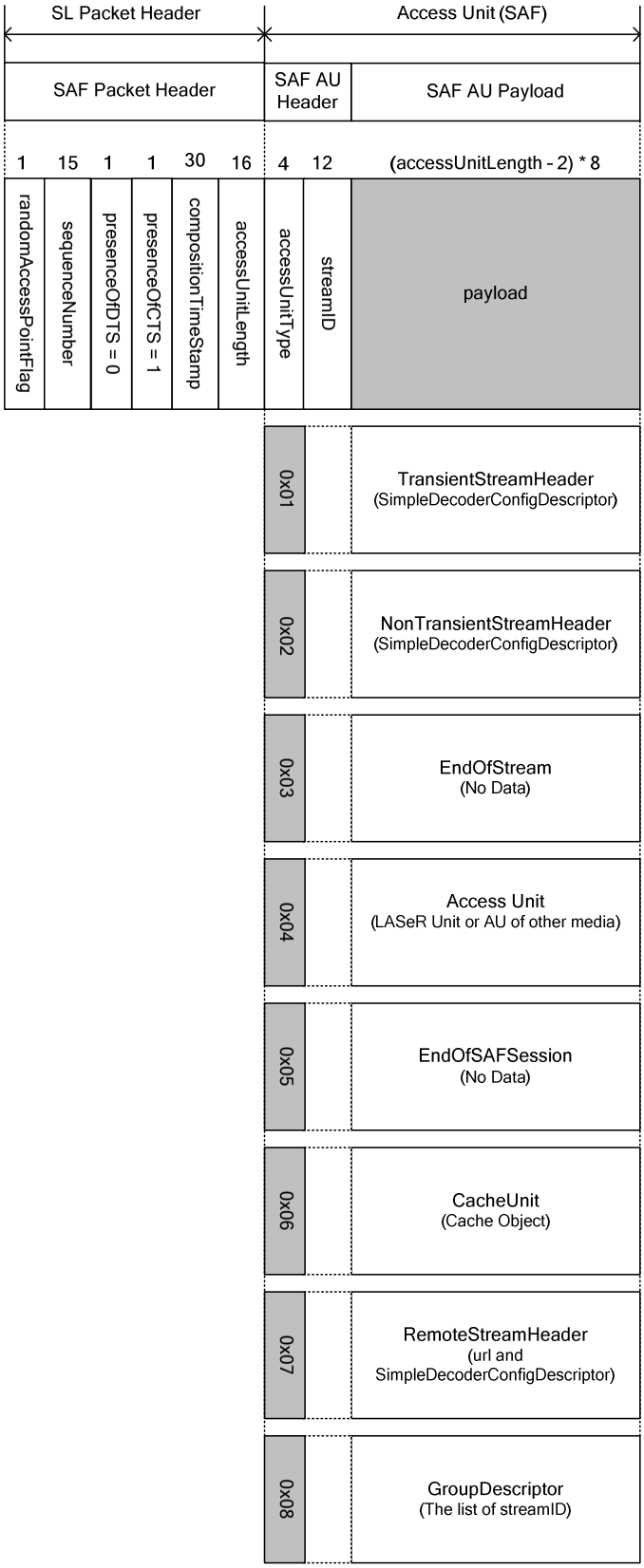


Figure 10



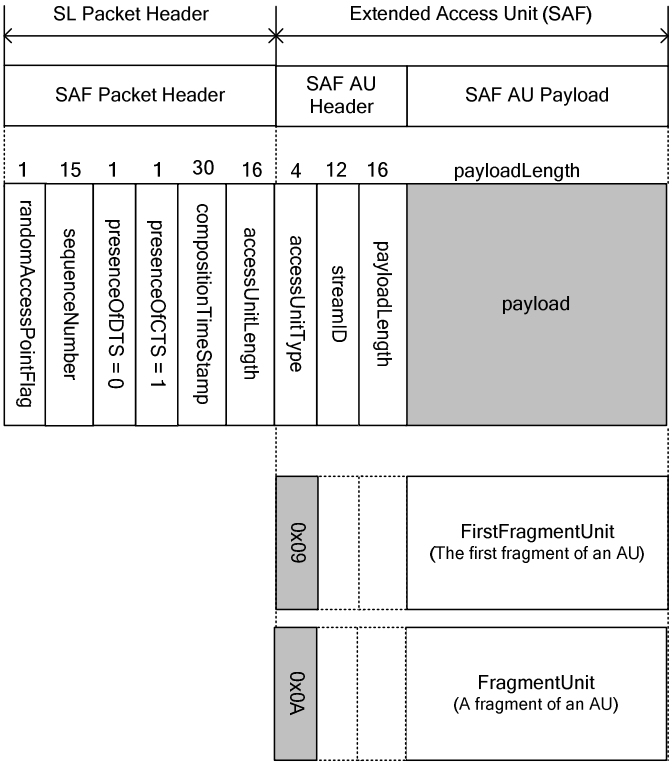


Figure 10 — SAF Packet architecture

7.3.1. Syntax

```
class SAF_Packet {
    SAF_PacketHeader packetHeader;
    byte[packetHeader.accessUnitLength] packetPayload;
}
```

7.3.2. Semantics

`packetHeader` – a `SAF_PacketHeader` element as specified in 7.4.

`packetPayload` – a payload that contains an opaque payload at this level of the specification.

7.4. SAF Packet Header

7.4.1. Syntax

```
class SAF_PacketHeader {
    bit(1) randomAccessPointFlag;
    bit(15) sequenceNumber;
    const bit(1) presenceOfDTS = 0;
    const bit(1) presenceOfCTS = 1;
    bit(30) compositionTimeStamp;
    uint(16) accessUnitLength;
}
```

7.4.2. Semantics

`sequenceNumber` – if present, successive packets shall either have the same sequence number or the value be continuously incremented as a modulo counter. Note – when two packets have the same sequence number, one can be ignored. Sequence numbers are incremented per SAF packet.

`randomAccessPointFlag` – when set to one indicates that random access to the content of this elementary stream is possible here. For SAF packets of type TransientStreamHeader, NonTransientStreamHeader or RemoteStreamHeader, the value of this flag shall be ignored.

`compositionTimeStamp` – is a composition time stamp.

For SAF packets of type AccessUnit, the composition time tc of the first composition unit resulting from this packet is reconstructed from this composition time stamp according to the formula:

$$tc = (compositionTimeStamp/timeStampResolution + k * 2^{30}/timeStampResolution)$$

where k is the number of times that the `compositionTimeStamp` counter has wrapped around. The value of this field from two different packets may be same, if their `streamIDs` are different.

For streams which would require a different decoding time stamp and composition time stamp, such as MPEG-4 Part 10 (AVC) streams, the packets shall be provided in decoding order with composition time stamps, which means that time stamps may not be monotonically growing.

For SAF packets of type cacheUnit, this field specifies the validity period in second. In this case, the value 0 is reserved and the value 0x3FFFFFFF means that the Cache Unit never expires.

For SAF packets of type TransientStreamHeader, NonTransientStreamHeader or RemoteStreamHeader, this field specifies when the packet shall be processed.

When TransientStreamHeader, NonTransientStreamHeader or RemoteStreamHeader using a stream ID already in use are received, they shall be ignored by the terminal.

`accessUnitLength` – is the length in bytes of the SAF access unit conveyed in the SAF packet. The value of this field shall be at least 2. Value 1 is reserved for future ISO use. Value 0 signals that the payload of this SAF Packet is a SAF extended AU, as specified in clause 7.6.

7.5. SAF Access Unit

A SAF Access Unit consists of a two-byte header (SAF Header) and a byte-aligned payload (SAF Payload).

7.5.1. Syntax

```
class safAU {
  bit(4) accessUnitType;
  bit(12) streamID;
  byte(8)[packetHeader.accessUnitLength-2] payload;
}
```

7.5.2. Semantics

`accessUnitType` – an indication about the type of the payload. Detailed values of accessUnitType and the data corresponding to each type are defined in Table 8

Table 8 — accessUnitType values and corresponding data in the payload

Value	Type of access unit payload	Data in payload
0x00	Reserved	-
0x01	TransientStreamHeader	A SimpleDecoderConfigDescriptor
0x02	NonTransientStreamHeader	A SimpleDecoderConfigDescriptor
0x03	EndofStream	(no data)
0x04	AccessUnit	An Access Unit
0x05	EndOfSAFSession	(no data)
0x06	CacheUnit	A cache object
0x07	RemoteStreamHeader	An url and a SimpleDecoderConfigDescriptor
0x08	GroupDescriptor	-
0x09	FirstFragmentUnit	The first Fragment of an Access Unit
0x0A	FragmentUnit	A Fragment of an Access Unit (not the first fragment)
0x0B	SAFConfiguration	A safConfiguration object
0x0C	StopCache	-
0x0B ~ 0x0F	Reserved	-

`streamID` – the reference of the media stream this AU belongs to. This identifier is local to the current SAF session.

`payload` – the data part of the access unit. The size of the payload is signalled by the accessUnitLength field in the packet header as specified in 7.4. The type of data in this field is varied by accessUnitType as defined in Table 8.

For values of `safAU.accessUnitType` that refer to TransientStreamHeader and NonTransientStreamHeader, the `payload` shall convey a SimpleDecoderConfigDescriptor whose syntax and semantics is specified in 7.6.

A TransientStreamHeader shall be used to indicate that the stream shall be used once and by only one media element. Continuous media should be marked as transient.

NOTE – Implementations may optimize the resources for a stream using TransientStreamHeader as soon as no more media element references the stream.

For values of `safAU.accessUnitType` that refer to LASeR access unit or cache unit, the `payload` shall convey a access unit or cache unit which syntax and semantics are specified in this clause.

For values of `safAU.accessUnitType` that refer to an access unit, the `payload` shall convey an access unit for specific media whose syntax and semantics is opaque to this standard.

7.6. SimpleDecoderConfigDescriptor

7.6.1. Syntax

```
class SimpleDecoderConfigDescriptor {
    bit(8) objectTypeIndication;
    bit(8) streamType;
    bit(24) timeStampResolution;
    bit(16) bufferSizeDB;
    if (streamType == 0xFF && objectTypeIndication == 0xFF) {
      bit(16) mimeTypeLength;
      byte mimeType[mimeTypeLength];
    }
    SimpleDecoderSpecificInfo decSpecificInfo[0 .. 1];
```

```
}
```

7.6.2. Semantics

The `SimpleDecoderConfigDescriptor` provides information about the decoder type and the required decoder resources needed for the associated media stream. This is needed at the receiving terminal to determine whether it is able to decode the media stream. A stream type identifies the category of the stream while the optional decoder specific information descriptor contains stream specific information for the set up of the decoder in a stream specific format that is opaque to this layer.

`objectTypeIndication` – an indication of the object or scene description type that needs to be supported by the decoder for this elementary stream as per the table on objectTypeIndication of ISO/IEC 14496-1. ISO-defined as well as registered object type indications can be found at www.mp4ra.org, web site of the MPEG-4 Registration Authority.

`streamType` – conveys the type of this elementary stream as per the streamType table of ISO/IEC 14496-1.

`bufferSizeDB` – is the size of the decoding buffer for this media stream in byte.

`decSpecificInfo[]` – an array of zero or one decoder specific information classes as specified in subclause 7.7.

`mimeTypeLength` – an unsigned integer indicating the size of mimeType in bytes.

`mimeType` – conveys the MIME Type of the stream as defined in [RFC 2045] encoded in UTF-8.

`timeStampResolution` – is the resolution of the time stamps in clock ticks per second.

7.7. SimpleDecoderSpecificInfo

The decoder specific information constitutes an opaque container with information for a specific media decoder. The existence and semantics of decoder specific information depends on the values of `streamType` and `objectTypeIndication`.

For values of `objectTypeIndication` that refer to streams complying with media standard the syntax and semantics of decoder specific information is defined in each standard.

For values of `objectTypeIndication` that refer to streams complying with LASeR scene description, the decoder specific information shall carry a LASeR header as defined in 6.5.

7.8. RemoteStreamHeader

7.8.1. Syntax

```
class RemoteStreamHeader {
    bit(8) objectTypeIndication;
    bit(8) streamType;
    bit(24) timeStampResolution;
    bit(16) bufferSizeDB;
    if (streamType == 0xFF && objectTypeIndication == 0xFF) {
      bit(16) mimeTypeLength;
      byte mimeType[mimeTypeLength];
    }
    bit(16) urlLength;
```

```
    byte url[urlLength];
    SimpleDecoderSpecificInfo decSpecificInfo[0 .. 1];
}
```

7.8.2. Semantics

The `RemoteStreamHeader` appears as payload of all SAF Access Units whose `safAU.accessUnitType` value is `0x07`. The `RemoteStreamHeader` is a simple extension of the `SimpleDecoderConfigDescriptor`, with the addition of an url.

`objectTypeIndication` – an indication of the object or scene description type that needs to be supported by the decoder for this elementary stream as per the table on objectTypeIndication of ISO/IEC 14496-1.

`streamType` – conveys the type of this elementary stream as per the streamType table of ISO/IEC 14496-1.

`bufferSizeDB` – is the size of the decoding buffer for this media stream in byte.

`urlLength` – is the size of the url in byte.

`url` – is a UTF-8 string carrying the url of the media access units.

`decSpecificInfo[]` – an array of zero or one decoder specific information classes as specified in subclause 7.7.

`mimeTypeLength` – an unsigned integer indicating the size of mimeType in bytes.

`mimeType` – conveys the MIME Type of the stream as defined in [RFC 2045] encoded in UTF-8.

`timeStampResolution` – is the resolution of the time stamps in clock ticks per second.

7.9. Cache Unit

A cache object is the payload of a cacheUnit packet and conveys a url and data. If a terminal requests a url, and a cache unit matching the requested url is already present in the terminal, then the terminal may directly load the corresponding data, without requesting the data referred to by this url from the server. A cache unit can be permanent and stored in memory as soon as it is retrieved. A cache object is not expired during the period defined by its receiving time and its receiving time plus the time specified in the `compositionTimeStamp` field of packet header expressed in seconds. After the end time, the cache object is expired and its content cannot be executed. The streamID field of the SAF Packet is ignored for a cacheUnit.

7.9.1. Syntax

```
class cacheUnit {
  bit(1) replace;
  bit(1) permanent;
  bit(6) reserved = 0;
  unit(16) urlLength;
  byte(urlLength) url;
  byte[packetHeader.accessUnitLength-urlLength-5] payload; // 5 is 3 bytes above
and 2 in the AU header
}
```

7.9.2. Semantics

`replace` – if true, this cacheUnit replaces any previous cacheUnit for the same url; if false, this cacheUnit is appended to any previous cacheUnit with the same url.

`permanent` – if true, the cacheUnit shall be kept, if the terminal has enough resources, after the end of the application for a duration stored in the `compositionTimeStamp` of this SAF Packet.

`urlLength` – an unsigned integer indicating the size of url in bytes.

`url` – the url of the presentation conveyed in a `payload`. This url shall not include a protocol. Example: "www.acme.com/service/bar.jpg"

`payload` – the data, in a format opaque to this specification. The size of this field is signaled by the `accessUnitLength` field in SL packet header.

7.10. EndOfStream

The endOfStream unit signals the end of an elementary stream defined in this SAF session. Once this unit is received, all further SAF Packets for this stream shall be ignored.

An elementary stream is defined in the SAF session upon reception of a StreamHeader or a RemoteStreamHeader unit, and until an endOfStream unit is received.

7.11. EndOfSAFSession

The endOfSAFSession unit signals the end of the SAF session. Once this unit is received, all SAF Packets for this session shall be ignored. When a SAFSession is closed (either by *EndOfSAFSession*, of by other means), all unused media stream originated from this SAF session may be discarded.

We call an unused media stream a stream that has no elements in the scene tree that reference the streamID of this media stream. For efficiency the content of <lsr:conditional> shall not be taken into account regarding streamID references.

When a Saf Session is closed (either by *EndOfSAFSession*, or by other means), all streams that have a String streamID shall not be discarded.

7.12. SAF Extended Access Unit

7.12.1. Syntax

```
class safXAU {
  bit(4) accessUnitType;
  bit(12) streamID;
  bit(16) payloadLength;
  byte(8)[payloadLength] payload;
}
```

7.12.2. Semantics

The value `accessUnitLength` in SAF Packets containing Extended Access Units shall be 0.

`accessUnitType` – the semantic is the same as in clause 7.5.2.

`streamID` – the semantic is the same as in clause 7.5.2.

payload – the data part of the access unit. The size of the payload is signalled in payloadLength. The type of data in this field is varied by accessUnitType as defined in Table 8 and in clause 7.5.2.

7.13. GroupingDescriptor

7.13.1. Syntax

```
class groupingDescriptor {
    bit(8) number_of_elements;
    for (int i=0 ; i<number_of_element ; i++) {
        bit(12) streamID;
     }
}
```

7.13.2. Semantics

number_of_elements – number of elements in this group

streamID – a streamID of elementary stream grouped into a single decoder

This descriptor signals grouping of elementary streams to use more than one decoder buffers for a single decoder to improve switching behaviours between the elementary streams in the same group. The terminal shall continuously receive elementary streams referred by this descriptor and fill the decoding buffers. No more than one stream referred by this descriptor shall be decoded at the same time. This may require allocation of more than one decoding buffer for a single decoder. Note: When this descriptor is used, even though the decoder buffers are not connected to the decoder, the AU is removed from the decoder buffer when the DTS is arrived.

7.14. SAF Fragment Unit

The fragment unit is a specialized SAF Extended Access Unit. accessUnitLength of this packet shall be zero.

7.14.1. Syntax

```
class safFU {
  bit(4) accessUnitType;
  bit(12) streamID;
  bit(16) payloadLength;
  bit(8) fragmentSeqNum;
  byte(8)[payloadLength-1] payload;
}
```

7.14.2. Semantics

accessUnitType – the value of this field shall be 10

streamID – same as in 7.5.2.

payloadLength – the length of the data part of the access unit.

fragmentSeqNum – the fragment sequence number on 8 bits, possibly wrapping.

payload – the data part of the access unit. The size of the payload is signalled by the payloadLength field in the packet header as specified in 7.4.

7.15. SAF First Fragment Unit

The first fragment unit is a specialized SAF Extended Access Unit. accessUnitLength of this packet must be zero.

7.15.1. Syntax

```
class safFFU {
  bit(4) accessUnitType;
  bit(12) streamID;
  bit(16) payloadLength;
  bit(4) carriedAccessUnitType;
  bit(4) reserved;
  bit(32) totalLengthOfAccessUnit;
  byte(8)[payloadLength-5] payload;
}
Semantics
```

`accessUnitType` – the value of this field shall be 9

`streamID` – the reference of the media stream this AU belongs to.

`carriedAccessUnitType` – an indication about the type of the access unit whose fragment is carried in the payload. Detailed values of `accessUnitType` and the data corresponding to each type are defined in **Erreur ! Source du renvoi introuvable.**

`totalLengthOfAccessUnit` – the length of the access unit, which shall be equal to the accumulated length of the fragments.

`payload` – the data part of the access unit. The size of the payload is signalled by the `payloadLength` field in the packet header as specified in 7.4.

7.16. SAF Configuration

7.16.1. Syntax

```
class safConfiguration {
  bit(1) add;
  bit(1) cacheThisSession;
  bit(1) hasMainStreamID;
  bit(1) hasStreamDependencies;
  if (add == 0 && hasMainStreamID == 1) {
      bit(12) mainStreamID;
  } else {
    bit(4) reserved = 0;
  }
  if (hasStreamDependencies) {
    uint(16) sd_count;
    for (int i = 0; i < sd_count; i++) {
          bit(12) streamID;
          bit(12) dependsOnStreamID;
    }
  }
  // globalStreamTable: external or internal global streamID references
  uint(8) count;
  for (int i = 0; i < count; i++) {
    bit(12) localStreamIdForThisGlobal[[i]];
      bit(12) globalNameLength;
```

```
    byte[globalNameLength] globalName[[i]];
  }
  if (cacheThisSession == 1) {
    bit(1) permanent;
    bit(31) validity;
    unit(16) urlLength;
    byte(urlLength) cacheURL;
  }
  byte[accessUnitLength - sizeof(previous elements)] extensionData;
}
```

7.16.2. Semantics

add – if set to 1, this packet defines an addition to the previous SAF Configuration. If set to 0, the current global stream table is reset and the global stream table defined in this packet becomes the current global stream table. Existing stream dependencies shall not be re-assigned, either when appending a new configuration or when redefining it.

cacheThisSession – if set, the current session should be cached. If the connection is closed before the arrival of a stopCache instruction, nothing is stored in cache.

permanent – if true, the cached session shall be kept, if the terminal has enough resources, after the end of the application for a duration stored in the validity field.

validity – the validity period of this cached session expressed in seconds

urlLength – if non-zero, the cacheURL field contains a string of length urlLengthwhich is the URL at which the current session should be cached. If zero, the original URL of the current session should be used for caching. Note: the presence of a URL should be the exception.

cacheURL – the URL to be used for the caching of this session.

mainStreamID – in cases where there are multiple scene streams, the mainstreamed allows the author to specify which is the main stream. In the absence of a SAF Configuration packet, the main stream ID is 0.

streamID – stream ID for which a coding dependency is being specified.

dependsOnStreamID –ID of the stream upon which the specified stream is dependent.

localStreamIdForThisGlobal – defines the number used as stream ID for the stream whose name is globalName.

globalName – defines the name of the global stream, which is the identifier used for reference across SAF session boundaries.

7.17. Stop Cache

7.17.1. Syntax

```
class stopCache {
}
```

7.17.2. Semantics

When this packet arrives, any caching of the current session ends.

8. Profiles

8.1. Overview

The LASeR mini and full profiles are defined in the SceneGraph dimension of MPEG-4 Systems profiles. The purpose of LASeR mini is to allow very constrained implementations of LASeR, including J2ME implementations. One specific feature of mini is that the implementation does not need to implement 2 trees, one DOM tree and one composition tree, or otherwise keep the memory of the decoded value for each attribute.

8.2. LASeR mini

8.2.1. Applications

Rich-media services on mid- and lower-end embedded devices.

8.2.2. List of Tools/Functionalities

8.2.2.1. video element

The attribute **transformBehavior** shall be restricted to values “pinned | pinned90 | pinned180 | pinned270”.

The attribute **overlay** shall be restricted to values “top” or “fullscreen”.

8.2.2.2. stroking

The attributes **stroke-linecap** and **stroke-linejoin** are restricted to the value “butt” and “miter” respectively.

The attributes **stroke-miterlimit, stroke-dasharray** and **stroke-dashoffset** are forbidden.

8.2.2.3. animatable attributes

The following attributes are not animatable:

- the CTM scale of an object painted with a gradient with **gradientUnits**=”objectBoundingBox”.
- the width and height of a rect painted with a gradient with **gradientUnits**=”objectBoundingBox”.
- the viewBox, width or height of a scene where a gradient has **gradientUnits**=”userSpaceOnUse”.
- in general, any animation that requires the recomputation of a gradient at each frame.

8.2.2.4. inheritance

Attributes with a possible value of “inherit” are only allowed on objects which use their value directly, not on objects that pass the value to their children. The “inherit” value is forbidden.

NOTE this restriction will be implemented in a schema that will be used for validation of the profile.

Hints such as *-rendering are only allowed on the root svg.

pointer-events is only allowed on the root svg.

8.2.2.5. animation

For elements **set**, **animate**, **animateColor**, **animateTransform** and **animateMotion**, the following restrictions apply:

- the value of the attribute **fill** is restricted to “freeze”,
- the value of the attribute **additive** is restricted to “replace”,
- either attribute **from** or **values** must be specified.

Any content requiring the restoration of a DOM value of an attribute after it has been animated is non conformant. Authors shall make sure that, when deleting an animation or replacing an xlink:href, any previously animated value is made irrelevant by any appropriate mean (e.g. deletion of the target element, replacement of the animated value, etc.)

8.2.2.6. xlink and xml

Attributes xml:base, xml:lang, xml:space, xlink:title, xlink:type, xlink:role, xlink:arcrole, xlink:actuate and xlink:show are forbidden.

8.2.2.7. LASeR Commands

color-rendering, shape-rendering, text-rendering, image-rendering are not updatable in this profile.

8.2.3. Comparison with existing profiles and object types

This is a subset of LASeR full.

8.2.4. Profile definition

This table defines the allowed elements in the profile and the list of their possible attributes, possibly with restrictions when listed in bold.

Element name	Attributes
a	id externalResourcesRequired display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform xlink:href
animate	id lsr:enabled begin end dur repeatCount repeatDur restart **attributeName fill=”freeze”** to **xlink:href(no replace)** from by values calcMode keyTimes keySplines **additive=”replace”** accumulate
animateColor	id lsr:enabled begin end dur repeatCount repeatDur restart **attributeName fill=”freeze”**to **xlink:href(no replace)** from by values calcMode keyTimes keySplines **additive=”replace”** accumulate
animateMotion	id path keyPoints rotate lsr:enabled begin end dur repeatCount repeatDur restart **fill=”freeze”** to **xlink:href(no replace)** from by values calcMode keyTimes keySplines **additive=”replace”** accumulate
animateTransform	id type lsr:enabled begin end dur repeatCount repeatDur restart attributeName **fill=”freeze”**to **xlink:href(no replace)** from by values calcMode keyTimes keySplines **additive=”replace”** accumulate
audio	id externalResourcesRequired begin end clipBegin clipEnd repeatCount repeatDur syncBehavior syncTolerance audio-level syncReference requiredFeatures requiredExtensions systemLanguage requiredFormats xlink:href
circle	id cx cy r display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
conditional	externalResourcesRequired id begin type lsr:enabled
cursorManager	id x y xlink:href
defs	id
desc	id
ellipse	id rx ry cx cy display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
foreignObject	id externalResourcesRequired width height x y viewport-fill viewport-fill-opacity display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats
g	id externalResourcesRequired display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
image	id externalResourcesRequired width height x y **transformBehavior(no geometric)** opacity display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform xlink:href
line	id x1 y1 x2 y2 display stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
linearGradient	id gradient-units x1 x2 y1 y2 display visibility
ev:listener	id lsr:enabled delay event observer timeAttribute handler propagate defaultAction target

metadata	id
mpath	id xlink:href
path	id d display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
polygon	id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
polyline	id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
radialGradient	id gradient-units cx cy r display visibility
rect	id width height x y rx ry display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
rectClip	id externalResourcesRequired size display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
selector	id externalResourcesRequired choice display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
set	id lsr:enabled begin end dur repeatCount repeatDur restart **attributeName fill="freeze"** to **xlink:href(no replace)**
simpleLayout	id externalResourcesRequired delta display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform
stop	id offset display stop-color stop-opacity visibility
svg	id externalResourcesRequired **width(not animatable if gradient with gradientUnits="userSpaceOnUse") height(not animatable if gradient with gradientUnits="userSpaceOnUse") viewBox(not animatable if gradient with gradientUnits="userSpaceOnUse")** preserveAspectRatio zoomAndPan viewport-fill viewport-fill-opacity syncBehavior syncBehaviorDefault syncTolerance syncToleranceDefault display pointer-events visibility color-rendering shape-rendering image-rendering text-rendering
switch	id externalResourcesRequired display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats
text	id display display-align fill fill-opacity fill-rule font-family font-size font-style font-weight line-increment stroke stroke-opacity stroke-width text-anchor visibility requiredFeatures requiredExtensions systemLanguage requiredFormats editable x y transform text-decoration
title	id
tspan	id display fill fill-opacity fill-rule font-family font-size font-style font-weight line-increment stroke stroke-opacity stroke-width text-anchor visibility requiredFeatures requiredExtensions systemLanguage requiredFormats text-decoration
use	id externalResourcesRequired overflow x y display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats transform xlink:href
video	id externalResourcesRequired begin end clipBegin clipEnd repeatCount repeatDur syncBehavior syncBehaviorDefault syncTolerance syncToleranceDefault audio-level syncReference requiredFeatures requiredExtensions systemLanguage requiredFormats xlink:href width height x y **overlay="top | LASeR_fullscreen" transformBehavior(no geometric)** transform

8.2.5. Level definitions

Level 0: no restriction.

8.3. LASeR full

8.3.1. Applications

Advanced rich-media services on higher end embedded devices.

8.3.2. List of Tools/Functionalities

This is the complete profile.

8.3.3. Comparison with existing profiles and object types

This profile is a superset of LASeR mini.

8.3.4. Profile definition

Table 7 defines the allowed elements in the profile and the list of their possible attributes.

8.3.5. Level definitions

Level 0: no restriction.

8.4. LASeR Core

8.4.1. Applications

Rich-media services on mid- and lower-end embedded devices.

The purpose of LASeR Core is to allow J2ME implementations of LASeR. One specific feature of Core is that the implementation does **not** need to implement 2 trees, one DOM tree and one composition tree, or otherwise keep the memory of the decoded value for each attribute.

8.4.2. List of Tools/Functionalities

8.4.2.1. video element

The attribute **transformBehavior** shall be restricted to values “pinned | pinned90 | pinned180 | pinned270”.

The attribute **overlay** shall be restricted to values “top”.

8.4.2.2. stroking

The attributes **stroke-linecap** and **stroke-linejoin** are restricted to the value “butt” and “miter” respectively.

The attributes **stroke-miterlimit, stroke-dasharray** and **stroke-dashoffset** are forbidden.

8.4.2.3. inheritance

Attributes with a possible value of “inherit” are only allowed on objects which use their value directly, not on objects that pass the value to their children. The “inherit” value is forbidden.

Hints such as *-rendering are only allowed on the root svg.

pointer-events is only allowed on the root svg.

level 1: pointer-events is limited to values bounding-box and none.

8.4.2.4. animation

For elements **animate**, **animateColor**, **animateTransform** and **animateMotion**, the following restrictions apply:

- the value of the attribute **fill** is restricted to “freeze”,
- the value of the attribute **additive** is restricted to “replace”,
- the value of **calcMode** is restricted to “linear” or “discrete”, but on animateMotion,
- either attribute **from** or **values** must be specified.

The value of restart is restricted to “***always***”.

Any content requiring the restoration of a DOM value of an attribute after it has been animated is non conformant. In other words, authors shall make sure that, e.g. when deleting an animation or replacing an xlink:href of an animation element, any previously animated value is made irrelevant by any appropriate mean (e.g. deletion of the target element, replacement of the animated value, etc).

8.4.2.5. xlink and xml

Attributes xml:base, xml:lang, xml:space, xlink:title, xlink:type, xlink:role, xlink:arcrole, xlink:actuate and xlink:show are forbidden.

8.4.2.6. LASeR Commands

color-rendering, shape-rendering, text-rendering, image-rendering, attributeName are not updatable.

Update of attributes from timed elements are restricted in the following manner:

- accumulate, additive, attributeName, by, calcMode, dur, fill, from, id, keyPoints, keySplines, keyTimes, max, min, path, repeatCount, repeatDur, restart, to, values, xlink:actuate, xlink:arcrole, xlink:href, xlink:role, xlink:show, xlink:title, xlink:type, xml:base cannot be updated
- begin, end, clipBegin and clipEnd may be updated

8.4.2.7. uDOM

uDOM is not mandatory in LASeR Core profile

8.4.2.8. Navigation

The events focusin and focusout are not supported. The attribute focusable, focusHighlight and nav-* are not supported.

8.4.2.9. text

The attributes x and y of text can only take single values, not lists of coordinates.

The attribute xml:space is mandatory with value “preserve” on text and tspan.

The possible children for text are restricted to: tspan.

8.4.2.10. Units

Only % and px are allowed as length units in LASeR Core.

8.4.2.11. transform

The ref() construct is not allowed in transform attribute values.

8.4.2.12. System Paint Servers

System paint servers are not allowed.

8.4.2.13. Events

The following events are not allowed: DOMFocusIn, DOMFocusOut, click, keydown, keyup, resize, scroll, zoom, rotate, timer, connection*.

8.4.2.14. SMIL hyperlinking

Hyperlinks to an element, whether timed or not, are not allowed.

8.4.2.15. use

The xlink:href attribute of the use element is restricted to designating the ID of an element of the current scene. More complex URLs are not part of LASeR Core.

8.4.3. Comparison with existing profiles and object types

LASeR Core is a superset of LASeR mini

8.4.4. Profile definition

This table defines the allowed elements in the profile and the list of their possible attributes, possibly with restrictions when listed in bold.

Element name	Attributes
a	id display transform xlink:href
animate	id lsr:enabled begin end dur repeatCount repeatDur **restart="always" attributeName fill="freeze"** to **xlink:href(no replace)** from by values **calcMode="discrete|linear"** keyTimes **additive="replace"** accumulate
animateColor	id lsr:enabled begin end dur repeatCount repeatDur **restart="always" attributeName fill="freeze"**to **xlink:href(no replace)** from by values **calcMode="discrete|linear"**keyTimes **additive="replace"** accumulate
animateMotion	id path keyPoints rotate lsr:enabled begin end dur repeatCount repeatDur **restart="always" fill="freeze"** to **xlink:href(no replace)** from by values **calcMode="discrete|linear|paced"** keyTimes **additive="replace"** accumulate
lsr:animateScroll	id xml:base **xlink:href(no replace)** by **from(no id) to(no id)** delayAtStart delayAtEnd speed direction begin dur end **fill="freeze"** restart repeatCount repeatDur
animateTransform	id type lsr:enabled begin end dur repeatCount repeatDur **restart="always" attributeName fill="freeze"**to **xlink:href(no replace)** from by values **calcMode="discrete|linear"**keyTimes **additive="replace"** accumulate
audio	id begin end clipBegin clipEnd repeatCount repeatDur audio-level xlink:href
circle	id cx cy r display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
conditional	id begin type lsr:enabled
lsr:cursorManager	id x y xlink:href
defs	id
desc	id
ellipse	id rx ry cx cy display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
foreignObject	id width height x y viewport-fill viewport-fill-opacity display
g	id display transform
image	id width height x y **transformBehavior(no geometric)** display transform xlink:href
line	id x1 y1 x2 y2 display stroke stroke-opacity stroke-width transform vector-effect
ev:listener	id lsr:enabled event observer handler propagate defaultAction target
metadata	id
mpath	id xlink:href
path	id d display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
polygon	id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
polyline	id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
rect	id width height x y rx ry display fill fill-opacity fill-rule stroke stroke-opacity stroke-width transform vector-effect
lsr:rectClip	id size display transform
lsr:selector	id choice display transform
set	id lsr:enabled begin end dur repeatCount repeatDur **restart="always" attributeName fill="freeze"** to **xlink:href(no replace)**
lsr:setScroll	id begin increment to direction **xlink:href(no replace)** xml:base
lsr:simpleLayout	id delta display transform
svg	id width height viewBox preserveAspectRatio **zoomAndPan="disable"** viewport-fill viewport-fill-opacity display pointer-events color-rendering shape-rendering image-rendering text-rendering
text	id display fill fill-opacity fill-rule font-family font-size font-style font-weight stroke stroke-opacity stroke-

	width text-anchor editable x y transform text-decoration **xml:space=”preserve”**
title	id
tspan	id display fill fill-opacity fill-rule font-family font-size font-style font-weight stroke stroke-opacity stroke-width text-decoration **xml:space=”preserve”**
use	id overflow x y display transform xlink:href
lsr:updates	begin clipBegin clipEnd end flow xlink:href id repeatCount repeatDur security lsr:syncReference
video	id begin display end clipBegin clipEnd repeatCount repeatDur audio-level xlink:href width height x y **overlay=”top”** lsr:fullscreen **transformBehavior(no geometric)** transform

8.4.5. Level definitions

Level 0: only one scene stream active at a time

9. Compatibility of SAF Packet

This clause is informative.

The Packet defined in subclause 7.3 is compatible with the SL Packet defined in ISO/IEC 14496-1:2004 with the predefined configuration defined in Table 8.

Table 8 — Detailed SLConfigDescriptor values for `predefined=0x03`

fields in SLConfigDescriptor	predefined value
useAccessUnitStartFlag	0
useAccessUnitEndFlag	0
useRandomAccessPointFlag	1
useTimeStampsFlag	1
timeStampResolution	1000
timeStampLength	30
AU_length	16
degradationPriorityLength	0
AU_seqNumLength	15

`useAccessUnitStartFlag` – indicates that the `accessUnitStartFlag` is present in each packet header of this elementary stream.

`useAccessUnitEndFlag` – indicates that the `accessUnitEndFlag` is present in each packet header of this elementary stream.

`useRandomAccessPointFlag` – indicates that the `RandomAccessPointFlag` is present in each packet header of this elementary stream.

`useTimeStampsFlag`: indicates that time stamps are used for synchronisation of this media stream. They are conveyed in the packet headers. Generally it is possible for video to have different values of decoding time and composition time in general, but it shall be assumed that decoding time is same with composition time for SAF stream. Therefore, `compositionTimeStamp` shall only be presented in the packet header for SAF stream.

`timeStampResolution` – is the resolution of the time stamps in clock ticks per second.

`timeStampLength` – is the length of the time stamp fields in packet headers.

`AU_Length` – is the length of the `accessUnitLength` fields in packet headers for this media stream.

`degradationPriorityLength` – is the length of the `degradationPriority` field in packet headers for this media stream.

`AU_seqNumLength` – is the length of the `AU_sequenceNumber` field in SL packet headers for this elementary stream.

10. Carriage of LASeR and SAF

10.1. Storage of LASeR in MP4 files

10.1.1. LASeR Track Structure

In the terminology of the ISO Base Media File Format specification, LASeR tracks are scene tracks. They therefore use:

a) a handler_type of 'sdsm' in the HandlerBox;

b) a video media header `'vmhd'`;

c) and, as defined below, a derivative of the `SampleEntry`.

An access unit that starts with a 'replace scene' or 'refresh scene' command is a Sync Sample, and is marked as such in the sync sample table. 'Refresh scene' commands may be placed into a 'switch pictures' track as defined in the AVC file format, as they are logically equivalent to AVC SI pictures. The use of shadow sync is deprecated, having been superseded by the new switch pictures facility.

The timescale for the LASeR stream should be suitably chosen to achieve the desired accuracy of timing of access units.

10.1.2. Resources

A LASeR track may contain a meta-data atom ('meta') with resources in it. Those resources 'shadow' data in the same directory as the ISO file itself came from.

If there is no primary meta-data, then the `handler_type` in the handler box of the meta-data should be set to the code `'null'`.

It is also permitted to store a single access unit of LASeR as the primary item in a meta-box. In that case, the `handler_type` is set to `'lsr1'`. The actual LASeR access unit is stored in a binary XML box inside the meta box, or is referenced as the primary item. In this case (there is a 'static' LASeR scene as the primary item) the meta-box may be at file-level. Otherwise, if it is used as to store resources for the LASeR scene, the meta-box would be stored within the LASeR track.

Items within the meta-box can be referred to using the URL forms documented in subclause 8.44.7 of [ISO/IEC 14496-12]. If a URL form for tracks within the same `'moov'` atom as the LASeR track is needed, the fragment syntax "#trackID=<n>" where <n> is the desired track identifier, may be used.

10.1.3. Composition

LASeR tracks in an ISO file that has other audio or video tracks are composed with those tracks. The composition falls into two classes: temporal composition, and audio and visual composition.

As defined in ISO/IEC 14496-14, the default behavior is that time-parallel tracks (streams) in ISO-family files have their time-lines 'locked together' unless 'sync' track references are used. Therefore an ISO file with two (or more) tracks, one of which is a LASeR track, and no 'sync' references, has the timelines of those tracks synchronized. The LASeR scene cannot set the time of those other tracks independently of the time of the LASeR track. If this is not desired, an embedded stream (or ISO container) should be used or the time-lines 'unlocked' by using 'sync' track references.

The visual and audio composition of the other tracks may be defined by the LASeR stream. If the LASeR stream does not reference those tracks, then the audio/visual composition of those tracks with the LASeR stream or with each other is not defined by this specification. LASeR refers to these tracks by streamID, which is the same as the TrackID in ISO family files.

10.1.4. LASeR Stream Definition

This subclause defines the sample entry and sample format for LASeR video elementary streams.

10.1.5. Sample description name and format

10.1.6. Definition

Box Types: 'lsr1', 'lsrC', 'm4ds','btrt'
Container: Sample Table Box ('stbl')
Mandatory: The lsr1 box is mandatory
Quantity: One or more sample entries may be present

An LASeR sample entry shall contain an LASeR Configuration Box, as defined below. This includes a `Laser Header`.

An optional `MPEG4BitRateBox` may be present in the LASeR sample entry to signal the bit rate information of the LASeR stream. Extension descriptors that should be inserted into the Elementary Stream Descriptor, when used in MPEG-4, may also be present (These two boxes are identical to those in ISO/IEC 14496-15.).

Multiple sample descriptions may be used, as permitted by the ISO Base Media File Format specification, to indicate sections that use different configurations.

10.1.7. Syntax

```
   // Visual Sequences
class LASeRConfigurationBox extends Box('lsrC') {
   LASeRHeader() LSRHdr;
}


class MPEG4BitRateBox extends Box('btrt'){
   unsigned int(32) bufferSizeDB;
   unsigned int(32) maxBitrate;
   unsigned int(32) avgBitrate;
}

class MPEG4ExtensionDescriptorsBox extends Box('m4ds') {
   Descriptor Descr[0 .. 255];
}

class LASeRSampleEntry() extends SampleEntry ('lsr1'){
   LASeRConfigurationBox   config;
   MPEG4BitRateBox ();              // optional
   MPEG4ExtensionDescriptorsBox (); // optional
}
```

10.1.8. Semantics

`Descr` is a descriptor which should be placed in the `ElementaryStreamDescriptor` when this stream is used in an MPEG-4 systems context. This does not include `SLConfigDescriptor` or `DecoderConfigDescriptor`, but includes the other descriptors in order to be placed after the `SLConfigDescriptor`.

`bufferSizeDB` gives the size of the decoding buffer for the elementary stream in bytes

`maxBitrate` gives the maximum rate in bits/second over any window of one second

`avgBitrate` gives the average rate in bits/second over the entire presentation

10.1.9. Sample Format

An LASeR sample is an LASeR access unit.

10.2. Carriage of SAF Streams over HTTP

SAF streams can be carried over HTTP with the content type: "application/saf" (for generic SAF) or "application/saf+laser" (for SAF including a LASeR scene)

10.3. Carriage of SAF Streams over RTP

RFC 3640 shall be used to define the mapping of SAF information to RTP information.

Mapping of SAF_packet headers to RTP payload format is illustrated in Figure 11. Since RFC 3640 allows aggregation of SL packets, CTS-delta may be present if several SAF_packets are carried by one RTP packet.

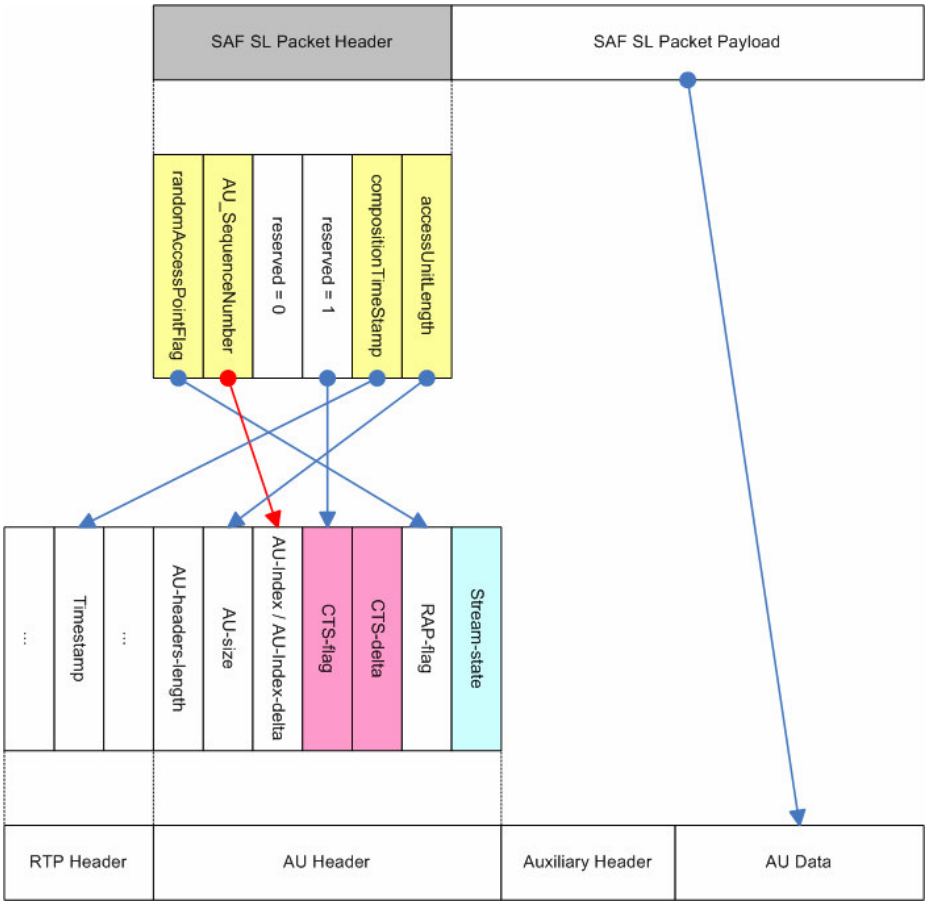


Figure 11 — Mapping of SAF_SL_packet to RTP payload format in RFC 3640

10.4. Carriage of SAF Streams over MPEG-2 Systems

As all SAF information is defined in terms reusing MPEG-4 Sync Layer definitions, the mapping of SL to MPEG-2 Systems shall be used to transport SAF information over MPEG-2 Systems.

11. Electronic Attachments

Even if some of the schemas below are marked normative, the text of the specification has precedence over the schemas.

The electronic attachments to this specification are:

- LASeRML/saf1.xsd: (normative) documents the syntax of an XML equivalent to the SAF binary syntax for use in LASeR conformance and reference software activities

- LASeRML/laser-datatypes1.xsd: (normative) documents all datatypes used in the other schemas

- LASeRML/laser1.xsd: (normative) documents the XML syntax of LASeR Commands for use in LASeR conformance and reference software activities

- LASeRML/svg1.xsd: (normative) documents the XML syntax of the common part between SVG and LASeR; it is not intended to validate SVGT documents; this is for use in LASeR conformance and reference software activities

- LASeRML/xml.xsd: (informative) copy of the W3C XML schema for XML Base (http://www.w3.org/TR/xmlbase/), which is referenced by the other schemas and is provided here for convenience

- LASeRML/xlink.xsd: (informative) copy of the W3C XML schema for Xlink (http://www.w3.org/TR/xlink/), which is referenced by the other schemas and is provided here for convenience

- LASeRML/ev.xsd: (informative) copy of the W3C XML schema for the listener element, which is referenced by the other schemas and is provided here for convenience

- intermediateXML/sdl.txt: (normative) documented in the next subclause

- LASeRML/saf2.xsd: (normative) documents the syntax of an XML equivalent to the SAF binary syntax for use in LASeR conformance and reference software activities

- LASeRML/laser-datatypes2.xsd: (normative) documents all datatypes used in the other schemas

- LASeRML/laser2.xsd: (normative) documents the XML syntax of LASeR Commands for use in LASeR conformance and reference software activities

- LASeRML/svg2.xsd: (normative) documents the XML syntax of the common part between SVG and LASeR; it is not intended to validate SVGT documents; this is for use in LASeR conformance and reference software activities

- sdl_amd1.txt: (normative) documented in the next subclause

12. Binary Syntax for the LASeR Encoding

12.1. Decoding Process

12.1.1. Introduction

The decoding process is described in SDL [ISO/IEC 14496-1]. This subclause describes the semantics of global decoding variables and the representation of types. The entry points are the classes LASeRHeader and LASeRUnit.

NOTE 1 A decoder compliant with ISO/IEC 23001-1 will decode the syntax as specified in this clause when the configuration according to clause 14 is used.

NOTE 2 "enumeration" is a syntax extension over SDL. The syntax of "enumeration" is:

- the keyword enumeration
- an optional descriptive string
- a set of enumeration values, whose encoding is a zero-based integer representing their rank in the list

12.1.2. Data types

Colors may be represented as indices into a color palette. The color palette is sent piecewise at the beginning of each access unit, within the class colorInitialisation. The variable colorIndex is a count of all received colors. colorIndexBits represents the optimal number of bits used to encode a color index.

In each *Initialisation section, table are filled (such as red[], green[] and blue[] for colorInitialisation) and a *Index is used (such as colorIndex for colorInitialisation). When the value of resetEncodingContext in LASeRUnitHeader is true, then the tables are reset to empty and the *Index is reset to -1.

Font names may be represented as indices into a font table. The font table is sent piecewise at the beginning of each access unit, within the class fontInitialisation. The variable fontIndex is a count of all received font names. fontIndexBits represents the optimal number of bits used to encode a font index.

vluimsbfX is a variable length code unsigned integer, most significant bit first. The first n bits (Ext) which are 1 except of the n-th bit which is 0, indicate that the integer is encoded by n*(X-1) bits.
An example for X = 5 is shown in Figure 12.

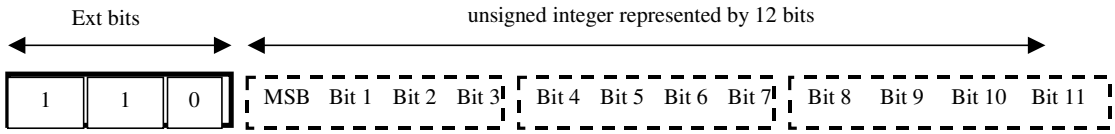


Figure 12 — Example for the vluimsbf5 data type

In ***attr_custom_0to1float***, a floating point number between 0 and 1 is quantified linearly on 8 bits, between 0x0 and 0x255.

In ***matrix***, the first two lines of a 3x3 transformation matrix are represented:

$$\begin{pmatrix} xx & xy & xz \\ yx & yy & yx \\ 0 & 0 & 1 \end{pmatrix}$$

The numbers xx, xy, yx and yy are represented as fixed point numbers with 8 bits of fractional part.

In ***attr_custom_coordinate*** and ***attr_custom_pointSequence***, the actual floating point coordinate ***c*** is reconstructed from the syntax element ***uint(x) coord*** through the following equation:

- if the highest bit of ***coord*** is set, ***c*** = ***coord*** – 2^{x}, otherwise ***c*** = ***coord***

- ***c*** is divided by $2^{resolution}$.

resolution and ***coordBits*** are conveyed in the LASeRHeader. The encoding of ***preserveAspectRatio*** is:

- 2 bits for x (Min=1, Mid=2, Max=3),
- 2 bits for y (Min=1, Mid=2, Max=3)
- 1 bit for meet (0) or slice (1)

All time values are encoded in ticks: the tick for clipBegin and clipEnd is a 1000^{th} of a second, and is defined by LASeRHeader.timeResolution for all other time fields.

In ***attr_custom_anyURI***, a URL will be encoded as a string, unless it is a data: URL, in which case the header of the URL is encoded as a string, and the resource part shall be conveyed in the ***byte[len] data.***

The function log2sup(), used in *Initialisation, represents the number of bits necessary to represent the number passed as parameter, which is the ceiling of the logarithm of base 2 of the parameter plus one.

12.1.3. Carriage of private data

LASeR allows for the carriage of private data. Private data may be skipped by a decoder compliant with this specification. The coding elements privateAnyXMLElement, privateElement, privateChildren, privateAnyXMLAttribute and privateAttribute are designed to carry private data. The entry points are privateAnyXMLElement and privateAnyXMLAttribute. These elements refer to configuration information defined in privateDataIdentifierInitialisation and anyXMLInitialisation. The coding elements privateOpaqueElement and privateOpaqueAttribute are designed to carry opaque private data.

LASeR allows for several encodings for private data. Table 9 shows the code points for private data encoding. In opaque mode, the value privateDataIdentifierIndex should be used to discriminate the “owner” of the data.

Table 9 — Code points for private data encoding

codePoint	privateDataContainer	extConfiguration
	defines encoding	*defines configuration for*
0	anyXML as defined in 12.2.3.20	not used
1	opaque binary data	opaque binary data
2	23001-1 encoding	23001-1 compliant decoderInit
3-7	reserved for ISO use	reserved for ISO use
8-15	to be defined by RA	to be defined by RA

In ***privateDataIdentifierInitialisation***, private data identifiers are defined in the access unit where they are first used. They are referred to in other constructs by index. The goal of private data identifiers is to:

- tag private binary data within privateOpaqueElement
- be used as name space urn within privateElement, privateAttribute and privateAnyXMLAttribute.

In ***anyXMLInitialisation***, element and attribute names are declared, for use in the anyXML mode (privateDataType == 0) of privateElement, privateAttribute and privateAnyXMLAttribute. Each name consists in an optional index into the private data identifier table, pointing at a URN, followed by a string for the local part of the name. For each element name, a set of attribute names is stored. The first element name is left empty, and the corresponding set of attribute names is used to convey private attributes placed on LASeR elements, or with a name space different from their host private element.

If codePoint 2 is used, then the data is 23001-1 compliant private extensions. Decoders not compliant with the signalled 23001-1 private extension (identified by the namespace) are not mandated to decode this private extension data and may skip it.

The following sentences ONLY apply for decoders compliant with both 23001-1 and 14496-20 :

If no decoderInit signalled in LASeRHeader, then the decoderInit is [ref to DecoderInit for V2]

- If decoderInit signalled then decoderInit is compatible/extension of [ref to DecoderInit for V2]
- The attached 'encoding' schemas shall be used for the configuration of the 23001-1 (encoding schemas are normative)
- The Classification scheme and type codec association described in section 14 shall be used(classificiation schemes and association are normative)
- The transformation process between the validation schema and the encoding schemas are described informatively in clause 13.

12.1.4. Animations and updates

In ***attr_custom_animatedValue***, ***attr_custom_animatedValues*** and ***attr_custom_updateValue***, the meaning of escapeFlag and escapeEnum is the following. In the case where an attribute can have either a range of values, or some enumerated values, escapeFlag is set when the encoded value is an enumerated value rather than the usual range, and escapeEnum holds that enumerated value. This is the case for any attribute which is a property and which may have the value ‘inherit’, for the attribute choice which may have ‘none’, ‘clip’ and ‘delta’, for the time-related attributes which may have ‘indefinite’ and ‘media’, for the attribute rotate which may have ‘auto’ and ‘auto-reverse’ for sync-related attributes which may have ‘default’, for the attribute line-increment which may have ‘auto’ and ‘inherit’ and for future focus-related attributes which may have ‘auto’ and ‘self’. The enumeration value to use for escapeEnum is to be found in the normal encoding of the attribute in question, e.g. for choice, the enumeration to use can be found in the class typ_choiceType. For animated properties, any value can be used for escapeEnum, as the only encodable value is ‘inherit’.

The key values of animations are encoded in a small number of types defined in Table 10. The correspondance of attribute to encoded type is defined in Table 11.

Table 10 — Types for encoded key values of animations

Type	Encoded value
ANIMTYPE_string	0
ANIMTYPE_float	1
ANIMTYPE_path	2
ANIMTYPE_pointSeq	3
ANIMTYPE_fraction	4
ANIMTYPE_color	5
ANIMTYPE_enum	6
reserved	7
ANIMTYPE_floats	8
ANIMTYPE_point	9
ANIMTYPE_id	10
ANIMTYPE_font	11
ANIMTYPE_uri	12

Table 11 — Attribute to encoded type correspondance

Attribute name	Encoded type for animation	Attribute name	Encoded type for animation
a.target	ANIMTYPE_string	r	ANIMTYPE_float
audio-level	ANIMTYPE_fraction	rotate	ANIMTYPE_float
choice	ANIMTYPE_enum	rx	ANIMTYPE_float
color	ANIMTYPE_color	ry	ANIMTYPE_float
color-rendering	ANIMTYPE_enum	shape-rendering	ANIMTYPE_enum
cx	ANIMTYPE_float	size	ANIMTYPE_point
cy	ANIMTYPE_float	stop-color	ANIMTYPE_color

d	ANIMTYPE_path	stop-opacity	ANIMTYPE_fraction
display	ANIMTYPE_enum	stroke	ANIMTYPE_color
display-align	ANIMTYPE_enum	stroke-dasharray	ANIMTYPE_floats
editable	ANIMTYPE_enum	stroke-dashoffset	ANIMTYPE_float
fill	ANIMTYPE_color	stroke-linecap	ANIMTYPE_enum
fill-opacity	ANIMTYPE_fraction	stroke-linejoin	ANIMTYPE_enum
fill-rule	ANIMTYPE_enum	stroke-miterlimit	ANIMTYPE_float
nav-next	ANIMTYPE_id	stroke-opacity	ANIMTYPE_fraction
nav-prev	ANIMTYPE_id	stroke-width	ANIMTYPE_float
nav-right	ANIMTYPE_id	target	ANIMTYPE_string
nav-up	ANIMTYPE_id	text-anchor	ANIMTYPE_enum
nav-up-right	ANIMTYPE_id	transform	ANIMTYPE_floats
nav-up-left	ANIMTYPE_id	type	ANIMTYPE_enum
nav-down	ANIMTYPE_id	vector-effect	ANIMTYPE_enum
nav-down-right	ANIMTYPE_id	viewBox	ANIMTYPE_floats
nav-down-left	ANIMTYPE_id	viewport-fill	ANIMTYPE_color
nav-left	ANIMTYPE_id	viewport-fill-opacity	ANIMTYPE_fraction
focusable	ANIMTYPE_id	visibility	ANIMTYPE_enum
font-family	ANIMTYPE_font	width	ANIMTYPE_float
font-size	ANIMTYPE_float	x	ANIMTYPE_float
font-style	ANIMTYPE_enum	x1	ANIMTYPE_float
font-weight	ANIMTYPE_enum	x2	ANIMTYPE_float
gradientUnits	ANIMTYPE_enum	xlink:href	ANIMTYPE_uri
height	ANIMTYPE_float	y	ANIMTYPE_float
image-rendering	ANIMTYPE_enum	y1	ANIMTYPE_float
listener.target	ANIMTYPE_id	y2	ANIMTYPE_float
opacity	ANIMTYPE_fraction		
pathLength	ANIMTYPE_float		
pointer-events	ANIMTYPE_ints		
points	ANIMTYPE_pointSeq		
preserveAspectRatio	ANIMTYPE_enum		

12.1.5. Encoding of enumerations

The SDL described in clause 12.2 makes use of the following syntax: "// Enumeration:" followed by a space separated list of "string{value}", where *value* is the decimal value used to encode the *string* in the bit field following on the next line. Values not listed but encodable in this bit field are ISO reserved.

The SDL for the LASeR Encoding is attached as sdl.txt and inlined below.

12.2. Binary Syntax

12.2.1. Main Structure

```
class element_a {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_target;
  if(has_target) {
    attr_custom_byteAlignedString target;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class privateAttributeContainer {
  bit(4) codePoint;
  switch(codePoint){
    case 0:
      privateAnyXMLAttribute anyXML;
      break;
    case 1:
      privateOpaqueAttribute opaque;
      break;
    case 2:
      attr_any reserved;
      break;
    default:
      attr_custom_extension ext;
      break;
    }
}
class elements {
  bit(6) ch4;
  switch(ch4){
    case 0:
      element_a a;
      break;
    case 1:
      element_animate animate;
      break;
    case 2:
      element_animate animateColor;
      break;
    case 3:
      element_animateMotion animateMotion;
      break;
    case 4:
      element_animateTransform animateTransform;
      break;
    case 5:
      element_audio audio;
      break;
    case 6:
      element_circle circle;
      break;
    case 7:
      element_defs defs;
      break;
    case 8:
      element_desc_metaData_title desc;
      break;
    case 9:
      element_ellipse ellipse;
```

```
      break;
    case 10:
      element_foreignObject foreignObject;
      break;
    case 11:
      element_g g;
      break;
    case 12:
      element_image image;
      break;
    case 13:
      element_line line;
      break;
    case 14:
      element_linearGradient linearGradient;
      break;
    case 15:
      element_desc_metaData_title metadata;
      break;
    case 16:
      element_mpath mpath;
      break;
    case 17:
      element_path path;
      break;
    case 18:
      element_polygon polygon;
      break;
    case 19:
      element_polygon polyline;
      break;
    case 20:
      element_radialGradient radialGradient;
      break;
    case 21:
      element_rect rect;
      break;
    case 22:
      element_sameg sameg;
      break;
    case 23:
      element_sameline sameline;
      break;
    case 24:
      element_samepath samepath;
      break;
    case 25:
      element_samepathfill samepathfill;
      break;
    case 26:
      element_samepolygon samepolygon;
      break;
    case 27:
      element_samepolygonfill samepolygonfill;
      break;
    case 28:
      element_samepolygonstroke samepolygonstroke;
      break;
    case 29:
      element_samepolygon samepolyline;
      break;
    case 30:
      element_samepolygonfill samepolylinefill;
      break;
    case 31:
      element_samepolygonstroke samepolylinestroke;
      break;
    case 32:
      element_samerect samerect;
      break;
    case 33:
      element_samerectfill samerectfill;
      break;
    case 34:
      element_sametext sametext;
      break;
    case 35:
      element_sametextfill sametextfill;
      break;
    case 36:
      element_sameuse sameuse;
      break;
    case 37:
      element_script script;
      break;
    case 38:
      element_set set;
```

```
      break;
    case 39:
      element_stop stop;
      break;
    case 40:
      element_switch switch;
      break;
    case 41:
      element_text text;
      break;
    case 42:
      element_desc_metaData_title title;
      break;
    case 43:
      element_tspan tspan;
      break;
    case 44:
      element_use use;
      break;
    case 45:
      element_video video;
      break;
    case 46:
      element_listener listener;
      break;
    case 47:
      element_conditional conditional;
      break;
    case 48:
      element_cursorManager cursorManager;
      break;
    case 49:
      element_any extElement;
      break;
    case 50:
      privateElementContainer privateElementContainer;
      break;
    case 51:
      element_rectClip rectClip;
      break;
    case 52:
      element_selector selector;
      break;
    case 53:
      element_simpleLayout simpleLayout;
      break;
    case 54:
      attr_custom_byteAlignedString textContent;
      break;
    }
}
class element_animate {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_accumulate;
  if(has_accumulate) {
    attr_accumulate accumulate;
  }
  bit(1) has_additive;
  if(has_additive) {
    attr_additive additive;
  }
  bit(1) has_by;
  if(has_by) {
    attr_custom_AnimatedValue by;
  }
  bit(1) has_calcMode;
  if(has_calcMode) {
    attr_calcMode calcMode;
  }
  bit(1) has_from;
  if(has_from) {
    attr_custom_AnimatedValue from;
  }
  bit(1) has_keySplines;
  if(has_keySplines) {
    attr_custom_fraction12List keySplines;
```

```
  }
  bit(1) has_keyTimes;
  if(has_keyTimes) {
    attr_custom_fraction12List keyTimes;
  }
  bit(1) has_values;
  if(has_values) {
    attr_custom_AnimatedValues values;
  }
  bit(1) has_attributeType;
  if(has_attributeType) {
    attr_attributeType attributeType;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_time dur;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_animFill fill;
  }
  bit(1) has_repeatCount;
  if(has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if(has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if(has_restart) {
    attr_restart restart;
  }
  bit(1) has_to;
  if(has_to) {
    attr_custom_AnimatedValue to;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) enabled;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_animateMotion {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_accumulate;
  if(has_accumulate) {
    attr_accumulate accumulate;
  }
  bit(1) has_additive;
  if(has_additive) {
    attr_additive additive;
  }
  bit(1) has_by;
  if(has_by) {
    attr_custom_AnimatedValue by;
  }
  bit(1) has_calcMode;
  if(has_calcMode) {
    attr_calcMode calcMode;
  }
  bit(1) has_from;
  if(has_from) {
    attr_custom_AnimatedValue from;
  }
  bit(1) has_keySplines;
  if(has_keySplines) {
    attr_custom_fraction12List keySplines;
  }
  bit(1) has_keyTimes;
  if(has_keyTimes) {
```

```
    attr_custom_fraction12List keyTimes;
  }
  bit(1) has_values;
  if(has_values) {
    attr_custom_AnimatedValues values;
  }
  bit(1) has_attributeType;
  if(has_attributeType) {
    attr_attributeType attributeType;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_time dur;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_animFill fill;
  }
  bit(1) has_repeatCount;
  if(has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if(has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if(has_restart) {
    attr_restart restart;
  }
  bit(1) has_to;
  if(has_to) {
    attr_custom_AnimatedValue to;
  }
  bit(1) has_keyPoints;
  if(has_keyPoints) {
    attr_floatList keyPoints;
  }
  bit(1) has_path;
  if(has_path) {
    attr_custom_path path;
  }
  bit(1) has_rotate;
  if(has_rotate) {
    attr_rotate rotate;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) enabled;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_animateTransform {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  attr_rotscatra type;
  bit(1) has_accumulate;
  if(has_accumulate) {
    attr_accumulate accumulate;
  }
  bit(1) has_additive;
  if(has_additive) {
    attr_additive additive;
  }
  bit(1) has_by;
  if(has_by) {
    attr_custom_AnimatedValue by;
  }
```

```
    bit(1) has_calcMode;
    if(has_calcMode) {
      attr_calcMode calcMode;
    }
    bit(1) has_from;
    if(has_from) {
      attr_custom_AnimatedValue from;
    }
    bit(1) has_keySplines;
    if(has_keySplines) {
      attr_custom_fraction12List keySplines;
    }
    bit(1) has_keyTimes;
    if(has_keyTimes) {
      attr_custom_fraction12List keyTimes;
    }
    bit(1) has_values;
    if(has_values) {
      attr_custom_AnimatedValues values;
    }
    bit(1) has_attributeType;
    if(has_attributeType) {
      attr_attributeType attributeType;
    }
    bit(1) has_begin;
    if(has_begin) {
      attr_smil_times begin;
    }
    bit(1) has_dur;
    if(has_dur) {
      attr_time dur;
    }
    bit(1) has_fill;
    if(has_fill) {
      attr_animFill fill;
    }
    bit(1) has_repeatCount;
    if(has_repeatCount) {
      attr_repeatCount repeatCount;
    }
    bit(1) has_repeatDur;
    if(has_repeatDur) {
      attr_repeatDur repeatDur;
    }
    bit(1) has_restart;
    if(has_restart) {
      attr_restart restart;
    }
    bit(1) has_to;
    if(has_to) {
      attr_custom_AnimatedValue to;
    }
    bit(1) has_href;
    if(has_href) {
      attr_custom_anyURI href;
    }
    bit(1) enabled;
    bit(1) has_attr_any;
    if(has_attr_any) {
      attr_any any;
    }
    object_content child0;
  }
  class element_audio {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    bit(1) has_rare;
    if(has_rare) {
      attr_custom_rare rare;
    }
    bit(1) has_begin;
    if(has_begin) {
      attr_smil_times begin;
    }
    bit(1) has_dur;
    if(has_dur) {
      attr_time dur;
    }
    bit(1) externalResourcesRequired;
    bit(1) has_repeatCount;
    if(has_repeatCount) {
      attr_repeatCount repeatCount;
    }
    bit(1) has_repeatDur;
    if(has_repeatDur) {
```

```
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if (has_restart) {
    attr_restart restart;
  }
  bit(1) has_syncBehavior;
  if(has_syncBehavior) {
    attr_syncBehavior syncBehavior;
  }
  bit(1) has_syncTolerance;
  if(has_syncTolerance) {
    attr_syncTolerance syncTolerance;
  }
  bit(1) has_type;
  if(has_type) {
    attr_custom_byteAlignedString type;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_clipBegin;
  if(has_clipBegin) {
    attr_time clipBegin;
  }
  bit(1) has_clipEnd;
  if(has_clipEnd) {
    attr_time clipEnd;
  }
  bit(1) has_syncReference;
  if(has_syncReference) {
    attr_custom_anyURI syncReference;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_circle {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_cx;
  if(has_cx) {
    attr_custom_coordinate cx;
  }
  bit(1) has_cy;
  if(has_cy) {
    attr_custom_coordinate cy;
  }
  attr_custom_coordinate r;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_defs {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
```

```
    attr_custom_paint stroke;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_desc_metaData_title {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_ellipse {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_cx;
  if(has_cx) {
    attr_custom_coordinate cx;
  }
  bit(1) has_cy;
  if(has_cy) {
    attr_custom_coordinate cy;
  }
  attr_custom_coordinate rx;
  attr_custom_coordinate ry;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_foreignObject {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  attr_custom_coordinate height;
  attr_custom_coordinate width;
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
```

```
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  bit(1) opt_group;
  if(opt_group) {
    vluimsbf5 occ1;
    for(int t=0;t<occ1;t++) {
      privateElementContainer child0[[t]];
    }
  }
}
class element_g {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_image {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_height;
  if(has_height) {
    attr_custom_coordinate height;
  }
  bit(1) has_opacity;
  if(has_opacity) {
    attr_custom_0to1float opacity;
  }
  bit(1) has_preserveAspectRatio;
  if(has_preserveAspectRatio) {
    attr_preserveAspectRatio preserveAspectRatio;
  }
  bit(1) has_type;
  if(has_type) {
    attr_custom_byteAlignedString type;
  }
  bit(1) has_width;
  if(has_width) {
    attr_custom_coordinate width;
  }
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_transformBehavior;
  if(has_transformBehavior) {
    attr_transformBehavior transformBehavior;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
```

```
  object_content child0;
}
class element_line {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_x1;
  if(has_x1) {
    attr_custom_coordinate x1;
  }
  attr_custom_coordinate x2;
  bit(1) has_y1;
  if(has_y1) {
    attr_custom_coordinate y1;
  }
  attr_custom_coordinate y2;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_linearGradient {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_gradientUnits;
  if(has_gradientUnits) {
    attr_gradientUnits gradientUnits;
  }
  bit(1) has_x1;
  if(has_x1) {
    attr_custom_coordinate x1;
  }
  bit(1) has_x2;
  if(has_x2) {
    attr_custom_coordinate x2;
  }
  bit(1) has_y1;
  if(has_y1) {
    attr_custom_coordinate y1;
  }
  bit(1) has_y2;
  if(has_y2) {
    attr_custom_coordinate y2;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_mpath {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
```

```
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_path {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  attr_custom_path d;
  bit(1) has_pathLength;
  if(has_pathLength) {
    attr_custom_fixed_16_8 pathLength;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_polygon {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  attr_custom_pointSequence points;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_radialGradient {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_cx;
  if(has_cx) {
    attr_custom_coordinate cx;
  }
  bit(1) has_cy;
  if(has_cy) {
    attr_custom_coordinate cy;
  }
```

```
    bit(1) has_gradientUnits;
    if(has_gradientUnits) {
      attr_gradientUnits gradientUnits;
    }
    bit(1) has_r;
    if(has_r) {
      attr_custom_coordinate r;
    }
    bit(1) has_attr_any;
    if(has_attr_any) {
      attr_any any;
    }
    object_content child0;
  }
  class element_rect {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    bit(1) has_rare;
    if(has_rare) {
      attr_custom_rare rare;
    }
    bit(1) has_fill;
    if(has_fill) {
      attr_custom_paint fill;
    }
    bit(1) has_stroke;
    if(has_stroke) {
      attr_custom_paint stroke;
    }
    attr_custom_coordinate height;
    bit(1) has_rx;
    if(has_rx) {
      attr_custom_coordinate rx;
    }
    bit(1) has_ry;
    if(has_ry) {
      attr_custom_coordinate ry;
    }
    attr_custom_coordinate width;
    bit(1) has_x;
    if(has_x) {
      attr_custom_coordinate x;
    }
    bit(1) has_y;
    if(has_y) {
      attr_custom_coordinate y;
    }
    bit(1) has_attr_any;
    if(has_attr_any) {
      attr_any any;
    }
    object_content child0;
  }
  class element_sameg {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    objectSame_content child0;
  }
  class element_sameline {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    bit(1) has_x1;
    if(has_x1) {
      attr_custom_coordinate x1;
    }
    attr_custom_coordinate x2;
    bit(1) has_y1;
    if(has_y1) {
      attr_custom_coordinate y1;
    }
    attr_custom_coordinate y2;
    objectSame_content child0;
  }
  class element_samepath {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    attr_custom_path d;
    objectSame_content child0;
  }
```

```
class element_samepathfill {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  attr_custom_path d;
  objectSame_content child0;
}
class element_samepolygon {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  attr_custom_pointSequence points;
  objectSame_content child0;
}
class element_samepolygonfill {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  attr_custom_pointSequence points;
  objectSame_content child0;
}
class element_samepolygonstroke {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  attr_custom_pointSequence points;
  objectSame_content child0;
}
class element_samerect {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  attr_custom_coordinate height;
  attr_custom_coordinate width;
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  objectSame_content child0;
}
class element_samerectfill {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  attr_custom_coordinate height;
  attr_custom_coordinate width;
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  objectSame_content child0;
}
class element_sametext {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
```

```
  bit(1) has_x;
  if(has_x) {
    attr_coordinateList x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_coordinateList y;
  }
  objectSame_content child0;
}
class element_sametextfill {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_x;
  if(has_x) {
    attr_coordinateList x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_coordinateList y;
  }
  objectSame_content child0;
}
class element_sameuse {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  objectSame_content child0;
}
class element_script {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_type;
  if(has_type) {
      attr_script type;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_set {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_attributeType;
  if(has_attributeType) {
    attr_attributeType attributeType;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
```

```
    attr_time dur;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_animFill fill;
  }
  bit(1) has_repeatCount;
  if(has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if(has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if(has_restart) {
    attr_restart restart;
  }
  bit(1) has_to;
  if(has_to) {
    attr_custom_AnimatedValue to;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) enabled;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_stop {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  attr_custom_fixed_16_8 offset;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_switch {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_text {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
```

```
    if(has_rare) {
      attr_custom_rare rare;
    }
    bit(1) has_fill;
    if(has_fill) {
      attr_custom_paint fill;
    }
    bit(1) has_stroke;
    if(has_stroke) {
      attr_custom_paint stroke;
    }
    attr_editable editable;
    bit(1) has_rotate;
    if(has_rotate) {
      attr_floatList rotate;
    }
    bit(1) has_x;
    if(has_x) {
      attr_coordinateList x;
    }
    bit(1) has_y;
    if(has_y) {
      attr_coordinateList y;
    }
    bit(1) has_attr_any;
    if(has_attr_any) {
      attr_any any;
    }
    object_content child0;
  }
  class element_tspan {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    bit(1) has_rare;
    if(has_rare) {
      attr_custom_rare rare;
    }
    bit(1) has_fill;
    if(has_fill) {
      attr_custom_paint fill;
    }
    bit(1) has_stroke;
    if(has_stroke) {
      attr_custom_paint stroke;
    }
    bit(1) has_attr_any;
    if(has_attr_any) {
      attr_any any;
    }
    object_content child0;
  }
  class element_use {
    bit(1) has_id;
    if(has_id) {
      attr_custom_ID id;
    }
    bit(1) has_rare;
    if(has_rare) {
      attr_custom_rare rare;
    }
    bit(1) has_fill;
    if(has_fill) {
      attr_custom_paint fill;
    }
    bit(1) has_stroke;
    if(has_stroke) {
      attr_custom_paint stroke;
    }
    bit(1) externalResourcesRequired;
    bit(1) has_overflow;
    if(has_overflow) {
      attr_overflow overflow;
    }
    bit(1) has_x;
    if(has_x) {
      attr_custom_coordinate x;
    }
    bit(1) has_y;
    if(has_y) {
      attr_custom_coordinate y;
    }
    bit(1) has_href;
    if(has_href) {
      attr_custom_anyURI href;
    }
```

```
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_video {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_time dur;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_height;
  if(has_height) {
    attr_custom_coordinate height;
  }
  bit(1) has_overlay;
  if(has_overlay) {
    attr_overlay overlay;
  }
  bit(1) has_preserveAspectRatio;
  if(has_preserveAspectRatio) {
    attr_preserveAspectRatio preserveAspectRatio;
  }
  bit(1) has_repeatCount;
  if(has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if(has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if (has_restart) {
    attr_restart restart;
  }
  bit(1) has_syncBehavior;
  if(has_syncBehavior) {
    attr_syncBehavior syncBehavior;
  }
  bit(1) has_syncTolerance;
  if(has_syncTolerance) {
    attr_syncTolerance syncTolerance;
  }
  bit(1) has_transformBehavior;
  if(has_transformBehavior) {
    attr_transformBehavior transformBehavior;
  }
  bit(1) has_type;
  if(has_type) {
    attr_custom_byteAlignedString type;
  }
  bit(1) has_width;
  if(has_width) {
    attr_custom_coordinate width;
  }
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_clipBegin;
  if(has_clipBegin) {
    attr_time clipBegin;
  }
  bit(1) has_clipEnd;
  if(has_clipEnd) {
    attr_time clipEnd;
```

```
  }
  bit(1) has_fullscreen;
  if (has_fullscreen) {
    bit(1) fullscreen;
  }
  bit(1) has_syncReference;
  if(has_syncReference) {
    attr_custom_anyURI syncReference;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_listener {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_defaultAction;
  if(has_defaultAction) {
    attr_defaultAction defaultAction;
  }
  bit(1) has_event;
  if(has_event) {
    attr_custom_event event;
  }
  bit(1) has_handler;
  if(has_handler) {
    attr_custom_anyURI handler;
  }
  bit(1) has_observer;
  if(has_observer) {
    attr_custom_IDREF observer;
  }
  attr_phase phase;
  bit(1) has_propagate;
  if(has_propagate) {
    attr_propagate propagate;
  }
  bit(1) has_target;
  if(has_target) {
    attr_custom_IDREF target;
  }
  bit(1) enabled;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_conditional {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) externalResourcesRequired;
  bit(1) enabled;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  updateListType_content child0;
  bit(1) opt_group;
  if (opt_group) {
    privateAttributeContainer privateAttributes;
  }
}
class updateListType_content {
     vluimsbf5 encoding-length;
     attr_custom_align encoding-align-before;
     vluimsbf5 occ0;
     for(int t=0;t<occ0+1;t++) {
          updates child0;
```

```
    }
    attr_custom_align encoding-align-after;
}
class updates {
  bit(4) ch4;
  switch(ch4){
    case 0:
      update_Add Add;
      break;
    case 1:
      update_Clean Clean;
      break;
    case 2:
      update_Delete Delete;
      break;
    case 3:
      update_Insert Insert;
      break;
    case 4:
      update_NewScene NewScene;
      break;
    case 5:
      update_RefreshScene RefreshScene;
      break;
    case 6:
      update_Replace Replace;
      break;
    case 7:
      update_Restore Restore;
      break;
    case 8:
      update_Save Save;
      break;
    case 9:
      update_SendEvent SendEvent;
      break;
    case 10:
      update_any ext;
      break;
    case 11:
      attr_custom_byteAlignedString textContent;
      break;
    default:
      break;
    }
}
class update_Add {
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_operandAttribute;
  if(has_operandAttribute) {
    attr_AttributeName operandAttribute;
  }
  bit(1) has_operandElementId;
  if(has_operandElementId) {
    attr_custom_IDREF operandElementId;
  }
  attr_custom_IDREF ref;
  bit(1) has_value;
  if(has_value) {
    attr_custom_updateValue value;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}
class update_Clean {
  attr_custom_byteAlignedString groupID;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}
class update_Delete {
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_index;
  if(has_index) {
    attr_index index;
  }
  attr_custom_IDREF ref;
  bit(1) has_attr_any;
```

```
  if(has_attr_any) {
    attr_any any;
  }
}
class update_Insert {
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_index;
  if(has_index) {
    attr_index index;
  }
  attr_custom_IDREF ref;
  bit(1) has_value;
  if(has_value) {
    attr_custom_updateValue value;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  bit(1) opt_group;
  if(opt_group) {
    updatable_elements child0;
  }
}
class updatable_elements {
  bit(1) ch4;
  switch(ch4){
    case 0:
      bit(6) ch6;
      switch(ch6){
        case 0:
          element_a a;
          break;
        case 1:
          element_animate animate;
          break;
        case 2:
          element_animate animateColor;
          break;
        case 3:
          element_animateMotion animateMotion;
          break;
        case 4:
          element_animateTransform animateTransform;
          break;
        case 5:
          element_audio audio;
          break;
        case 6:
          element_circle circle;
          break;
        case 7:
          element_defs defs;
          break;
        case 8:
          element_desc_metaData_title desc;
          break;
        case 9:
          element_ellipse ellipse;
          break;
        case 10:
          element_foreignObject foreignObject;
          break;
        case 11:
          element_g g;
          break;
        case 12:
          element_image image;
          break;
        case 13:
          element_line line;
          break;
        case 14:
          element_linearGradient linearGradient;
          break;
        case 15:
          element_desc_metaData_title metadata;
          break;
        case 16:
          element_mpath mpath;
          break;
        case 17:
          element_path path;
          break;
```

```
        case 18:
          element_polygon polygon;
          break;
        case 19:
          element_polygon polyline;
          break;
        case 20:
          element_radialGradient radialGradient;
          break;
        case 21:
          element_rect rect;
          break;
        case 22:
          element_script script;
          break;
        case 23:
          element_set set;
          break;
        case 24:
          element_stop stop;
          break;
        case 25:
          element_svg svg;
          break;
        case 26:
          element_switch switch;
          break;
        case 27:
          element_text text;
          break;
        case 28:
          element_desc_metaData_title title;
          break;
        case 29:
          element_tspan tspan;
          break;
        case 30:
          element_use use;
          break;
        case 31:
          element_video video;
          break;
        case 32:
          element_listener listener;
          break;
        }
      break;
    case 1:
      bit(3) ch61;
      switch(ch61){
        case 0:
          element_conditional conditional;
          break;
        case 1:
          element_cursorManager cursorManager;
          break;
        case 2:
          element_any extElement;
          break;
        case 3:
          privateElementContainer privateElement;
          break;
        case 4:
          element_rectClip rectClip;
          break;
        case 5:
          element_selector selector;
          break;
        case 6:
          element_simpleLayout simpleLayout;
          break;
        }
      break;
    }
}
class element_svg {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
```

```
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_baseProfile;
  if(has_baseProfile) {
    attr_custom_byteAlignedString baseProfile;
  }
  bit(1) has_contentScriptType;
  if(has_contentScriptType) {
    attr_custom_byteAlignedString contentScriptType;
  }
  bit(1) externalResourcesRequired;
  attr_custom_valueWithUnits height;
  bit(1) has_playbackOrder;
  if(has_playbackOrder) {
    attr_playbackOrder playbackOrder;
  }
  bit(1) has_preserveAspectRatio;
  if(has_preserveAspectRatio) {
    attr_preserveAspectRatio preserveAspectRatio;
  }
  bit(1) has_snapshotTime;
  if(has_snapshotTime) {
    attr_time snapshotTime;
  }
  bit(1) has_syncBehaviorDefault;
  if(has_syncBehaviorDefault) {
    attr_syncBehaviorDefault syncBehaviorDefault;
  }
  bit(1) has_syncToleranceDefault;
  if(has_syncToleranceDefault) {
    attr_syncToleranceDefault syncToleranceDefault;
  }
  bit(1) has_timelineBegin;
  if(has_timelineBegin) {
    attr_timeLineBegin timelineBegin;
  }
  bit(1) has_version;
  if(has_version) {
    attr_custom_byteAlignedString version;
  }
  bit(1) has_viewBox;
  if(has_viewBox) {
    attr_viewBox viewBox;
  }
  attr_custom_valueWithUnits width;
  bit(1) has_zoomAndPan;
  if(has_zoomAndPan) {
    attr_zoomAndPan zoomAndPan;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_cursorManager {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_x;
  if(has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if(has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class privateElementContainer {
  bit(4) codePoint;
```

```
  switch(codePoint){
    case 0:
      privateAnyXMLElement anyXML;
      break;
    case 1:
      privateOpaqueElement opaque;
      break;
    case 2:
      element_any reserved;
      break;
    default:
      attr_custom_extension ext;
      break;
  }
}
class element_rectClip {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_size;
  if(has_size) {
    attr_point size;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_selector {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_choice;
  if(has_choice) {
    attr_choice choice;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_simpleLayout {
  bit(1) has_id;
  if(has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if(has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
```

```
  }
  bit(1) has_delta;
  if(has_delta) {
    attr_point delta;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class update_NewScene {
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  element_svg child;
}
class update_RefreshScene {
  vluimsbf5 time;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  element_svg child;
}
class update_Replace {
  bit(1) has_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  bit(1) has_index;
  if(has_index) {
    attr_index index;
  }
  bit(1) has_operandAttribute;
  if(has_operandAttribute) {
    attr_AttributeName operandAttribute;
  }
  bit(1) has_operandElementId;
  if(has_operandElementId) {
    attr_custom_IDREF operandElementId;
  }
  attr_custom_IDREF ref;
  bit(1) has_value;
  if(has_value) {
    attr_custom_updateValue value;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  bit(1) opt_group;
  if(opt_group) {
    vluimsbf5 occ1;
    for(int t=0;t<occ1;t++) {
      updatable_elements child0[[t]];
    }
  }
}
class update_Restore {
  attr_custom_byteAlignedString groupID;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}
class update_Save {
  attr_custom_ElementAttributeList elementAttributeList;
  attr_custom_byteAlignedString groupID;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}
class update_SendEvent {
  attr_custom_event event;
  bit(1) has_intvalue;
  if(has_intvalue) {
    signedInt intvalue;
  }
  bit(1) has_pointvalue;
  if(has_pointvalue) {
    attr_point pointvalue;
  }
  attr_custom_IDREF ref;
```

```
  bit(1) has_stringvalue;
  if(has_stringvalue) {
    attr_custom_byteAlignedString stringvalue;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}

// AMD1 elements
// to be enclosed in an element_any with extensionID = 2

class SVGAmd1Extension {
  vluimsbf5 occ0;
  for(int t=0;t<occ0+1;t++) {
    bit(4) ch5;
    switch(ch5){
    case 0:
      element_animation animation;
      break;
    case 1:
      element_discard discard;
      break;
    case 2:
      element_font font;
      break;
    case 3:
        element_fontFace font_face;
      break;
    case 4:
        element_fontFaceSrc font_face_src;
      break;
    case 5:
        element_fontFaceUri font_face_uri;
      break;
    case 6:
      element_glyph glyph;
      break;
    case 7:
      element_handler handler;
      break;
    case 8:
        element_hkern element_hkern;
      break;
    case 9:
        element_missingGlyph missing_glyph;
      break;
    case 10:
        element_prefetch element_prefetch;
      break;
    case 11:
      element_solidColor solidColor;
      break;
    case 12:
      element_tBreak tBreak;
      break;
    case 13:
      element_textArea textArea;
      break;
    default:
      break;
    }
  }
}
class element_animation {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if (has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if (has_dur) {
    attr_time dur;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_fill;
  if (has_fill) {
    attr_animFill fill;
  }
```

```
  bit(1) has_height;
  if (has_height) {
    attr_custom_coordinate height;
  }
  bit(1) has_preserveAspectRatio;
  if (has_preserveAspectRatio) {
    attr_preserveAspectRatio preserveAspectRatio;
  }
  bit(1) has_repeatCount;
  if (has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if (has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if (has_restart) {
    attr_restart restart;
  }
  bit(1) has_syncBehavior;
  if (has_syncBehavior) {
    attr_syncBehavior syncBehavior;
  }
  bit(1) has_syncTolerance;
  if (has_syncTolerance) {
    attr_syncTolerance syncTolerance;
  }
  bit(1) has_type;
  if (has_type) {
    attr_custom_byteAlignedString type;
  }
  bit(1) has_width;
  if (has_width) {
    attr_custom_coordinate width;
  }
  bit(1) has_x;
  if (has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if (has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_clipBegin;
  if (has_clipBegin) {
    attr_time clipBegin;
  }
  bit(1) has_clipEnd;
  if (has_clipEnd) {
    attr_time clipEnd;
  }
  bit(1) has_syncReference;
  if (has_syncReference) {
    attr_custom_anyURI syncReference;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_discard {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if (has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
```

```
  object_content child0;
}
class element_font {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_horiz_adv_x;
  if(has_horiz_adv_x) {
    attr_custom_fixed_16_8 horiz_adv_x;
  }
  bit(1) has_horiz_origin_x;
  if(has_horiz_origin_x) {
    attr_custom_fixed_16_8 horiz_origin_x;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_fontFace {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_accent_height;
  if(has_accent_height) {
    attr_custom_fixed_16_8 accent_height;
  }
  bit(1) has_alphabetic;
  if (has_alphabetic) {
    attr_custom_fixed_16_8 alphabetic;
  }
  bit(1) has_ascent;
  if (has_ascent) {
    attr_custom_fixed_16_8 ascent;
  }
  bit(1) has_bbox;
  if (has_bbox) {
    attr_custom_byteAlignedString bbox;
  }
  bit(1) has_cap_height;
  if(has_cap_height) {
    attr_custom_fixed_16_8 cap_height;
  }
  bit(1) has_descent;
  if (has_descent) {
    attr_custom_fixed_16_8 descent;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_font_family;
  if(has_font_family) {
    attr_custom_byteAlignedString font_family;
  }
  bit(1) has_font_stretch;
  if(has_font_stretch) {
    attr_custom_byteAlignedString font_stretch;
  }
  bit(1) has_font_style;
  if(has_font_style) {
    attr_custom_byteAlignedString font_style;
  }
  bit(1) has_font_variant;
  if(has_font_variant) {
    attr_custom_byteAlignedString font_variant;
  }
  bit(1) has_font_weight;
  if(has_font_weight) {
    attr_custom_byteAlignedString font_weight;
  }
  bit(1) has_hanging;
  if (has_hanging) {
    attr_custom_fixed_16_8 hanging;
  }
  bit(1) has_ideographic;
  if (has_ideographic) {
    attr_custom_fixed_16_8 ideographic;
```

```
  }
  bit(1) has_mathematical;
  if (has_mathematical) {
    attr_custom_fixed_16_8 mathematical;
  }
  bit(1) has_overline_position;
  if(has_overline_position) {
    attr_custom_fixed_16_8 overline_position;
  }
  bit(1) has_overline_thickness;
  if(has_overline_thickness) {
    attr_custom_fixed_16_8 overline_thickness;
  }
  bit(1) has_panose_1;
  if(has_panose_1) {
    attr_custom_byteAlignedString panose_1;
  }
  bit(1) has_slope;
  if (has_slope) {
    attr_custom_fixed_16_8 slope;
  }
  bit(1) has_stemh;
  if (has_stemh) {
    attr_custom_fixed_16_8 stemh;
  }
  bit(1) has_stemv;
  if (has_stemv) {
    attr_custom_fixed_16_8 stemv;
  }
  bit(1) has_strikethrough_position;
  if(has_strikethrough_position) {
    attr_custom_fixed_16_8 strikethrough_position;
  }
  bit(1) has_strikethrough_thickness;
  if(has_strikethrough_thickness) {
    attr_custom_fixed_16_8 strikethrough_thickness;
  }
  bit(1) has_underline_position;
  if(has_underline_position) {
    attr_custom_fixed_16_8 underline_position;
  }
  bit(1) has_underline_thickness;
  if(has_underline_thickness) {
    attr_custom_fixed_16_8 underline_thickness;
  }
  bit(1) has_unicode_range;
  if(has_unicode_range) {
    attr_custom_byteAlignedString unicode_range;
  }
  bit(1) has_units_per_em;
  if(has_units_per_em) {
    attr_custom_fixed_16_8 units_per_em;
  }
  bit(1) has_widths;
  if (has_widths) {
    attr_custom_byteAlignedString widths;
  }
  bit(1) has_x_height;
  if(has_x_height) {
    attr_custom_coordinate x_height;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_fontFaceSrc {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child1;
}
class element_fontFaceUri {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
```

```
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_glyph {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_arabic_form;
  if(has_arabic_form) {
    attr_custom_byteAlignedString arabic_form;
  }
  bit(1) has_d;
  if (has_d) {
    attr_custom_path d;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_glyph_name;
  if(has_glyph_name) {
    attr_custom_byteAlignedString glyph_name;
  }
  bit(1) has_horiz_adv_x;
  if(has_horiz_adv_x) {
    attr_custom_fixed_16_8 horiz_adv_x;
  }
  bit(1) has_lang;
  if (has_lang) {
    attr_custom_byteAlignedString lang;
  }
  bit(1) has_unicode;
  if (has_unicode) {
    attr_custom_byteAlignedString unicode;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_handler {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) externalResourcesRequired;
  bit(1) has_type;
  if (has_type) {
    attr_script type;
  }
  bit(1) has_event;
  if (has_event) {
    attr_custom_event event;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_hkern {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
```

```
  }
  bit(1) has_g1;
  if (has_g1) {
    attr_custom_byteAlignedString g1;
  }
  bit(1) has_g2;
  if (has_g2) {
    attr_custom_byteAlignedString g2;
  }
  bit(1) has_k;
  if (has_k) {
    attr_custom_fixed_16_8 k;
  }
  bit(1) has_u1;
  if (has_u1) {
    attr_custom_byteAlignedString u1;
  }
  bit(1) has_u2;
  if (has_u2) {
    attr_custom_byteAlignedString u2;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_missingGlyph {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_d;
  if (has_d) {
    attr_custom_path d;
  }
  bit(1) has_horiz_adv_x;
  if(has_horiz_adv_x) {
    attr_custom_fixed_16_8 horiz_adv_x;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_prefetch {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_bandwidth;
  if (has_bandwidth) {
    vluimsbf5 bandwidth;
  }
  bit(1) has_mediaCharacterEncoding;
  if (has_mediaCharacterEncoding) {
    attr_custom_byteAlignedString mediaCharacterEncoding;
  }
  bit(1) has_mediaContentEncodings;
  if (has_mediaContentEncodings) {
    attr_custom_byteAlignedString mediaContentEncodings;
  }
  bit(1) has_mediaSize;
  if (has_mediaSize) {
    vluimsbf5 mediaSize;
  }
  bit(1) has_mediaTime;
  if (has_mediaTime) {
    vluimsbf5 mediaTime;
  }
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
```

```
  object_content child0;
}
class element_solidColor {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if(has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if(has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_tBreak {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child1;
}
class element_textArea {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_fill;
  if (has_fill) {
    attr_custom_paint fill;
  }
  bit(1) has_stroke;
  if (has_stroke) {
    attr_custom_paint stroke;
  }
  bit(1) has_editable;
  if(has_editable) {
    attr_editable editable;
  }
  bit(1) has_height;
  if (has_height) {
    attr_custom_coordinate height;
  }
  bit(1) has_width;
  if (has_width) {
    attr_custom_coordinate width;
  }
  bit(1) has_x;
  if (has_x) {
    attr_custom_coordinate x;
  }
  bit(1) has_y;
  if (has_y) {
    attr_custom_coordinate y;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class update_Activate {
  attr_custom_IDREF ref;
  bit(1) has_attr_any;
  if (has_attr_any) {
```

```
    attr_any any;
  }
}
class update_Deactivate {
  attr_custom_IDREF ref;
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
}
class update_ReleaseResource {
  attr_custom_anyURI ref;
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
}
class LASeRAmd1Extension {
  vluimsbf5 occ1;
  for(int t=0;t<occ1+1;t++) {
    bit(2) ch5;
    switch(ch5){
      case 0:
        element_animateScroll animateScroll;
        break;
      case 1:
        element_setScroll setScroll;
        break;
      case 2:
        element_streamSource streamSource;
        break;
      case 3:
        element_updates updates;
        break;
      }
  }
}
class element_animateScroll {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if (has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_by;
  if (has_by) {
    attr_scrollStop by;
  }
  bit(1) has_delayAtEnd;
  if (has_delayAtEnd) {
    attr_time delayAtEnd;
  }
  bit(1) has_delayAtStart;
  if (has_delayAtStart) {
    attr_time delayAtStart;
  }
  bit(1) has_direction;
  if (has_direction) {
    attr_direction direction;
  }
  bit(1) has_dur;
  if (has_dur) {
    attr_time dur;
  }
  bit(1) has_fill;
  if (has_fill) {
    attr_animFill fill;
  }
  bit(1) has_from;
  if (has_from) {
    attr_scrollStop from;
  }
  bit(1) has_repeatCount;
  if (has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if (has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
```

```
  if (has_restart) {
    attr_restart restart;
  }
  bit(1) has_speed;
  if (has_speed) {
    attr_time speed;
  }
  bit(1) has_to;
  if (has_to) {
    attr_scrollStop to;
  }
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_setScroll {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if (has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_direction;
  if (has_direction) {
    attr_direction direction;
  }
  bit(1) has_increment;
  if (has_increment) {
    attr_scrollStop increment;
  bit(1) has_to;
  if (has_to) {
    attr_scrollStop to;
  }
  bit(1) has_href;
  if (has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class element_streamSource {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_height;
  if (has_height) {
    attr_custom_coordinate height;
  }
  bit(1) has_mode;
  if (has_mode) {
    attr_streamSourceMode mode;
  }
  bit(1) has_sourceIndex;
  if (has_sourceIndex) {
    vluimsbf5 sourceIndex;
  }
  attr_sources sources;
  bit(1) has_width;
  if (has_width) {
    attr_custom_coordinate width;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
```

```
class element_updates {
  bit(1) has_id;
  if (has_id) {
    attr_custom_ID id;
  }
  bit(1) has_rare;
  if (has_rare) {
    attr_custom_rare rare;
  }
  bit(1) has_begin;
  if(has_begin) {
    attr_smil_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_time dur;
  }
  bit(1) has_repeatCount;
  if(has_repeatCount) {
    attr_repeatCount repeatCount;
  }
  bit(1) has_repeatDur;
  if(has_repeatDur) {
    attr_repeatDur repeatDur;
  }
  bit(1) has_restart;
  if (has_restart) {
    attr_restart restart;
  }
  bit(1) has_syncBehavior;
  if(has_syncBehavior) {
    attr_syncBehavior syncBehavior;
  }
  bit(1) has_syncTolerance;
  if(has_syncTolerance) {
    attr_syncTolerance syncTolerance;
  }
  bit(1) has_href;
  if(has_href) {
    attr_custom_anyURI href;
  }
  bit(1) has_clipBegin;
  if(has_clipBegin) {
    attr_time clipBegin;
  }
  bit(1) has_clipEnd;
  if(has_clipEnd) {
    attr_time clipEnd;
  }
  bit(1) has_flow;
  if(has_flow) {
    // Enumeration: any{0} complete{1} segment{2}
    bit(2) flow;
  }
  bit(1) has_syncReference;
  if(has_syncReference) {
    attr_custom_anyURI syncReference;
  }
  bit(1) has_security;
  if(has_security) {
    // Enumeration: new{0} parent{1}
    bit(2)security;
  }
  bit(1) has_attr_any;
  if (has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class LASeRAmd1Command {
  vluimsbf5 occ3;
  for(int t=0;t<occ3+1;t++) {
    bit(2) ch5;
    switch(ch5){
    case 0:
      privateElementContainer privateElementContainer; // to allow anyXML at the updates level (top level of the AU)
      break;
    case 1:
      update_Activate Activate;
      break;
    case 2:
      update_Deactivate Deactivate;
      break;
    case 3:
      update_ReleaseResource ReleaseResource;
      break;
    }
```

```
  }
}

//**************************************************************************************************
// start specific classes
//**************************************************************************************************

class LASeRUnit {
  LASeRUnitHeader h;
  initialisations c;
  vluimsbf5 occ1;
  for(int t=0;t<occ1+1;t++) {
    updates child0;
  }
  bit(1) opt_group;
  if(opt_group) {
    attr_custom_extension ext;
  }
}
class LASeRUnitHeader {
  bit(1) resetEncodingContext;
  bit(1) opt_group;
  if(opt_group) {
    attr_custom_extension ext;
  }
}
class initialisations {
  colorInitialisation c;
  fontInitialisation f;
  privateDataIdentifierInitialisation p;
  anyXMLInitialisation a;
  extendedInitialisation e;
}


//**************************************************************************************************
// extensions
//**************************************************************************************************

//for elements
class element_any {
  uint(extensionIDBits) extensionID;
  vluimsbf5 len; //length in bits
  if (extensionID == 2) { //LASeR AMD1
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ2;
    for(int t=0;t<occ2+1;t++) {
      bit(2) ch5;
      switch(ch5){
        case 0:
          SVGAmd1Extension SVGAmd1Extension;
          break;
        case 2:
          LASeRAmd1Extension LASeRAmd1Extension;
          break;
      }
    }
  } else {
    bit[len] toSkip;
  }
}


class update_any {
  uint(extensionIDBits) extensionID;
  vluimsbf5 len; //length in bits
  if (extensionID == 2) { //LASeR AMD1
    const vluimsbf5 reserved = 87; // or "const bit(10) reserved = 599;"
    vluimsbf5 occ2;
    for(int t=0;t<occ2+1;t++) {
      const bit(2) reserved = 1;
      LASeRAmd1Command LASeRAmd1Command;
    }
  } else {
    bit[len] toSkip;
  }
}

//for attributes
class attr_any {
  do {
    uint(extensionIDBits) extensionID;
    vluimsbf5 len; //length in bits
    if (extensionID == 2) { //LASeR AMD1
      bit(1) hasRequiredFonts;
      if (hasRequiredFonts) {
        attr_custom_byteAlignedString requiredFonts;
```

```
      }
      bit(1) hasX;
      if (hasX) {
        attr_custom_coordinate x;
      }
      bit(1) hasY;
      if (hasY) {
        attr_custom_coordinate y;
      }
    } else {
      bit[len] toSkip;
      bit(1) hasNextExtension;
    }
  } while (hasNextExtension);
}

//*************************************************************************************************
// LASeRHeader
//*************************************************************************************************

class LASeRHeader {
  uint(8) profile;
  uint(8) level;
  bit(3) reserved;
  bit(2) pointsCodec;
  bit(4) pathComponents;
  bit(1) useFullRequestHost;
  bit(1) hasTimeResolution;
  if (hasTimeResolution) {
    uint(16) timeResolution;
  }
  bit(4) colorComponentBits_minus_1;
  colorComponentBits = colorComponentBits_minus_1 + 1;
  bit(4) resolution;
  bit(5) coordBits;
  bit(4) scaleBits_minus_coordBits;
  bit(1) newSceneIndicator;
  bit(3) reserved;
  // extensionIDBits defines the number of bits of extension tags
  bit(4) extensionIDBits;
  bit(1) hasExtConfiguration;
  if (hasExtConfiguration) {
    vluimsbf5 len;
    byte[len] extConfiguration; //extConfiguration is defined in Table 10
  }
  bit(1) hasExtension;
  if (hasExtension) {
    attr_custom_extension ext;
  }
}

//*************************************************************************************************
// Initialisation codecs
//*************************************************************************************************

int colorIndex = -1;

class colorInitialisation {
  bit(1) hasColors;
  if (hasColors) {
    // a color table in front of each AU
    vluimsbf5 nbColors;
    for (int i = 0; i < nbColors; i++) {
      colorIndex = colorIndex + 1;
      uint(colorComponentBits) red[[colorIndex]];
      uint(colorComponentBits) green[[colorIndex]];
      uint(colorComponentBits) blue[[colorIndex]];
    }
    colorIndexBits = log2sup(colorIndex);
  }
}

int fontIndex = -1;

class fontInitialisation {
  bit(1) hasFonts;
  if (hasFonts) {
    // a font table in front of each AU
    vluimsbf5 nbFonts;
    for (int i = 0; i < nbFonts; i++) {
      fontIndex = fontIndex + 1;
      attr_custom_byteAlignedString font[[fontIndex]];
    }
    fontIndexBits = log2sup(fontIndex);
  }
}
```

```
int privateDataIdentifierIndex = -1;

class privateDataIdentifierInitialisation {
  bit(1) hasPrivateDataIdentifiers;
  if (hasPrivateDataIdentifiers) {
    // a privateDataIdentifiertable in front of each AU
    vluimsbf5 nbPrivateDataIdentifiers;
    for (int i = 0; i < nbPrivateDataIdentifiers; i++) {
      privateDataIdentifierIndex = privateDataIdentifierIndex + 1;
      // the purpose of these strings is to ensure non collision between different private data
      // examples are URNs, uuids, ...
      attr_custom_byteAlignedString privateDataIdentifier[[privateDataIdentifierIndex]];
    }
    privateDataIdentifierIndexBits = log2sup(privateDataIdentifierIndex);
  }
}

int tagIndex = -1;

class anyXMLInitialisation {
  bit(1) hasTags;
  if (hasTags) {
    // a tag table in front of each AU
    vluimsbf5 nbTags;
    for (int i = 0; i < nbTags; i++) {
      if (i == 0) {
        // tag 0 for use for priv. attrs on LASeR elements
        bit(1) hasAttrs;
        if (hasAttrs) {
          // the first bin is reserved for private attributes of LASeR elements
          // and to attributes with a privateDataIdentifier different from their parent element
          vluimsbf5 nbAttrNames[[0]];
          for (int t = 0; t < nbAttrNames[[0]]; t++) {
            uint(privateDataIdentifierIndexBits) privateDataIdentifierIndex[[0]][[t]];
            attr_custom_byteAlignedString attrName[[0]][[t]];
          }
        }
      } else {
        tagIndex = tagIndex + 1;
        uint(privateDataIdentifierIndexBits) privateDataIdentifierIndex[[i]];
        attr_custom_byteAlignedString tag[[i]];
        bit(1) hasAttrs;
        if (hasAttrs) {
          // each private element tag has a bin for private attributes
          vluimsbf5 nbAttrNames[[tagIndex]];
          for (int t = 0; t < nbAttrNames[[tagIndex]]; t++) {
            attr_custom_byteAlignedString attrName[[tagIndex]][[t]];
          }
        }
      }
    }
    tagIndexBits = log2sup(tagIndex);
  }
}

class extendedInitialisation {
  // stringIDTable: external string ID references
  vluimsbf5 countG;
  for (int i = 0; i < countG; i++) {
    vluimsbf5 binaryIdForThisStringID[[i]];
    attr_custom_byteAlignedString stringID[[i]];
  }
  bit(1) hasExtension;
  if (hasExtension) {
    vluimsbf5 len;
    // globalStreamTable: external global streamID references
    vluimsbf5 countGS;
    for (int i = 0; i < count; i++) {
      vluimsbf5 localStreamIdForThisGlobal[[i]];
      byteAlignedStringClass globalName[[i]];
    }
    bit[len2] remainingData;
    // len2 is defined implicitly as: len □$(H s□(Bizeof(globalStreamTable)
  }
}
```

12.2.2. Generic Data Types

```
class object_content {
  bit(1) opt_group;
  if (opt_group) {
    privateAttributeContainer privateAttributes;
  }
  bit(1) opt_group1;
  if (opt_group1) {
    vluimsbf5 occ1;
    for(int t=0;t<occ1;t++) {
      elements child0[[t]];
    }
  }
}
class attr_AttributeName {
  bit(1) choice;
  switch(choice) {
    case 0:
      // Enumeration: a.target{0} accumulate{1} additive{2} audio-level{3} bandwidth{4} begin{5} calcMode{6}
children{7} choice{8} clipBegin{9} clipEnd{10} color{11} color-rendering{12} cx{13} cy{14} d{15} delta{16}
display{17} display-align{18} dur{19} editable{20} enabled{21} end{22} event{23} externalResourcesRequired{24}
fill{25} fill-opacity{26} fill-rule{27} focusable{28} font-family{29} font-size{30} font-style{31} font-variant{32}
font-weight{33} fullscreen{34} gradientUnits{35} handler{36} height{37} image-rendering{38} keyPoints{39}
keySplines{40} keyTimes{41} line-increment{42} listener.target{43} mediaCharacterEncoding{44}
mediaContentEncodings{45} mediaSize{46} mediaTime{47} nav-down{48} nav-down-left{49} nav-down-right{50} nav-left{51}
nav-next{52} nav-prev{53} nav-right{54} nav-up{55} nav-up-left{56} nav-up-right{57} observer{58} offset{59}
opacity{60} overflow{61} overlay{62} path{63} pathLength{64} pointer-events{65} points{66} preserveAspectRatio{67}
r{68} repeatCount{69} repeatDur{70} requiredExtensions{71} requiredFeatures{72} requiredFormats{73} restart{74}
rotate{75} rotation{76} rx{77} ry{78} scale{79} shape-rendering{80} size{81} solid-color{82} solid-opacity{83} stop-
color{84} stop-opacity{85} stroke{86} stroke-dasharray{87} stroke-dashoffset{88} stroke-linecap{89} stroke-
linejoin{90} stroke-miterlimit{91} stroke-opacity{92} stroke-width{93} svg.height{94} svg.width{95} syncBehavior{96}
syncBehaviorDefault{97} syncReference{98} syncTolerance{99} syncToleranceDefault{100} systemLanguage{101} text-
align{102} text-anchor{103} text-decoration{104} text-display{105} text-rendering{106} textContent{107}
transform{108} transformBehavior{109} translation{110} vector-effect{111} viewBox{112} viewport-fill{113} viewport-
fill-opacity{114} visibility{115} width{116} x{117} x1{118} x2{119} xlink:actuate{120} xlink:arcrole{121}
xlink:href{122} xlink:role{123} xlink:show{124} xlink:title{125} xlink:type{126} xml:base{127} xml:lang{128} y{129}
y1{130} y2{131} zoomAndPan{132}
      bit(8) AttributeName;
      break;
  case 1:
    int max=+2;
    for (int t=0;t<max;t++) {
    vluimsbf5 item[[t]];
    }
    break;
  }
}
class attr_accumulate {
    // Enumeration: none{0} sum{1}
    bit(1) accumulate;
}
class attr_additive {
    // Enumeration: replace{0} sum{1}
    bit(1) additive;
}
class attr_calcMode {
    // Enumeration: discrete{0} linear{1} paced{2} spline{3}
    bit(2) calcMode;
}
class attr_attributeType {
  // Enumeration: CSS{0} XML{1} auto{2}
  bit(2) attributeType;
}
class attr_smil_times {
   bit(1) choice;
   switch(choice) {
   case 0:
       vluimsbf5 max;
       for(int t=0;t<max;t++) {
      custom_smil_time item[[t]];
       }
       break;
   case 1:
       // Enumeration: indefinite {0}
       break;
   }
}
class attr_time {
  bit(1) choice;
  switch(choice) {
    case 0:
      signedInt int0;
      break;
    case 1:
      // Enumeration: auto{0} indefinite{1} media{2} none{3}
```

```
    bit(2) time;
    break;
    default:
      break;
  }
}
class signedInt {
  bit(1) sign; // 1 is negative
  vluimsbf5 value;
}
class attr_animFill {
    // Enumeration: freeze{0} remove{1}
    bit(1) animFill;
}
class attr_repeatCount {
    bit(1) choice;
    switch(choice) {
      case 0:
        attr_custom_fixed_16_8 fixed16_8Type0;
        break;
      case 1:
        // Enumeration: indefinite{0}
        break;
      default:
        break;
      }
}
class attr_repeatDur {
    bit(1) choice;
    switch(choice) {
      case 0:
        vluimsbf5 dur;
        break;
      case 1:
        // Enumeration: indefinite{0}
        break;
      default:
        break;
      }
}
class attr_restart {
    // Enumeration: always{0} never{1} whenNotActive{2}
    bit(2) restart;
}
class attr_floatList {
  vluimsbf5 max;
  for (int t=0;t<max;t++) {
    attr_custom_fixed_16_8 item[[t]];
  }
}
class attr_rotate {
    bit(1) choice;
    switch(choice) {
      case 0:
        attr_custom_fixed_16_8 fixed16_8Type0;
        break;
      case 1:
          // Enumeration: auto{0} auto-reverse{1}
          bit(1) rotate;
        break;
      default:
        break;
      }
}
class attr_rotscatra {
    // Enumeration: rotate{0} scale{1} skewX{2} skewY{3} translate{4}
    bit(3) rotscatra;
}
class attr_syncBehavior {
    // Enumeration: canSlip{0} default{1} independent{2} locked{3}
    bit(2) syncBehavior;
}
class attr_syncTolerance {
    bit(1) choice;
    switch(choice) {
      case 0:
        vluimsbf5 vluimsbf50;
        break;
      case 1:
          // Enumeration: default{0}
        break;
      }
}
class attr_preserveAspectRatio {
    bit(1) choice;
    switch(choice) {
      case 0:
```

```
          bit(1) choice;
          switch(choice) {
            case 0:
              // Enumeration: none{0} xMaxYMax{1} xMaxYMid{2} xMaxYMin{3} xMidYMax{4} xMidYMid{5} xMidYMin{6}
xMinYMax{7} xMinYMid{8} xMinYMin{9}
              bit(4) preserveAspectRatio;
              break;
            case 1:
              // Enumeration: _reserved{0} defer xMaxYMax{1} defer xMaxYMid{2} defer xMaxYMin{3} defer xMidYMax{4}
defer xMidYMid{5} defer xMidYMin{6} defer xMinYMax{7} defer xMinYMid{8} defer xMinYMin{9}
              bit(4) preserveAspectRatio;
              break;
            }
        break;
      case 1:
          // Enumeration: reserved{0}
          bit(5) preserveAspectRatio;
        break;
      }
}
class attr_transformBehavior {
    // Enumeration: geometric{0} pinned{1} pinned_180{2} pinned_270{3} pinned_90{4}
    bit(4) transformBehavior;
}
class attr_gradientUnits {
    // Enumeration: objectBoundingBox{0} userSpaceOnUse{1}
    bit(1) gradientUnits;
}
class attr_editable {
    // Enumeration: none{0} simple{1}
    bit(1) editable;
}
class objectSame_content {
  bit(1) opt_group;
  if (opt_group) {
    vluimsbf5 occ0;
    for(int t=0;t<occ0;t++) {
      elements child0[[t]];
    }
  }
}
class attr_script {
    bit(1) choice;
    switch(choice) {
      case 0:
        attr_custom_byteAlignedString string0;
        break;
      case 1:
          // Enumeration: application/ecmascript{0} application/jar-archive{1}
          bit(1) script;
        break;
      }
}
class attr_overflow {
    // Enumeration: visible{0}
    bit(2) overflow;
}
class attr_overlay {
    bit(1) choice;
    switch(choice) {
      case 0:
        attr_custom_extension overlayExType0;
        break;
      case 1:
        // Enumeration: none{0} top{1}
        bit(1) overlay;
        break;
      default:
        break;
      }
}
class attr_defaultAction {
    // Enumeration: cancel{0} perform{1}
    bit(1) defaultAction;
}
class attr_phase {
    // Enumeration: default{0}
    bit(1) phase;
}
class attr_propagate {
    // Enumeration: continue{0} stop{1}
    bit(1) propagate;
}
class attr_index {
  vluimsbf5 value;
}
class attr_playbackOrder {
```

```
    // Enumeration: all{0} forwardOnly{1}
    bit(1) playbackOrder;
}
class attr_syncBehaviorDefault {
    // Enumeration: canSlip{0} independent{1} inherit{2} locked{3}
    bit(2) syncBehaviorDefault;
}
class attr_syncToleranceDefault {
    bit(1) choice;
    switch(choice) {
      case 0:
        vluimsbf5 vluimsbf50;
        break;
      case 1:
          // Enumeration: inherit{0}
        break;
      default:
        break;
      }
}
class attr_timeLineBegin {
    // Enumeration: onLoad{0} onStart{1}
    bit(1) timeLineBegin;
}
class attr_viewBox {
  int max=+4;
  for (int t=0;t<max;t++) {
    attr_custom_fixed_16_8 item[[t]];
  }
}
class attr_zoomAndPan {
    // Enumeration: disable{0} magnify{1}
    bit(1) zoomAndPan;
}
class attr_point {
  int max=+2;
  for (int t=0;t<max;t++) {
    attr_custom_coordinate item[[t]];
  }
}
class attr_choice {
    bit(1) choice;
    switch(choice) {
      case 0:
        bit(8) value;
        break;
      case 1:
          // Enumeration: all{0} none{1}
          bit(1) choice;
        break;
      }
}
class attr_coordinateList {
  vluimsbf5 len;
  for (int t = 0; t < len; t++) {
    attr_custom_coordinate coord[[t]];
  }
}

//AMD1
class attr_scrollStop {
  bit(1) choice;
  switch(choice) {
    case 0:
      attr_custom_IDREF IDREF0;
      break;
    case 1:
      int max=+2;
      for (int t=0;t<max;t++) {
        attr_custom_fixed_16_8 item[[t]];
      }
      break;
  }
}
class attr_direction {
  // Enumeration: down{0} left{1} right{2} up{3}
  bit(4) direction;
}
class attr_streamSourceMode {
  // Enumeration: keepOld{0} playList{1} replace{2} useOld{3}
    bit(3) streamSourceMode;
}
class attr_sources {
  vluimsbf5 max;
  for (int t=0;t<max;t++) {
    attr_custom_anyURI item[[t]];
  }
```

```
}
```

12.2.3. Specific Data Types

12.2.3.1. ID

12.2.3.2. Syntax

```
class attr_custom_ID {
    vluimsbf5 ID;
    bit(1) reserved;
    if (reserved) {
        vluimsbf5 len;
        bit[len] reserved;
    }
}
```

12.2.3.3. Semantics

This class allocates a number for an id.

12.2.3.4. IDRef

12.2.3.5. Syntax

```
class attr_custom_IDREF {
    vluimsbf5 href;
    bit(1) reserved;
    if (reserved) {
        vluimsbf5 len;
        bit[len] reserved;
    }
}
```

12.2.3.6. Semantics

This class allows pointing to the integer value defined by the ID_class.

12.2.3.7. AnyURI

12.2.3.8. Syntax

```
class attr_custom_anyURI {
    bit(1) hasUri;
    if (hasUri) {
        attr_custom_byteAlignedString uri;
        bit(1) hasData; // for a data URL, the actual data part is sent below, the header is sent in the uri field
        if (hasData) {
            vluimsbf5 len;
            byte[len] data;
        }
    }
    bit(1) hasID;
    if (hasID) {
        attr_custom_IDREF idref;
    }
    bit(1) hasStreamID;
    if (hasStreamID) {
        attr_custom_IDREF ref;
    }
}
```

12.2.3.9. Semantics

This class allows the encoding of three forms of URI usable in LASeR: string, stream ID and element ID.

12.2.3.10. Color

12.2.3.11. Syntax

```
class attr_custom_paint {
  bit(1) hasIndex;
  if (hasIndex) {
    uint(colorIndexBits) color0;
  } else {
      bit(2) ch2;
      switch(ch2) {
        case 0: // enum
         //Enumeration: inherit{0} currentColor{1} none{2}
        bit(2) color;
          break;
        case 1: // URI
              attr_custom_anyURI uri;
          break;
        case 2: // System Paint Server
          attr_custom_byteAlignedString serverName;
          break;
        case 3:
            attr_custom_extension colorExType0; // extensibility
          break;
      }
  }
}
```

12.2.3.12. Semantics

This class allows pointing to a color. The `colorIndexBits` value shall be initialised using the initialisation parameters as defined in subclause 6.6.2.3. The value for each color shall be initialised using the colorInitialisation class in the LASeRUnit, whose syntax is given below in subclause 12.2.

12.2.3.13. Matrix

12.2.3.14. Syntax

```
class custom_matrix {
    bit(1) isNotMatrix;
    if (isNotMatrix) {
      bit(1) isRef; // modification for the encoding of ref(svg[,x,y])
      if (isRef) {
        bit(1) hasXY;
        if (hasXY) {
          attr_custom_fixed_16_8 valueX;
          attr_custom_fixed_16_8 valueY;
        }
      } else {
        attr_custom_extension ext;
      }
    } else {
      bit(1) xx_yy_present;
      if (xx_yy_present) {
          uint(coordBits+scaleBits_minus_coordBits) xx;
          uint(coordBits+scaleBits_minus_coordBits) yy;
      }
      bit(1) xy_yx_present;
      if (xy_yx_present) {
          uint(coordBits+scaleBits_minus_coordBits) xy;
          uint(coordBits+scaleBits_minus_coordBits) yx;
      }
      bit(1) xz_yz_present;
      if (xz_yz_present) {
          uint(coordBits+scaleBits_minus_coordBits) xz;
          uint(coordBits+scaleBits_minus_coordBits) yz;
      }
    }
}
```

12.2.3.15. Semantics

This class decodes a matrix value. The `coordBits` and `scaleBits` values shall be initialised using the initialisation parameters as defined in subclause 6.6.2.3.

12.2.3.16. Fraction

12.2.3.17. Syntax

```
class attr_custom_0to1float {
    uint(8) quantifiedValue; // uniform quantization of a float between 0 and 1
}
```

12.2.3.18. Semantics

This class decodes an opacity value. This class is a `UniformQuantizer` where v_{min} = 0 v_{max} = 1 and nbits = 8 (see UniformQuantizer advanced optimised decoder in [ISO/IEC 23001-1]**Erreur ! Source du renvoi introuvable.**.)

12.2.3.19. Path

12.2.3.20. Syntax

```
class attr_custom_path {
    attr_custom_pointSequence seq;
    vluimsbf5 nbOfTypes;
    for (int i = 0; i < nbOfTypes; i++) {
        // Enumeration: "pathSegmentTypes" C{0} H{1} L{2} M{3} Q{4} S{5} T{6} V{7} Z{8} c{9} h{10} l{11} m{12} q{13}
s{14} t{15} v{16} z{17}
        uint(5) type[[i]];
    }
}
```

12.2.3.21. Semantics

The class `attr_custom_path` decodes a path value.

The decoding of a path value (e.g. the d attribute of a path element) is achieved by decoding a sequence of points and a list of types. The sequence of point is decoded as specified in 13.2.3.8. The decoded point coordinates are absolute values. When decoding the list of types, an implicit MoveTo shall be inserted before the first decoded type.

A decoder can use the decoded coordinate values directly for rendering by changing relative drawing types (a.k.a drawing instructions) to their absolute counterpart (e.g. changing ‘c’ to ‘C’). A decoder may reconstruct an SVG path, exactly as it was in the original document, by walking in the list of drawing types and retrieving the appropriate number of coordinates according to the drawing type. For relative types (e.g. ‘c’, ‘l’), the associated coordinate values may be restored to their relative values using the previous coordinate values.

12.2.3.22. PointSequence

12.2.3.23. Syntax

```
class attr_custom_pointSequence {
    vluimsbf5 nbPoints;
    uint(1) flag;
    if (flag == 0) {
        if (nbPoints < 3) {
            uint(5) bits;
            for (int i = 0; i < nbPoints; i++) {
                uint(bits) x[[i]];
                uint(bits) y[[i]];
            }
        } else {
            uint(5) bits;
            uint(bits) x[0];
            uint(bits) y[0];
            uint(5) bitsx;
            uint(5) bitsy;
            for (int i = 1; i < nbPoints; i++) {
                uint(bitsx) dx;
                uint(bitsy) dy;
                x[i] = dx + x[i-1];
                y[i] = dy + y[i-1];
            }
        }
    } else {
      if (pointsCodec == 0) { // pointsCodec is a LASeRHeader attribute
        uint(4) kvalue;
        uint(5) bits;
        uint(bits) x[0];
        uint(bits) y[0];
        int XMvalue,YMvalue = 0;
        int CodeNum = 0;
        int Diff =0;
        for(int i=1; i < nbPoints; i++) {
            // to calculate X point
            do {
                bit(1) bitX;
                XMvalue ++;
            } while (bitX == 0);
            const bit(1) endX = 1;
            uint(XMvalue+kvalue) INFO_dx;
            CodeNum = GetCodeNum(kvalue, XMvalue, INFO_dx);
            Diff = GetDiff(CodeNum);
            x[i] = x[i-1] + Diff;
            // to calculate Y point
            do {
                bit(1) bitY;
                YMvalue ++;
            } while (bitY == 0);
            const bit(1) endY = 1;
            uint(YMvalue+kvalue) INFO_dy;
            CodeNum = GetCodeNum(kvalue, YMvalue, INFO_dy);
            Diff = GetDiff(CodeNum);
            y[i] = y[i-1] + Diff;
        }
    } else {
        attr_custom_extension ext;
    }
  }
}

uint GetDiff(int codeNum){
      int diff;
      if (codeNum == 0) {
          diff = 0;
          return diff;
      } else {
            if ((codeNum%2) == 0 ) {
                diff = -codeNum/2;
                return diff;
            }
            else {
                diff = (codeNum+1)/2;
                return diff;
            }
      }
}

uint GetCodeNum(int k, int Mvalue, int INFO){
    return 2^(k+Mvalue) + INFO - 2^k ;
}
```

12.2.3.24. Semantics of the SDL elements

- flag - Flag indicating FL encoding (flag = 0) or EG encoding (flag = 1).

- kvalue - Parameter for EG encoding that varies according to geometric distribution. For example, as kvalue increases, the slope of the geometric distribution becomes gentler.

- XMvalue, YMvalue - The number of leading zeros.

- CodeNum - Code number.

- dx - Differential value between x-coordinate values of a current point and a previous point.

$dx = x[i] – x[i – 1]$

- dy - Differential value between y-coordinate values of the current point and the previous point.

$dy = y[i] – y[i – 1]$

- INFO - Value having information about dx or dy..

12.2.3.25. Decoding Process

When the LASeR binary stream is assumed to be decoded into a point sequence of (x_0, y_0), (x_1, y_1), …, (x_n, y_n),

- The number of points in the point sequence is extracted from the LASeR binary stream
- The flag is extracted from the LASeR binary stream
- If the value of the flag is zero which indicates the point sequence is encoded using the fixed length coding, decoding according to the following process
  - If the number of points is two or less:
    - Each value of x_0, y_0, x_1 and y_1 is extracted by reading “bits” number of bits
  - Otherwise:
    (i) Each value of x_0 and y_0 is extracted by reading “bits” number of bits
    (ii) The number “bitsx” of the bits required for a differential value dx of x-coordinates and the number “bitsy” of the bits required for a differential value dy of y-coordinates are extracted
    (iii) dx and dy are extracted by reading “bitsx” bits and “bitsy” bits, respectively, and then $x_i = x_{i-1}$ + dx and $y_i = y_{i-1}$ + dy are calculated
    (iv) i is incremented and the previous step (iii) is performed (n-1) times.
- If the value of the flag is one which indicates the point sequence is encoded using the Exp-Golomb coding, decoding according to the following process
  - The number of points of the point sequence is extracted from the LASeR binary stream
  - The parameter k is extracted from the LASeR binary stream
  - “bits” number of bits are read and then the first point coordinates (x0, y0) are decoded
  - For each point except the point (x_0, y_0), the following process is performed to decode one point (x_i, y_i).

(i) Bits are read one at a time until "1" is detected and the total number of read bits is set to M

(ii) The read "1" is discarded

(iii) (M+k) bits are read and assigned to INFO

(iv) CodeNum = 2^{M+k} + INFO – 2^{k} is calculated

(v) dx is calculated from CodeNum

(vi) $x_i = x_{i-1}$ + dx is calculated

(vii) Bits are read one by one until "1" is detected and the total number of read bits is set to M

(viii) The read "1" is discarded

(ix) (M+k) bits are read and assigned to INFO

(x) CodeNum = 2^{M+k} + INFO – 2^{k} is calculated

(xi) dy is calculated from CodeNum

(xii) $y_i = y_{i-1}$ + dy is calculated.

12.2.3.26. Encoding Process

This subclause is informative. When a point sequence is assumed to be comprised of (n+1) number of points: (x_0, y_0), (x_1, y_1), …, (x_n, y_n),

- Choose coding method between Exp-Golomb coding or fixed length coding
- When fixed length coding is chosen, encoding the point sequence according to the following process
  - If the number of points is two or less:
    (i) The minimum number of bits with which all of x_0, y_0, x_1, and y_1 can be encoded is calculated and encoded
    (ii) Points (x_0, y_0) and (x_1, y_1) are encoded using the number of bits calculated above
  - Otherwise:
    (i) The minimum number of bits with which the point (x_0, y_0) can be encoded is calculated and encoded
    (ii) Point (x_0, y_0) is encoded using the number of bits calculated above
    (iii) dx_{10}, …, dx_{nn-1} (here, $dx_{nn-1} = x_n - x_{n-1}$) are calculated and then the number, bitsx, of bits required for encoding them is calculated
    (iv) dy_{10}, …, dy_{nn-1} (here, $dy_{nn-1} = y_n - y_{n-1}$) are calculated and then the number, bitsy, of bits required for encoding them is calculated
    (v) The numbers of bits, bitsx and bitsy are encoded
    (vi) dx_{10}, dy_{10}, …, dx_{nn-1}, dy_{nn-1} are encoded
- When Exp-Golomb coding is chosen, encoding the point sequence according to the following process
  - The minimum number of bits with which the point (x_0, y_0) can be encoded is calculated and encoded
  - Point (x_0, y_0) is encoded using the minimum number of bits

- For each point except the point (x_0, y_0), the following process is performed to encode one point (x_i, y_i).

(i) A difference, "diffx," between x_i and x_{i-1} is mapped into an EG code number CodeNum without a sign, according to the rule given below:

If (diffx >= 0) CodeNum = diffx * 2 – 1

else CodeNum = |diffx| * 2

(ii) M denoting the number of leading zeros is calculated by $M = | \log_2(CodeNum + 2^k)| - k$

(iii) M number of "0" bits are recorded.

(iv) One "1" bit is recorded.

(v) The suffix offset "INFO," which carries information, is calculated by $INFO = CodeNum + 2^k - 2^{M+k}$

(vi) INFO is recorded in (M+k) bits

(vii) A difference "diffy" between yi and yi-1 is mapped into an EG code number CodeNum

(viii) Without a sign, according to the rule:

If (diffy >= 0) CodeNum = diffy * 2 – 1; and

else CodeNum = |diffy| * 2.

(ix) The number "M" of leading zeros is calculated by $M = | \log_2(CodeNum + 2^k)| - k$

(x) M number of "0" bits are recorded

(xi) One "1" bit is recorded.

(xii) INFO is calculated by $INFO = CodeNum + 2^k - 2^{M+k}$

(xiii) INFO is recorded in the (M+k) bits.

12.2.3.27. ValueWithUnits

12.2.3.28. Syntax

```
class attr_custom_valueWithUnits {
    uint(32) value; // float represented as fixed point with 8 bits mantissa
    uint(3) units; // 0 no unit or px, 1 'in', 2 'cm', 3 'mm', 4 'pt', 5 'pc', 6 '%'
}
```

12.2.3.29. Semantics

This class decodes a value with unit.

12.2.3.30. AnimatedValues

12.2.3.31. Syntax

```
class attr_custom_AnimatedValues {
    uint(4) type;
    vluimsbf5 nbValue;
    for (int i=0; i < nbValue; i++) {
      bit(1) escapeFlag[[i]];
      if (escapeFlag[[i]]) {
          // case for inherit and other mixed enum+number cases
          bit(2) escapeEnum[[i]];
      } else {
        switch(type) {
          case 0: // string
              attr_custom_byteAlignedString value[[i]];
              break;
          case 1: // float
              attr_custom_fixed_16_8 value[[i]];
              break;
          case 12:
              attr_custom_anyURI value[[i]];
              break;
          case 2: // path
              attr_custom_path value[[i]];
              break;
          case 3: // pointSeq
               attr_custom_pointSequence value[[i]];
              break;
          case 4: // fraction
              attr_custom_0to1float value[[i]];
              break;
          case 5: // color
              attr_custom_paint value[[i]];
              break;
          case 6: // enum
          case 10: // id
              vluimsbf5 value[[i]];
              break;
          case 11: // font
              vluimsbf5 j;
              value[i] = fontTable[j];
              break;
          case 7: // ints
              vluimsbf5 nbInts;
              for (int k = 0; k < nbInts; k++) {
                  vluimsbf5 value[[i]][[k]];
              }
              break;
          case 8: // floats
              vluimsbf5 nbFloats;
              for (int k = 0; k < nbFloats; k++) {
                  attr_custom_fixed_16_8 value[[i]][[k]];
              }
              break;
          case 9: // point
              attr_custom_coordinate valueX[[i]];
              attr_custom_coordinate valueY[[i]];
              break;
          default:
              attr_custom_extension privateData;
              break;
        }
    }
  }
}
```

12.2.3.32. Semantics

This class decodes a list of animated values.

12.2.3.33. AnimatedValue

12.2.3.34. Syntax

```
class attr_custom_AnimatedValue {
  uint(4) type;
  bit(1) escapeFlag;
  if (escapeFlag) {
      // case for inherit and other mixed enum+number cases
      bit(2) escapeEnum;
  } else {
       switch(type) {
        case 0:
            attr_custom_byteAlignedString value;
            break;
        case 1:
            attr_custom_fixed_16_8 value;
            break;
        case 12:
            attr_custom_anyURI value;
            break;
        case 2:
            attr_custom_path value;
            break;
        case 3:
            attr_custom_pointSequence value;
            break;
        case 4:
            attr_custom_0to1float value;
            break;
        case 5:
            attr_custom_paint value;
            break;
        case 6: // enum
        case 10: // id
            vluimsbf5 value;
            break;
        case 11: // font
            vluimsbf5 j;
            value = fontTable[j];
            break;
        case 7: // ints
            vluimsbf5 nbInts;
            for (int k = 0; k < nbInts; k++) {
                vluimsbf5 value[[k]];
            }
            break;
        case 8: // floats
            vluimsbf5 nbFloats;
            for (int k = 0; k < nbFloats; k++) {
                attr_custom_fixed_16_8 value[[k]];
            }
            break;
        case 9: // point
            attr_custom_coordinate valueX;
            attr_custom_coordinate valueY;
            break;
        default:
            attr_custom_extension privateData;
            break;
      }
  }
}
```

12.2.3.35. Semantics

This class decodes one animated value.

12.2.3.36. AlignedString

12.2.3.37. Syntax

```
class attr_custom_byteAlignedString {
    aligned vluimsbf8 slen;
    byte[slen] UTF-8string;
}
```

12.2.3.38. Semantics

This class decodes a string.

12.2.3.39. Fixed16_8

12.2.3.40. Syntax

```
class attr_custom_fixed_16_8 {
    int(24) float; // float represented as fixed point with 16 bits mantissa
}
```

12.2.3.41. Semantics

This class decodes a fixed point number.

12.2.3.42. Coordinate

12.2.3.43. Syntax

```
class attr_custom_coordinate {
  uint(coordBits) coord;
}
```

12.2.3.44. Semantics

The `coordBits` value shall be initialised using the initialisation parameters as defined in subclause 6.6.2.3.

12.2.3.45. UpdateValue

12.2.3.46. Syntax

```
const int TYPE_boolean = 0;
const int TYPE_enum = 1;
const int TYPE_color = 2;
const int TYPE_fraction = 3;
const int TYPE_float = 4;
const int TYPE_time = 5;
const int TYPE_point = 6;
const int TYPE_matrix = 7;
const int TYPE_string = 8;
const int TYPE_points = 9;
const int TYPE_path = 10;
const int TYPE_ints = 11;
const int TYPE_floats = 12;
const int TYPE_smil_times = 13;
const int TYPE_unit = 14;
const int TYPE_index = 15;
const int TYPE_URI = 16;
const int TYPE_ID = 17;
const int TYPE_event = 18;
const int TYPE_scale = 19;
const int TYPE_coordinate = 20;
const int TYPE_keyTimes = 21;

const int updateTypeOfAttribute[] = {
        TYPE_string, // a.target
        TYPE_boolean, // accumulate
        TYPE_boolean, // additive
        TYPE_fraction, // audio-level
        TYPE_float, // bandwidth
        TYPE_smil_times, // begin
        TYPE_enum, // calcMode
        -1, // children
        TYPE_index, // choice
        TYPE_time, // clipBegin
        TYPE_time, // clipEnd
        TYPE_color, // color
        TYPE_enum, // color-rendering
        TYPE_coordinate, // cx
        TYPE_coordinate, // cy
        TYPE_path, // d
        TYPE_floats, //delta
        TYPE_enum, // display
        TYPE_enum, // display-align
        TYPE_time, // dur
        TYPE_boolean, // editable
        TYPE_boolean, // enabled
        TYPE_smil_times, // end
        TYPE_event, // event
        TYPE_boolean, // externalResourcesRequired
        TYPE_color, // fill
        TYPE_fraction, // fill-opacity
        TYPE_enum, // fill-rule
        TYPE_enum, // focusable
        TYPE_index, // font-family
        TYPE_float, // font-size
        TYPE_enum, // font-style
        TYPE_enum, // font-variant
        TYPE_enum, // font-weight
        TYPE_boolean, // fullscreen
        TYPE_enum, // gradient-units
        TYPE_ID, // handler
        TYPE_coordinate, // height
        TYPE_enum, // image-rendering
        TYPE_floats, // keyPoints
        TYPE_floats, // keySplines
        TYPE_keyTimes, // keyTimes
        TYPE_float, // line-increment
        TYPE_ID, // listener.target
        TYPE_string, // mediaCharacterEncoding
        TYPE_string, // mediaContentEncodings
        TYPE_float, // mediaSize
        TYPE_time, // mediaTime
        TYPE_ID, // nav-down
        TYPE_ID, // nav-down-left
        TYPE_ID, // nav-down-right
        TYPE_ID, // nav-left
        TYPE_ID, // nav-next
        TYPE_ID, // nav-prev
        TYPE_ID, // nav-right
        TYPE_ID, // nav-up
        TYPE_ID, // nav-up-left
```

```
        TYPE_ID, // nav-up-right
        TYPE_ID, // observer
        TYPE_fraction, // offset
        TYPE_fraction, / opacity
        TYPE_enum, // overflow
        TYPE_enum, // overlay
        TYPE_path, // path
        TYPE_float, // pathLength
        TYPE_enum, // pointer-events
        TYPE_points, // points
        -1, // preserveAspectRatio
        TYPE_coordinate, // r
        TYPE_float, // repeatCount
        TYPE_time, // repeatDur
        TYPE_string, // requiredExtensions
        TYPE_ints, // requiredFeatures
        TYPE_string, // requiredFormats
        TYPE_enum, // restart
        TYPE_floats, // rotate
        TYPE_float, // rotation
        TYPE_coordinate, // rx
        TYPE_coordinate, // ry
        TYPE_scale, // scale
        TYPE_enum, // shape-rendering
        TYPE_point, // size
        TYPE_color, // solid-color
        TYPE_fraction, // solid-opacity
        TYPE_color, // stop-color
        TYPE_fraction, // stop-opacity
        TYPE_color, // stroke
        TYPE_floats, // stroke-dasharray
        TYPE_float, // stroke-dashoffset
        TYPE_enum, // stroke-linecap
        TYPE_enum, // stroke-linejoin
        TYPE_float, // stroke-miterlimit
        TYPE_fraction, // stroke-opacity
        TYPE_float, // stroke-width
        TYPE_unit, // svg.height
        TYPE_unit, // svg.width
        TYPE_enum, // syncBehavior
        TYPE_enum, // syncBehaviorDefault
        TYPE_URI, // syncReference
        TYPE_float, // syncTolerance
        TYPE_float, // syncToleranceDefault
        TYPE_string, // systemLanguage
        TYPE_enum, // text-align
        TYPE_enum, // text-anchor
        TYPE_enum, // text-decoration
        TYPE_enum, // text-rendering
        TYPE_string, // textContent
        TYPE_matrix, // transform
        TYPE_enum, // transformBehavior
        TYPE_point, // translation
        TYPE_enum, // vector-effect,
        TYPE_floats, // viewBox
        TYPE_color, // viewport-fill
        TYPE_fraction, // viewport-fill-opacity
        TYPE_enum, // visibility
        TYPE_coordinate, // width
        TYPE_coordinate, // x
        TYPE_coordinate, // x1
        TYPE_coordinate, // x2
        TYPE_enum, // xlink:actuate
        TYPE_string, // xlink:arcrole
        TYPE_URI, // xlink:href
        TYPE_string, // xlink:role
        TYPE_enum, // xlink:show
        TYPE_string, // xlink:title
        TYPE_enum, // xlink:type
        TYPE_string, // xml:base
        TYPE_string, // xml:lang
        TYPE_coordinate, // y
        TYPE_coordinate, // y1
        TYPE_coordinate, // y2
        TYPE_boolean // zoomAndPan
    };

const int ITYPE_point = 0;
const int ITYPE_int = 1;
const int ITYPE_float = 2;
const int ITYPE_keyTime = 3;
const int ITYPE_0to1 = 4;
const int ITYPE_smil_time = 5;

const int indexedTypeOfAttribute[] = {
        -1, // a.target
        -1, // accumulate
```

```
        -1, // additive
        -1, // audio-level
        -1, // bandwidth
        ITYPE_smil_time, // begin
        -1, // calcMode
        -1, // children
        -1, // choice
        -1, // clipBegin
        -1, // clipEnd
        -1, // color
        -1, // color-rendering
        -1, // cx
        -1, // cy
        -1, // d
        -1, // delta
        -1, // display
        -1, // display-align
        -1, // dur
        -1, // editable
        -1, // enabled
        ITYPE_smil_time, // end
        -1, // event
        -1, // externalResourcesRequired
        -1, // fill
        -1, // fill-opacity
        -1, // fill-rule
        -1, // focusable
        -1, // font-family
        -1, // font-size
        -1, // font-style
        -1, // font-variant
        -1, // font-weight
        -1, // fullscreen
        -1, // gradient-units
        -1, // handler
        -1, // height
        -1, // image-rendering
        ITYPE_0to1, // keyPoints
        ITYPE_float, // keySplines
        ITYPE_keyTime, // keyTimes
        -1, // line-increment
        -1, // listener.target
        -1, // mediaCharacterEncoding
        -1, // mediaContentEncodings
        -1, // mediaSize
        -1, // mediaTime
        -1, // nav-down
        -1, // nav-down-left
        -1, // nav-down-right
        -1, // nav-left
        -1, // nav-next
        -1, // nav-prev
        -1, // nav-right
        -1, // nav-up
        -1, // nav-up-left
        -1, // nav-up-right
        -1, // observer
        -1, // offset
        -1, // opacity
        -1, // overflow
        -1, // overlay
        -1, // path
        -1, // pathLength
        -1, // pointer-events
        ITYPE_point, // points
        -1, // preserveAspectRatio
        -1, // r
        -1, // repeatCount
        -1, // repeatDur
        -1, // requiredExtensions
        ITYPE_int, // requiredFeatures
        -1, // requiredFormats
        -1, // restart
        -1, // rotate
        -1, // rotation
        -1, // rx
        -1, // ry
        -1, // scale
        -1, // shape-rendering
        -1, // size
        -1, // solid-color
        -1, // solid-opacity
        -1, // stop-color
        -1, // stop-opacity
        -1, // stroke
        ITYPE_float, // stroke-dasharray
        -1, // stroke-dashoffset
```

```
        -1, // stroke-linecap
        -1, // stroke-linejoin
        -1, // stroke-miterlimit
        -1, // stroke-opacity
        -1, // stroke-width
        -1, // svg.height
        -1, // svg.width
        -1, // syncBehavior
        -1, // syncBehaviorDefault
        -1, // syncReference
        -1, // syncTolerance
        -1, // syncToleranceDefault
        -1, // systemLanguage
        -1, // text-align
        -1, // text-anchor
        -1, // text-decoration
        -1, // text-rendering
        -1, // textContent
        -1, // transform
        -1, // transformBehavior
        -1, // translation
        -1, // vector-effect
        ITYPE_float, // viewBox
        -1, // viewport-fill
        -1, // viewport-fill-opacity
        -1, // visibility
        -1, // width
        -1, // x
        -1, // x1
        -1, // x2
        -1, // xlink:actuate
        -1, // xlink:arcrole
        -1, // xlink:href
        -1, // xlink:role
        -1, // xlink:show
        -1, // xlink:title
        -1, // xlink:type
        -1, // xml:base
        -1, // xml:lang
        -1, // y
        -1, // y1
        -1, // y2
        -1 // zoomAndPan
    };


class attr_custom_updateValue {
    if (updateHasIndex) { // is the hasIndex bit set in the update ? false if not in an update
        // indexed
        switch(indexedTypeOfAttribute[attributeIndex]) {
            case ITYPE_point:
                attr_custom_coordinate pointValueX;
                attr_custom_coordinate pointValueY;
                break;
            case ITYPE_int:
                byte intValue;
                break;
            case ITYPE_float:
                attr_custom_fixed_16_8 floatValue;
                break;
            case ITYPE_keyTime:
                bit(1) hasShort;
                if (hasShort) {
                  bit(1) isZero;
                  if (isZero) {
                    timevalue = 0;
                  } else {
                    timevalue = 1;
                  }
                } else {
                  uint(12) timevalue; // 0 to 1 float encoded on 12 bits
                }
                break;
            case ITYPE_smil_time:
                custom_smil_time timevalue;
                break;
            case ITYPE_0to1:
                attr_custom_0to1float valueX;
                attr_custom_0to1float valueY;
                break;
            default:
                attr_custom_extension privateData;
                break;
        }
    } else {
      switch(updateTypeOfAttribute[attributeIndex]) {
        case TYPE_boolean:
```

```
            bit(1) booleanvalue;
            break;
        case TYPE_enum:
            bit(1) isDefaultValue; // default value (inherit, auto, default ...)
            if (!isDefaultValue) {
              vluimsbf5 enumValue;
            }
            break;
        case TYPE_index:
            bit(1) isDefaultValue; // default value (inherit, auto, default ...)
            if (!isDefaultValue) {
                bit(1) escapeFlag;
                if (escapeFlag) {
                  // case for mixed enum+number cases
                  bit(2) escapeEnumVal;
                } else {
                  vluimsbf5 indexValue;
                }
            }
            break;
        case TYPE_ID:
            bit(1) isDefaultValue; // default value (inherit, auto, default ...)
            if (!isDefaultValue) {
                bit(1) escapeFlag;
                if (escapeFlag) {
                  // case for mixed enum+ID cases
                  bit(2) escapeEnumVal;
                } else {
                    vluimsbf5 IDValue;
                }
            }
            break;
        case TYPE_color:
            attr_custom_paint colorValue;
            break;
        case TYPE_fraction:
            bit(1) isDefaultValue; // default value (inherit, auto, default ...)
            if (!isDefaultValue) {
                attr_custom_0to1float fractionValue;
            }
            break;
        case TYPE_float:
            bit(1) isDefaultValue; // default value (inherit, auto, default ...)
            if (!isDefaultValue) {
                bit(1) escapeFlag;
                if (escapeFlag) {
                  // case for mixed enum+float cases
                  bit(2) escapeEnumVal;
                } else {
                    attr_custom_fixed_16_8 floatValue;
                }
            }
            break;
        case TYPE_time:
            attr_time timeValue;
            break;
        case TYPE_point:
            attr_custom_coordinate pointValueX;
            attr_custom_coordinate pointValueY;
            break;
        case TYPE_matrix:
            custom_matrix matrixValue;
            break;
        case TYPE_string:
            attr_custom_byteAlignedString stringValue;
            break;
        case TYPE_points:
            attr_custom_pointSequence pointValues;
            break;
        case TYPE_path:
            attr_custom_path pathValue;
            break;
        case TYPE_ints:
            vluimsbf5 nb;
            for (int i = 0; i < nb; i++) {
               uint(8) intValue[[i]];
            }
            break;
        case TYPE_floats:
            bit(1) isDefaultValue; // only for stroke-dasharray inherit
            if (!isDefaultValue) {
                bit(1) escapeFlag;
                if (escapeFlag) { // only for stroke-dasharray none
                  bit(2) escapeEnumVal;
                } else {
                    vluimsbf5 nb1;
                    for (int i = 0; i < nb1; i++) {
```

```
                    attr_custom_fixed_16_8 floatValue[[i]];
                }
            }
        }
        break;
      case TYPE_keyTimes:
        attr_custom_fraction12List timeValues;
        break;
      case TYPE_smil_times:
        attr_smil_times timeValues1;
        break;
      case TYPE_unit:
        attr_custom_valueWithUnits unitValue;
        break;
      case TYPE_URI:
        attr_custom_anyURI uri;
        break;
      case TYPE_event:
        attr_custom_event event;
        break;
      case TYPE_scale:
        attr_custom_fixed_16_8 scaleX;
        attr_custom_fixed_16_8 scaleY;
        break;
      case TYPE_coordinate:
        attr_custom_coordinate value;
        break;
      default:
        attr_custom_extension privateData;
        break;
    }
  }
}
```

12.2.3.47. Semantics

This class decodes a constant value in an update.

12.2.3.48. Fraction12List

12.2.3.49. Syntax

```
class attr_custom_fraction12List {
    vluimsbf5 len;
    for (int i = 0; i < len; i++) {
      bit(1) hasShort;
      if (hasShort) {
        bit(1) isZero;
        if (isZero) {
            float[i] = 0;
        } else {
            float[i] = 1;
        }
      } else {
        uint(12) float[[i]]; // 0 to 1 float encoded on 12 bits
      }
    }
}
```

12.2.3.50. Semantics

This class encodes a list of 0 to 1 floats coded on 12 bits.

12.2.3.51. Event

12.2.3.52. Syntax

```
class attr_custom_event {
  bit(1) choice;
  switch(choice) {
    case 0:
      attr_custom_byteAlignedString string0;
      break;
    case 1:
      // Enumeration: abort{0} accessKey{1} activate{2} activatedEvent{3} beginEvent{4} click{5} deactivatedEvent{6}
endEvent{7} error{8} executionTime{9} focusin{10} focusout{11} keydown{12} keyup{13} load{14} longAccessKey{15}
mousedown{16} mousemove{17} mouseout{18} mouseover{19} mouseup{20} pause{21} pausedEvent{22} play{23}
repeatEvent{24} repeatKey{25} resize{26} resumedEvent{27} scroll{28} shortAccessKey{29} textinput{30} unload{31}
zoom{32}
      bit(6) event;
      switch (event) {
        case 1:  //accessKey
        case 15: //longAccessKey
        case 29: //shortAccessKey
        case 25: //repeatKey
          // Enumeration: *{0} 0{1} 1{2} 2{3} 3{4} 4{5} 5{6} 6{7} 7{8} 8{9} 9{10} ANY_KEY{11} DOWN{12} FIRE{13}
LEFT{14} NO_KEY{15} RIGHT{16} SHARP{17} SOFT_KEY_1{18} SOFT_KEY_2{19} UP{20}
          vluimsbf5 keyCode;
          break;
      }
      break;
  }
}
```

12.2.3.53. Semantics

This class encodes an event. In the case of accessKey and longAccessKey events, there is a keyCode in addition to the event name.

12.2.3.54. RareAttributes

12.2.3.55. Syntax

```
//indexes for rare attributes
const int RARE_CLASS = 0;
const int RARE_AUDIO_LEVEL = 1;
const int RARE_COLOR = 2;
const int RARE_COLOR_RENDERING = 3;
const int RARE_DISPLAY = 4;
const int RARE_DISPLAY_ALIGN = 5;
const int RARE_FILL_OPACITY = 6;
const int RARE_FILL_RULE = 7;
const int RARE_IMAGE_RENDERING = 8;
const int RARE_LINE_INCREMENT = 9;
const int RARE_POINTER_EVENTS = 10;
const int RARE_SHAPE_RENDERING = 11;
const int RARE_SOLID_COLOR = 12;
const int RARE_SOLID_OPACITY = 13;
const int RARE_STOP_COLOR = 14;
const int RARE_STOP_OPACITY = 15;
const int RARE_STROKE_DASHARRAY = 16;
const int RARE_STROKE_DASHOFFSET = 17;
const int RARE_STROKE_LINECAP = 18;
const int RARE_STROKE_LINEJOIN = 19;
const int RARE_STROKE_MITERLIMIT = 20;
const int RARE_STROKE_OPACITY = 21;
const int RARE_STROKE_WIDTH = 22;
const int RARE_TEXT_ANCHOR = 23;
const int RARE_TEXT_RENDERING = 24;
const int RARE_VIEWPORT_FILL = 25;
const int RARE_VIEWPORT_FILL_OPACITY = 26;
const int RARE_VECTOR_EFFECT = 27;
const int RARE_VISIBILITY = 28;
const int RARE_REQUIREDEXTENSIONS = 29;
const int RARE_REQUIREDFEATURES = 30;
const int RARE_REQUIREDFORMATS = 31;
const int RARE_SYSTEMLANGUAGE =32;
const int RARE_XML_BASE = 33;
const int RARE_XML_LANG = 34;
const int RARE_XML_SPACE = 35;
const int RARE_FOCUSNEXT = 36;
const int RARE_FOCUSNORTH = 37;
const int RARE_FOCUSNORTHEAST = 38;
const int RARE_FOCUSNORTHWEST = 39;
const int RARE_FOCUSPREV = 40;
const int RARE_FOCUSSOUTH = 41;
const int RARE_FOCUSSOUTHEAST = 42;
const int RARE_FOCUSSOUTHWEST = 43;
const int RARE_FOCUSWEST = 44;
const int RARE_FOCUSABLE = 45;
const int RARE_FOCUSEAST = 46;
const int RARE_TRANSFORM = 47;
const int RARE_TEXT_DECORATION = 48; //DCOR
const int RARE_EXTENSION = 49; //DCOR
const int RARE_FONT_VARIANT = 50;
const int RARE_FONT_FAMILY = 51;
const int RARE_FONT_SIZE = 52;
const int RARE_FONT_STYLE = 53;
const int RARE_FONT_WEIGHT = 54;
const int RARE_HREF_TITLE = 55;
const int RARE_HREF_TYPE = 56;
const int RARE_HREF_ROLE = 57;
const int RARE_HREF_ARCROLE = 58;
const int RARE_HREF_ACTUATE = 59;
const int RARE_HREF_SHOW = 60;
const int RARE_END = 61;
const int RARE_MAX = 62;
const int RARE_MIN = 63;
//rare AMD1
const int RARE_SYNCMASTER = 0;
const int RARE_FOCUSHIGHLIGHT = 1;
const int RARE_INITIALVISIBILITY = 2;
const int RARE_FULLSCREEN = 3;
const int RARE_REQUIREDFONTS = 4;

class attr_custom_rare {
  uint(6) nbOfAttributes;
  for(int i = 0; i < nbOfAttributes; i++) {
    uint(6) attributeRARE;
    switch(attributeRARE) {
      case RARE_CLASS:
        attr_custom_byteAlignedString class;
        break;
      case RARE_AUDIO_LEVEL:
```

```
        attr_custom_0to1float audio-level;
        break;
      case RARE_COLOR:
        attr_custom_paint color;
        break;
      case RARE_COLOR_RENDERING:
        // Enumeration: auto{0} inherit{1} optimizeQuality{2} optimizeSpeed{3}
        bit(2) color-rendering;
        break;
      case RARE_DISPLAY:
        // Enumeration: block{0} compact{1} inherit{2} inline{3} inline-table{4} list-item{5} marker{6} none{7} run-
in{8} table{9} table-caption{10} table-cell{11} table-column{12} table-column-group{13} table-footer-group{14}
table-header-group{15} table-row{16} table-row-group{17}
        bit(5) display;
        break;
      case RARE_DISPLAY_ALIGN:
        // Enumeration: after{0} auto{1} before{2} center{3} inherit{4}
        bit(3) displayAlign;
        break;
      case RARE_FILL_OPACITY:
        attr_custom_0to1float f;
        break;
      case RARE_FILL_RULE:
        // Enumeration: evenodd{0} inherit{1} nonzero{2}
        bit(2) fill-rule;
        break;
      case RARE_IMAGE_RENDERING:
        // Enumeration: auto{0} inherit{1} optimizeQuality{2} optimizeSpeed{3}
        bit(2) image-rendering;
        break;
      case RARE_LINE_INCREMENT:
        bit(1) choice;
        switch(choice) {
          case 0:
            attr_custom_fixed_16_8 line-increment;
            break;
          case 1:
            // Enumeration: auto{0} inherit{1}
            bit(1) line-increment;
            break;
          default:
            break;
        }
        break;
      case RARE_POINTER_EVENTS:
        //  Enumeration: all{0} boundingBox{1} fill{2} inherit{3} none{4} painted{5} stroke{6} visible{7}
visibleFill{8} visiblePainted{9} visibleStroke{10}
        bit(4) pointer-events;
        break;
      case RARE_SHAPE_RENDERING:
        // Enumeration: auto{0} crispEdges{1} geometricPrecision{2} inherit{3} optimizeSpeed{4}
        bit(3) shape-rendering;
        break;
      case RARE_SOLID_COLOR:
        attr_custom_paint color;
        break;
      case RARE_SOLID_OPACITY:
        attr_custom_0to1float opacity;
        break;
      case RARE_STOP_COLOR:
        attr_custom_paint color;
        break;
      case RARE_STOP_OPACITY:
        attr_custom_0to1float opacity;
        break;
      case RARE_STROKE_DASHARRAY:
        bit(1) isInherit;
        if (!isInherit) {
            vluimsbf5 max;
            for (int i = 0; i < max; i++) {
                attr_custom_fixed_16_8 dash[[i]];
            }
        }
        break;
      case RARE_STROKE_DASHOFFSET:
        custom_fixed_16_8i dashOffset;
        break;
      case RARE_STROKE_LINECAP:
        // Enumeration: butt{0} inherit{1} round{2} square{3}
        bit(2) stroke-linecap;
        break;
      case RARE_STROKE_LINEJOIN:
        // Enumeration: bevel{0} inherit{1} miter{2} round{3}
        bit(2) stroke-linejoin;
        break;
      case RARE_STROKE_MITERLIMIT:
        custom_fixed_16_8i miterlimit;
```

```
      break;
    case RARE_STROKE_OPACITY:
      attr_custom_0to1float opacity;
      break;
    case RARE_STROKE_WIDTH:
      custom_fixed_16_8i width;
      break;
    case RARE_TEXT_ANCHOR:
      // Enumeration: end{0} inherit{1} middle{2} start{3}
      bit(2) textAnchor;
    case RARE_TEXT_RENDERING:
      // Enumeration: auto{0} geometricPrecision{1} inherit{2} optimizeLegibility{3} optimizeSpeed{4}
      bit(3) text-rendering;
      break;
    case RARE_VIEWPORT_FILL:
      attr_custom_paint color;
      break;
    case RARE_VIEWPORT_FILL_OPACITY:
      attr_custom_0to1float opacity;
      break;
    case RARE_VECTOR_EFFECT:
      // Enumeration: default{0} inherit{1} non-scaling-stroke{2}
      bit(4) vector-effect;
      break;
    case RARE_VISIBILITY:
      // Enumeration: hidden{0} inherit{1} visible{2}
      bit(2) visibility;
      break;
    case RARE_REQUIREDEXTENSIONS:
      attr_custom_byteAlignedString string;
      break;
    case RARE_REQUIREDFEATURES:
      vluimsbf5 max;
      for (int t=0;t<max;t++) {
        // Enumeration:
       http://www.w3.org/Graphics/SVG/feature/1.2/#Animation{0}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Audio{1}
       http://www.w3.org/Graphics/SVG/feature/1.2/#ComposedVideo{2}
       http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessing{3}
       http://www.w3.org/Graphics/SVG/feature/1.2/#ConditionalProcessingAttribute{4}
       http://www.w3.org/Graphics/SVG/feature/1.2/#CoreAttribute{5}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Discard{6}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Extensibility{7}
       http://www.w3.org/Graphics/SVG/feature/1.2/#ExternalResourcesRequired{8}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Font{9}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Gradient{10}
       http://www.w3.org/Graphics/SVG/feature/1.2/#GraphicsAttribute{11}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Handler{12}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Hyperlinking{13}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Image{14}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Listener{15}
       http://www.w3.org/Graphics/SVG/feature/1.2/#MediaAttribute{16}
       http://www.w3.org/Graphics/SVG/feature/1.2/#NavigationAttribute{17}
       http://www.w3.org/Graphics/SVG/feature/1.2/#OpacityAttribute{18}
       http://www.w3.org/Graphics/SVG/feature/1.2/#PaintAttribute{19}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Prefetch{20}
       http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-all{21}
       http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-animated{22}
       http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static{23}
       http://www.w3.org/Graphics/SVG/feature/1.2/#SVG-static-DOM{24}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Scripting{25}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Shape{26}
       http://www.w3.org/Graphics/SVG/feature/1.2/#SolidColor{27}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Structure{28}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Text{29}
       http://www.w3.org/Graphics/SVG/feature/1.2/#TextFlow{30}
       http://www.w3.org/Graphics/SVG/feature/1.2/#TimedAnimation{31}
       http://www.w3.org/Graphics/SVG/feature/1.2/#TransformedVideo{32}
       http://www.w3.org/Graphics/SVG/feature/1.2/#Video{33}
       http://www.w3.org/Graphics/SVG/feature/1.2/#XlinkAttribute{34}
       http://www.w3.org/TR/SVG11/feature#Animation{35}
       http://www.w3.org/TR/SVG11/feature#AnimationEventsAttribute{36}
       http://www.w3.org/TR/SVG11/feature#BasicClip{37}
       http://www.w3.org/TR/SVG11/feature#BasicFilter{38}
       http://www.w3.org/TR/SVG11/feature#BasicFont{39}
       http://www.w3.org/TR/SVG11/feature#BasicGraphicsAttribute{40}
       http://www.w3.org/TR/SVG11/feature#BasicPaintAttribute{41}
       http://www.w3.org/TR/SVG11/feature#BasicStructure{42}
       http://www.w3.org/TR/SVG11/feature#BasicText{43}
       http://www.w3.org/TR/SVG11/feature#Clip{44}
       http://www.w3.org/TR/SVG11/feature#ColorProfile{45}
       http://www.w3.org/TR/SVG11/feature#ConditionalProcessing{46}
       http://www.w3.org/TR/SVG11/feature#ContainerAttribute{47}
       http://www.w3.org/TR/SVG11/feature#CoreAttribute{48}
       http://www.w3.org/TR/SVG11/feature#Cursor{49}
       http://www.w3.org/TR/SVG11/feature#DocumentEventsAttribute{50}
       http://www.w3.org/TR/SVG11/feature#Extensibility{51}
```

```
        http://www.w3.org/TR/SVG11/feature#ExternalResourcesRequired{52}
        http://www.w3.org/TR/SVG11/feature#Filter{53}
        http://www.w3.org/TR/SVG11/feature#Font{54}
        http://www.w3.org/TR/SVG11/feature#Gradient{55}
        http://www.w3.org/TR/SVG11/feature#GraphicalEventsAttribute{56}
        http://www.w3.org/TR/SVG11/feature#GraphicsAttribute{57}
        http://www.w3.org/TR/SVG11/feature#Hyperlinking{58}
        http://www.w3.org/TR/SVG11/feature#Image{59}
        http://www.w3.org/TR/SVG11/feature#Marker{60}
        http://www.w3.org/TR/SVG11/feature#Mask{61}
        http://www.w3.org/TR/SVG11/feature#OpacityAttribute{62}
        http://www.w3.org/TR/SVG11/feature#PaintAttribute{63}
        http://www.w3.org/TR/SVG11/feature#Pattern{64}
        http://www.w3.org/TR/SVG11/feature#SVG{65}
        http://www.w3.org/TR/SVG11/feature#SVG-animation{66}
        http://www.w3.org/TR/SVG11/feature#SVG-dynamic{67}
        http://www.w3.org/TR/SVG11/feature#SVG-static{68}
        http://www.w3.org/TR/SVG11/feature#SVGDOM{69}
        http://www.w3.org/TR/SVG11/feature#SVGDOM-animation{70}
        http://www.w3.org/TR/SVG11/feature#SVGDOM-dynamic{71}
        http://www.w3.org/TR/SVG11/feature#SVGDOM-static{72}
        http://www.w3.org/TR/SVG11/feature#Script{73}
        http://www.w3.org/TR/SVG11/feature#Shape{74}
        http://www.w3.org/TR/SVG11/feature#Structure{75}
        http://www.w3.org/TR/SVG11/feature#Style{76}
        http://www.w3.org/TR/SVG11/feature#Text{77}
        http://www.w3.org/TR/SVG11/feature#View{78}
        http://www.w3.org/TR/SVG11/feature#ViewportAttribute{79}
        http://www.w3.org/TR/SVG11/feature#XlinkAttribute{80}
        http://www.w3.org/TR/SVGMobile/Basic/feature#SVGBasicDomCore{81}
        http://www.w3.org/TR/SVGMobile/Basic/feature#SVGBasicDomExtended{82}
        http://www.w3.org/TR/SVGMobile/Basic/feature#all{83}
        http://www.w3.org/TR/SVGMobile/Basic/feature#base{84}
        http://www.w3.org/TR/SVGMobile/Basic/feature#css{85}
        http://www.w3.org/TR/SVGMobile/Basic/feature#interactivity{86}
        http://www.w3.org/TR/SVGMobile/Tiny/feature#all{87}
        http://www.w3.org/TR/SVGMobile/Tiny/feature#base{88}
        http://www.w3.org/TR/SVGMobile/Tiny/feature#interactivity{89}
        org.w3c.dom.svg{90} org.w3c.dom.svg.all{91}
        org.w3c.dom.svg.animation{92}
        org.w3c.dom.svg.dynamic{93}
        org.w3c.dom.svg.static{94}
        org.w3c.svg{95}
        org.w3c.svg.all{96}
        org.w3c.svg.animation{97}
        org.w3c.svg.dynamic{98}
        org.w3c.svg.static{99}
        bit(8) feature[[t]];
      }
      break;
    case RARE_REQUIREDFORMATS:
      attr_custom_byteAlignedString string;
      break;
    case RARE_SYSTEMLANGUAGE:
      attr_custom_byteAlignedString string;
      break;
    case RARE_XML_BASE:
      attr_custom_byteAlignedString string;
      break;
    case RARE_XML_LANG:
      attr_custom_byteAlignedString string;
      break;
    case RARE_XML_SPACE:
      // Enumeration: default{0} preserve{1}
      bit(1) space;
      break;
    case RARE_FOCUSNEXT:
      custom_focus focusNext;
      break;
    case RARE_FOCUSNORTH:
      custom_focus focusNorth;
      break;
    case RARE_FOCUSNORTHEAST:
      custom_focus focusEast;
      break;
    case RARE_FOCUSNORTHWEST:
      custom_focus focusWest;
      break;
    case RARE_FOCUSPREV:
      custom_focus focusPrev;
      break;
    case RARE_FOCUSSOUTH:
      custom_focus focusSouth;
      break;
    case RARE_FOCUSSOUTHEAST:
      custom_focus focusSouthEast;
      break;
```

```
      case RARE_FOCUSSOUTHWEST:
        custom_focus focusSouthWes;
        break;
      case RARE_FOCUSWEST:
        custom_focus focusWest;
        break;
      case RARE_FOCUSABLE:
        // Enumeration: auto{0} false{1} true{2}
        bit(2) focusable;
        break;
      case RARE_FOCUSEAST:
        custom_focus focusEast;
        break;
      case RARE_TRANSFORM:
        custom_matrix matrix;
        break;
      case RARE_FONT_VARIANT:
        // Enumeration: inherit{0} normal{1} small-caps{2}
        bit(2) font-variant;
        break;
      case RARE_FONT_FAMILY:
        bit(1) isInherit;
        if (!isInherit) {
            uint(fontIndexBits) fontIndex;
        }
        break;
      case RARE_FONT_SIZE:
        custom_fixed_16_8i size;
        break;
      case RARE_FONT_STYLE:
        // Enumeration: inherit{0} italic{1} normal{2} oblique{3}
        bit(3) font-style;
        break;
      case RARE_TEXT_DECORATION:
        //     Enumeration:     LASeR_EMBOSS{0}     LASeR_ENGRAVE{1}     LASeR_LEFTDROPSHADOW{2}     LASeR_OUTLINE{3}
LASeR_RIGHTDROPSHADOW{4} inherit{5} none{6} underline{7}
        bit(4) textDecoration;
        break;
      case RARE_FONT_WEIGHT:
        //  Enumeration:  100{0}  200{1}  300{2}  400{3}  500{4}  600{5}  700{6}  800{7}  900{8}  bold{9}  bolder{10}
inherit{11} lighter{12} normal{13}
        bit(4) font-weight;
        break;
      case RARE_HREF_TITLE:
        attr_custom_byteAlignedString title;
        break;
      case RARE_HREF_TYPE:
        // Enumeration: simple{0}
        bit(3) type;
        break;
      case RARE_HREF_ROLE:
        attr_custom_anyURI role;
        break;
      case RARE_HREF_ARCROLE:
        attr_custom_anyURI arcrole;
        break;
      case RARE_HREF_ACTUATE:
        // Enumeration: onLoad{0} onRequest{1}
        bit(2) actuate;
        break;
      case RARE_HREF_SHOW:
        // Enumeration: embed{0} new{1} other{2} replace{3}
        bit(3) show;
        break;
      case RARE_END:
        attr_smil_times time;
        break;
      case RARE_MAX:
        attr_time time;
        break;
      case RARE_MIN:
        attr_time time;
        break;
      case RARE_EXTENSION:
          do {
              uint(extensionIDBits) extensionID;
              vluimsbf5 len; //length in bits
              if (extensionID == 2) { //LASeR AMD1
                   rare2 r;
              } else {
                   bit[len] toSkip;
              }
              bit(1) hasNextExtension;
              } while (hasNextExtension);
        break;
    }
  }
```

```
}
class rare2 {
  uint(2) nbOfAttributes;
  for(int i = 0; i < nbOfAttributes; i++) {
    uint(3) attributeRARE;
    switch(attributeRARE) {
      case RARE_SYNCMASTER:
        bit(1) syncMaster;
        break;
      case RARE_FOCUSHIGHLIGHT:
        // Enumeration: auto{0} none {1}
        bit(2) focusHighlight;
        break;
      case RARE_INITIALVISIBILITY:
        // Enumeration: whenStarted{0} always {1}
        bit(2) initialVisibility;
        break;
      case RARE_FULLSCREEN:
        // Enumeration: false(0) true(1)
        bit(2) fullscreen;
        break;
      case RARE_REQUIREDFONTS:
        attr_custom_byteAlignedString requiredFonts;
        break;
    }
  }
}
class custom_focus {
    bit(1) isEnum;
    if (isEnum) {
        // Enumeration: auto{0} self{1};
        bit(1) enum;
    } else {
        attr_custom_IDREF id;
    }
}
class custom_fixed_16_8i {
  bit(1) isInherit;
  if (!isInherit) {
    int(24) float;
  }
}
```

12.2.3.56. Time

12.2.3.57. Syntax

```
class custom_smil_time {
   bit(1) hasEvent;
   if (hasEvent) {
       bit(1) hasIdentifier;
       if (hasIdentifier) {
           attr_custom_IDREF idref;
       }
       attr_custom_event event;
   }
   bit(1) hasClock;
   if (hasClock) {
       bit(1) sign; //1 is negative
       vluimsbf5 clock;
   }
}
```

12.2.3.58. Semantics

The class `custom_smil_time` decodes a single time value in the begin and end attribute.

12.2.3.59. Private Data

12.2.3.60. Syntax

```
//*************************************************************************************************
// private data and anyXML
//*************************************************************************************************

class privateAnyXMLElement {
  vluimsbf5 len; //length in bytes
  aligned privateChildren pc;
}

class privateOpaqueElement {
  vluimsbf5 len; //length in bytes
  aligned uint(privateDataIdentifierIndexBits) privateDataIdentifierIndex;
  aligned byte[len - ((privateDataIdentifierIndexBits+7)>>3)] data;
}

class privateAnyXMLAttribute  {
  vluimsbf5 skipLen; //length in bytes
  aligned vluimsbf5 count;
  for (int i = 0; i < count; i++) {
    privateAttribute(0) attr[i];
    // private attribute stored in the first (0) bin, reserved for attr of LASeR elements
  }
}

class privateOpaqueAttribute {
  vluimsbf5 len; //length in bytes
  aligned uint(privateDataIdentifierIndexBits) privateDataIdentifierIndex;
  aligned byte[len - ((privateDataIdentifierIndexBits+7)>>3)] data;
}

class privateAttribute(index) {
  uint(log2sup(nbAttrNames[index])) attrIndex;
  attr_custom_byteAlignedString value;
}


class privateChildren {
    vluimsbf5 count;
    for (int i = 0; i < count; i++) {
        bit(1) isPrivate;
        if (isPrivate) {
            bit(1) isText;
            if (isText) {
                attr_custom_byteAlignedString textContent;
            } else {
                privateElement e;
            }
        } else {
            updatable_elements child[i];
        }
    }
}

class privateElement {
  uint(tagIndexBits) tagIndex;
  bit(1) hasID;
  if (hasID) {
    attr_custom_ID ID;
  }
  vluimsbf5 alen;
  for (int i = 0; i < alen; i++ ) {
    bit(1) hasNS;
    if (hasNS) {
      privateAttribute(0) att[i];
    } else {
      privateAttribute(tagIndex) att[i];
    }
  }
  bit(1) hasChildren;
  if (hasChildren) {
    privateChildren ch;
  }
}
```

12.2.3.61. Semantics

These classes hold private data.

12.2.3.62. Element-Attribute pair

12.2.3.63. Syntax

```
class attr_custom_ElementAttributeList {
  vluimsbf5 nbIds;
  for (int i = 0; i < nbIds; i++) {
      attr_custom_IDREF ref[[i]];
      attr_AttributeName attr[[i]];
    }
}
```

12.2.3.64. Semantics

The class `attr_custom_ElementAttributeList` holds a list of pairs of element ID and attribute ID.

12.2.3.65. ByteAlignment

12.2.3.66. Syntax

```
class attr_custom_align {
  aligned bit(0) dummy;
}
```

12.2.3.67. Semantics

This class executes byte-alignment.

12.2.3.68. Extension

12.2.3.69. Syntax

```
class attr_custom_extension {
    vluimsbf5 len;
    byte[len] privateData;
}
```

12.2.3.70. Semantics

This class holds skippable data of future extensions.

13. Usage of ISO/IEC 23001-1

13.1. Introduction

Clause 13 is normative for decoders compliant with both this specification and ISO/IEC 23001-1, otherwise it is informative.

In this clause, we describe how a compliant ISO/IEC 23001-1 decoder can be used to decode a LASeR bitstream. In order to decode a LASeR bitstream compliant with the binary syntax specified in clause 12, an ISO/IEC 23001-1 decoder should implement a set of specific type codecs and use the attached encoding schemas.

Compatibility with LASeR v1 binary syntax

The configuration of an ISO/IEC 15938-1 decoder able to decode LASeR bitstreams, as specified in 14496-20 version 1, will be updated synchronously with updates/extensions of LASeR schemas and data type codecs. The resulting configuration, specified by a default v2 decoderInit should be made compatible with the default configuration specified in 14496-20 version 1, so that a LASeR v1 bitstream is still decodable.

13.2. Electronic Attachments

- intermediateXML/svg1.xsd and intermediateXML/svg2.xsd: SVG encoding schema for use when using an ISO/IEC 23001-1 encoder to encode a LASeR stream.
- intermediateXML/laser1.xsd and intermediateXML/laser2.xsd: LASeR encoding schema for use when using an ISO/IEC 23001-1 encoder to encode a LASeR stream.
- intermediateXML/laser-datatypes1.xsd and intermediateXML/laser-datatypes2.xsd: data types encoding schema for use when using an ISO/IEC 23001-1 encoder to encode a LASeR stream.
- intermediateXML/ev.xsd: XML events encoding schema for use when using an ISO/IEC 23001-1 encoder to encode a LASeR stream.

13.3. Type Codecs

13.3.1. Classification Scheme

In the MPEG-7 framework, the use of a specific codec for a specific type is signalled using the codec configuration mechanism defined in ISO/IEC 23001-1. This mechanism associates a codec using its URI with a list of schema types. For that purpose, a URI is assigned to each codec in a `classificationScheme`, which defines the list of the specific codecs.

```
<ClassificationScheme uri="urn:mpeg:mpeg4:LASeR:2005:CodecTypeCS">
     <Term termID="1">
          <Name xml:lang="en">ID</Name>
          <Definition xml:lang="en">Encodes an ID a create an Id dictionary</Definition>
     </Term>
     <Term termID="2">
          <Name xml:lang="en">IDRef</Name>
          <Definition xml:lang="en">Encodes a reference to an id</Definition>
     </Term>
     <Term termID="3">
          <Name xml:lang="en">AnyURI</Name>
          <Definition xml:lang="en">Encodes a URI</Definition>
     </Term>
     <Term termID="4">
          <Name xml:lang="en">Paint</Name>
          <Definition xml:lang="en">Encodes a reference in a defined color table</Definition>
     </Term>
     <Term termID="5">
          <Name xml:lang="en">IDRefs</Name>
          <Definition xml:lang="en">Encodes a list of id references</Definition>
     </Term>
     <Term termID="6">
          <Name xml:lang="en">ByteAlign</Name>
          <Definition xml:lang="en">Encodes byte alignment</Definition>
     </Term>
     <Term termID="7">
          <Name xml:lang="en">Fraction</Name>
          <Definition xml:lang="en">Encodes an opacity value with a uniformQuantizer vmin = 0 vmax = 1 and nbits =
8</Definition>
     </Term>
     <Term termID="8">
          <Name xml:lang="en">Path</Name>
          <Definition xml:lang="en">Encodes a path</Definition>
     </Term>
     <Term termID="9">
          <Name xml:lang="en">PointSequence</Name>
          <Definition xml:lang="en">Encodes a list of coordinates</Definition>
     </Term>
     <Term termID="10">
          <Name xml:lang="en">ValueWithUnits</Name>
          <Definition xml:lang="en">Encodes a value with unit</Definition>
     </Term>
     <Term termID="11">
```

```
          <Name xml:lang="en">AnimatedValues</Name>
          <Definition xml:lang="en">Encodes a list of animated values</Definition>
     </Term>
     <Term termID="12">
          <Name xml:lang="en">AnimatedValue</Name>
          <Definition xml:lang="en">Encodes an animated value</Definition>
     </Term>
     <Term termID="13">
          <Name xml:lang="en">AlignedString</Name>
          <Definition xml:lang="en">Encodes a string in UTF_8 on byte boundary</Definition>
     </Term>
     <Term termID="14">
          <Name xml:lang="en">Fixed16_8</Name>
          <Definition xml:lang="en">Encodes a float using a fixed point encoding</Definition>
     </Term>
     <Term termID="15">
          <Name xml:lang="en">Coordinate</Name>
          <Definition xml:lang="en">Encodes a coordinate value</Definition>
     </Term>
     <Term termID="16">
          <Name xml:lang="en">UpdateValue</Name>
          <Definition xml:lang="en">Encodes an update value</Definition>
     </Term>
     <Term termID="17">
          <Name xml:lang="en">Fraction12List</Name>
          <Definition xml:lang="en">Encodes a list of 0 to 1 floats coded on 12 bits</Definition>
     </Term>
     <Term termID="18">
          <Name xml:lang="en">Event</Name>
          <Definition xml:lang="en">Encodes an enumeration of events</Definition>
     </Term>
     <Term termID="19">
          <Name xml:lang="en">RareAttributes</Name>
          <Definition xml:lang="en">Encodes a rare attributes list</Definition>
     </Term>
     <Term termID="20">
          <Name xml:lang="en">PrivateOpaqueAttribute</Name>
          <Definition xml:lang="en">Encodes a set of opaque private attributes</Definition>
     </Term>
     <Term termID="21">
          <Name xml:lang="en">PrivateOpaqueElement</Name>
          <Definition xml:lang="en">Encodes a opaque private element</Definition>
     </Term>
     <Term termID="22">
          <Name xml:lang="en">Time</Name>
          <Definition xml:lang="en">Encodes a time attribute</Definition>
     </Term>
     <Term termID="23">
          <Name xml:lang="en">Extension</Name>
          <Definition xml:lang="en">Encodes an extension</Definition>
     </Term>
     <Term termID="24">
          <Name xml:lang="en">PrivateAnyXMLAttribute</Name>
          <Definition xml:lang="en">Encodes a set of anyXML private attributes</Definition>
     </Term>
     <Term termID="25">
          <Name xml:lang="en">PrivateAnyXMLElement</Name>
          <Definition xml:lang="en">Encodes a anyXML private element</Definition>
     </Term>
</ClassificationScheme>
```

13.4. Type codecs for use with ISO/IEC 23001-1 decoders

13.4.1. ID Codec

This codec is mapped to the type `xsd:ID`. Syntax is defined by the class attr_custom_ID given in subclause 12.2.3.1.

13.4.2. IDRef Codec

This codec is mapped to the type `xsd:IDRef`. Syntax is defined by the class attr_custom_IDREF given in subclause 12.2.3.4.

13.4.3. AnyURI Codec

This codec is mapped to the type `xsd:anyURI`. Syntax is defined by the class attr_custom_anyURI given in subclause 12.2.3.7.

13.4.4. Paint Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:paintType`. Syntax is defined by the class attr_custom_paint given in subclause 12.2.3.10.

13.4.5. IDRefs Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:ElementAttributeListType`. Syntax is defined by the class attr_custom_ElementAttributeList given in subclause **Erreur ! Source du renvoi introuvable.**.

13.4.6. 13.4.6 ByteAlign Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:alignType`. Syntax is defined by the class attr_custom_align given in subclause **Erreur ! Source du renvoi introuvable.**.

13.4.7. Fraction Codec

This codec is mapped to type `urn:mpeg:mpeg4:LASeR:types:2005:opacityType`. Syntax is defined in [ISO/IEC 23001-1] subclause 8.5.2 and is expressed by class attr_custom_0to1float in subclause12.2.3.16.

13.4.8. Path Codec

This codec is mapped to the type `urn:mpeg:mpeg4:types:2005:pathType`. Syntax is defined by the class attr_custom_path given in clause subclause12.2.3.19.

13.4.9. PointSequence Codec

This codec is mapped to the type `urn:mpeg:mpeg4:types:2005:pointListType`. Syntax is defined by the class attr_custom_pointSequence given in subclause12.2.3.22.

13.4.10. ValueWithUnits Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:valueWithUnitsType`. Syntax is defined by the class attr_custom_valueWithUnits given in clause subclause 12.2.3.27.

13.4.11. AnimatedValues Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:AnimatedValuesType`. Syntax is defined by the class attr_custom_animatedValues given in subclause 12.2.3.30.

13.4.12. AnimatedValue Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:AnimatedValueType`. Syntax is defined by the class attr_custom_animatedValue given in subclause12.2.3.33.

13.4.13. AlignedString Codec

This codec is mapped to the `xsd:string`. Syntax is defined by the class attr_custom_byteAlignedString given in subclause 12.2.3.36.

13.4.14. Fixed16_8 Codec

This codec is mapped to the types `urn:mpeg:mpeg4:LASeR:types:2005:fixed16_8Type` and `urn:mpeg:mpeg4:LASeR:types:2005:offsetType`. Syntax is defined by the class attr_custom_fixed_16_8 given in subclause 12.2.3.39.

13.4.15. Coordinate Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:coordinateType`. Syntax is defined by the class attr_custom_coordinate given in subclause 12.2.3.42.

13.4.16. UpdateValue Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:updateValueType`. Syntax is defined by the class attr_custom_updateValue given in subclause 12.2.3.45.

13.4.17. Fraction12List Codec

This codec is mapped to the types `urn:mpeg:mpeg4:LASeR:types:2005:keySplinesType` and `:timeListType`. Syntax is defined by the class attr_custom_fraction12List given in subclause 12.2.3.4812.2.

13.4.18. Event Codec

This codec is mapped to the `urn:mpeg:mpeg4:LASeR:types:2005:eventType` type. Syntax is defined by the class attr_custom_event given in subclause 12.2.3.51.

13.4.19. RareAttributes Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:rareType` type of the data type encoding schema. Syntax is defined by the class attr_custom_rare given in subclause 12.2.3.54.

13.4.20. PrivateOpaqueAttribute Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:2005:opaqueAttributeType` type of the LASeR encoding schema. Syntax is defined by the class privateOpaqueAttribute given in subclause 0.

13.4.21. PrivateOpaqueElement Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:2005:opaqueElementType` type of the LASeR encoding schema.

Syntax is defined by the class privateOpaqueElement given in subclause 0.

13.4.22. Time Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:types:2005:smilTimeType` type of the LASeR encoding schema. Syntax is defined by the class custom_smil_time in subclause 12.2.3.56.

13.4.23. Extension Codec

This codec is mapped to the types `urn:mpeg:mpeg4:LASeR:types:2005:attrExtensionType` and `urn:mpeg:mpeg4:LASeR:2005:codecExtensionType`. Syntax is defined by the class `custom_extension` given in subclause 12.2.3.65.

13.4.24. PrivateAnyXMLAttribute Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:2005:anyXMLAttributeType` type of the LASeR encoding schema. Syntax is defined by the class privateAnyXMLAttribute given in subclause 0.

13.4.25. PrivateAnyXMLElement Codec

This codec is mapped to the type `urn:mpeg:mpeg4:LASeR:2005:anyXMLElementType` type of the LASeR encoding schema. Syntax is defined by the class privateAnyXMLElement given in subclause 0.

13.4.26. Decoding of private data

Private XML data encoded with ISO/IEC 23001-1 will be enclosed in the ISO reserved byte array in classes privateElementContainer and privateAttributeContainer specified in 14496-20 version 1.

```
…
class privateElementContainer{
  bit(2) privateDataType;
  vluimsbf5 len;
  if (privateDataType == 0) { // private data of type "anyXML"
    aligned privateChildren pc;
  } else if (privateDataType == 1) {
    aligned uint(nameSpaceIndexBits) nameSpaceIndex;
    aligned byte[len - ((nameSpaceIndexBits+7)>>3)] data;
  } else {
    aligned byte[len] reserved;// ISO reserved
 }
}…
```

13.5. DecoderInit

The default decoderInit for decoding LASeR binary content should have the the following constants:

Field	No. of bits	Value
SystemsProfileLevelIndication	8	'0x00'
UnitSizeCode	3	000
NoAdvancedFeatures	1	0
ReservedBits	5	11111
AdvancedFeaturesFlags_Length	8	0x08
InsertFlag	1	0
AdvancedOptimizedDecodersFlag	1	1
ReservedBitsZero	6	000000

The referenced schemas are the encoding schemas attached as separate files; the type-codec mapping is the one given earlier in 13.4.

The LASeRHeader can embed a BiM decoderInit in the extensionDescriptor field of the LASeRHeader, which allows signaling the ISO/IEC 23001-1 encoding of private data from additional namespaces.

```
…
bit(4) extensionIDBits;
bit(1) hasDecoderInit;
if (hasDecoderInit) {
    vluimsbf5 len;
    byte[len] decoderInit;
}
…
```

The referenced schemaURIs are (in this order) “urn:mpeg:mpeg4:LASeR:2005” and “urn:mpeg:mpeg4:LASeR:2006”, and correspond to intermediateXML/laser1.xsd and intermediateXML/laser2.xsd schemas; the type-codec mapping is the one given earlier in 13.4.

13.6. Decoding Process

This subclause specifies how the XML document obtained after decoding encoded binary data can be transformed for conformance to the LASeR validation schema specified in clause 11.

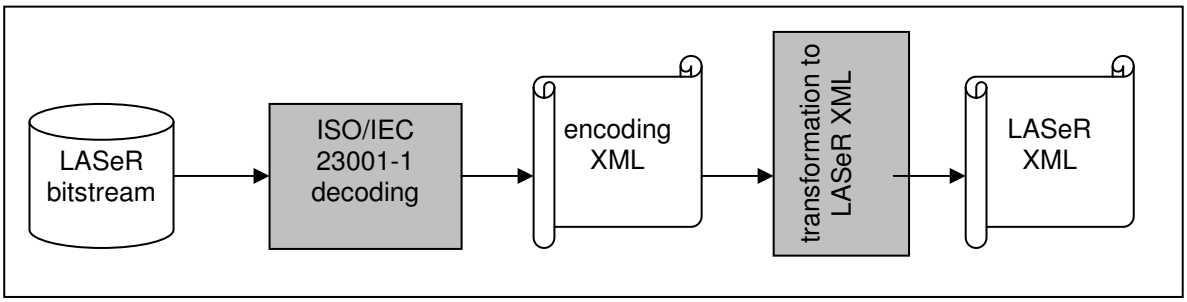


Figure 13 - ISO/IEC 23001-1 decoding process

Figure 1 shows the decoding process of a LASeR binary stream with a ISO/IEC 23001-1 decoder. The XML document obtained is called ***encoding XML***. It shall be conformant against the encoding schema (attached to the specification as a separate document). However, the encoding XML can be transformed into a LASeR XML document, i.e. conformant to the validation schema, according to the following transformations:

Ordinal Attributes Decoding

The ordinal attributes shall be transformed by removing successively from the beginning of their names: one Unicode U+005F character (LOW LINE), one Unicode digit from the Basic Latin block (U+0030 through U+0039), and another Unicode U+005F character (LOW LINE).

For example, attribute name “_1_stroke” should be transformed to “stroke”.

Extension Element textContent Decoding

The 'textContent' extension element shall be transformed by removing it from the document tree, and replacing it with its child nodes.

Rare Attributes Decoding

The attributes contained inside the 'rare' attribute shall be transformed as follows:

1. The value of the attribute is split into a list of strings separated by the Unicode U+0009 character (CHARACTER TABULATION).

2. Each item in the list obtained in step one shall be processed in turn using the following steps.

3. The name of the attribute shall be the part of the item up to but excluding the first Unicode U+003D character (EQUALS SIGN). The value of the attribute shall be the part of the item immediately following the first Unicode U+003D character (EQUALS SIGN) up to the end of the item.

4. If the attribute name begins with the string "xlink:", its namespace URI shall be defined to be the XLink namespace [W3C Xlink]. If the attribute name begins with the string "xml:", its namespace URI shall be defined to be the XML namespace. If the attribute name begins with the string "lsr:", its namespace URI shall be defined to be the LASeR namespace.Otherwise the name part shall be its local name.

5. On the owner element of the 'rare' attribute being transformed, an attribute shall be created using the name information derived from step (4). That attribute shall be given the value of the item as its value.

Once this processing has taken place, the 'rare' attribute shall be removed.

For example, attribute _0_rare="font-family=MONOSPACE	font-size=18	font-style=ITALIC	" should be transformed into 3 attributes: font-family="MONOSPACE" font-size=18 font-style="ITALIC".

Same-element Decoding

The sameX constructs allow the representation of an element A of type X based on a previous element B of same type. Instead of encoding the A element, only the differences between A and B are encoded in a sameX construct. The following constructs are possible:

samegType, *samelineType*, *samepathType*, *samepathfillType*, *samepolygonType*, *samepolygonfillType*, *samepolygonstrokeType, samepolygonType*, *samepolygonfillType*, *samepolygonstrokeType*, *samerectType*, *samerectfillType*, *sametextType, sametextfillType*, *sameuseType*.

In order to decode a sameX construct, a decoder shall maintain reference elements for each of the following element types: *polyline*, *polygon*, *path*, *g*, *use*, *text*, *rect* and *line.*

Initially, these reference elements are undefined. They are also undefined before and after decoding a LASeR Update.

When decoding an element of type X, this element becomes the reference element for the corresponding type as soon as all attributes are decoded. This allows a child element to use its parent as a reference element for the sameX construct.

When decoding a sameX construct, an element of type X is created and all attributes, not codable within this construct, are copied from the reference element of same type. Detailed description of codable attributes is given in subclause 12.2. Note that this newly created element does not replace the reference element.

Presence of a sameX construct, when the associated reference element is undefined, is forbidden.

Considerations for sameX constructs and script usage: In the following sequence of elements "*X*, script, and *X",* the second X element cannot be encoded using the sameX construct if the script element uses LASeR Updates, while it can if the script uses ECMAScript.

Elements that are decoded as sameX elements shall be transformed in turn as follows:

1. The string "same" shall be removed from the beginning of their name, and the string "stroke" or "fill", if present, shall be removed from the end of their name.

2. Given the name that remains after step (1), the first previous element in reverse document order with the same name is selected.

3. Each attribute owned by the selected element shall be processed in turn using the following steps.

a. If the attribute name is in the list of XX corresponding to the selected element, then processing shall skip to the next attribute of the selected element.

b. If the attribute is ‘fill’ and the original target element name ended with string “fill”, then processing shall skip to the next attribute of the selected element.

c. If the attribute is ‘stroke’ and the original target element name ended with string “stroke”, then processing shall skip to the next attribute of the selected element.

d. Otherwise, the attribute shall be copied over to the target same element together with its value.

Table 12 — attributes to skip for each "sameX" element

element name	attribute names
polyline	id, points
polygon	id, points
path	id, d
g	id
use	id, href
text	id, x, y
rect	id, x, y, width, height
line	id, x1, y1, x2, y2

For example,

```
<polyline fill="rgb(255,255,255)" stroke="rgb(0,0,0)" points="-96.5 …”/>
<samepolyline points="-76.95 …”/>
<samepolylinefill points="-74.5 447.5 -68.5 443.5"/>
```

should be transformed to:

```
<polyline fill="rgb(255,255,255)" stroke="rgb(0,0,0)" points="-96.5 …"/>
<polyline fill="rgb(255,255,255)" stroke="rgb(0,0,0)" points="-76.95 …"/>
<polyline stroke="rgb(0,0,0)" points="-74.5 447.5 -68.5 443.5"/>
```

Decoding of Encoding-related Elements

Each element prefixed by 'encoding-' shall be transformed by removing it from the document tree together with its content.

LASeR Amd1 extensions decoding

The extension elements LASeRAmd1, LASeRAmd1Extension, LASeRAmd1Command and SVGAmd1Extension shall be transformed by removing them from the document tree, and replacing them with their child nodes.

The extension elements from namespaces urn:mpeg:mpeg4:LASeR:2006, http://www.w3.org/2006/svg, and urn:mpeg:mpeg4:LASeR:types:2006, should be replaced by elements belonging to namespaces urn:mpeg:mpeg4:LASeR:2005, http://www.w3.org/2000/svg, and urn:mpeg:mpeg4:LASeR:types:2005, respectively.

Annex A
(informative)

Patent statements

The International Organization for Standardization and the International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this part of ISO/IEC 21000 may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from the companies listed below.

Attention is drawn to the possibility that some of the elements of this part of ISO/IEC 21000 may be the subject of patent rights other than those identified in this annex. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

The list below summarizes the formal patent statements received.

- France Telecom

Annex B
Media Type Registrations

B.1 Introduction

This appendix registers new MIME media types, "application/laser+xml", "application/laser", "application/laser+saf" and "application/saf" in conformance with [BCP 13].

B.2 Registration of Media Type application/laser+xml

This media type is to be used for scene descriptions in LASeRML.

MIME media type name:

application

MIME subtype name:

laser+xml

Required parameters:

None.

Optional parameters:

None

Encoding considerations:

Same as for application/xml. See [RFC3023] , section 3.2.

Restrictions on usage:

None

Security considerations:

As with other XML types and as noted in [RFC3023] section 10, repeated expansion of maliciously constructed XML entities can be used to consume large amounts of memory, which may cause XML processors in constrained environments to fail.

Several LASeR elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, LASeR documents may reference external media such as images, audio, video, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply.

In addition, because of the extensibility features for LASeR and of XML in general, it is possible that "application/laser+xml" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the LASeR namespaces and shall be ignored. Only in the case where the processor

recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the LASeR namespaces and in other namespaces.

Because LASeR is extensible, conformant "application/laser+xml" processors must expect that content received is well-formed XML, but it cannot be guaranteed that the content is valid to a particular DTD or Schema or that the processor will recognize all of the elements and attributes in the document.

LASeR has a published Test Suite as part of ISO/IEC 14496-4:2001/AMD17.

Published specification:

This media type registration is extracted from Appendix B of ISO/IEC 14496-20.

Additional information:
Person & email address to contact for further information:

Leonardo Chiariglione, Olivier Avaro, Philippe de Cuetos, Jean-Claude Dufourd (member-laser-media-types@googlegroups.com)

Intended usage:

COMMON

Author/Change controller:

The LASeR specification is a work product of the ISO/IEC/JTC1/SC29/WG11 (MPEG). ISO/MPEG has change control over these specifications.

B.3 Registration of Media Type application/laser+saf

This media type is to be used for scene descriptions in LASeR binary format packaged in SAF.

MIME media type name:

application

MIME subtype name:

laser+saf

Required parameters:

None.

Optional parameters:

None

Encoding considerations:

None

Restrictions on usage:

None

Security considerations:

Several LASeR elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, LASeR documents may reference external media such as images, audio, video, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply.

In addition, because of the extensibility features for LASeR in general, it is possible that "application/laser+saf" may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the LASeR namespaces and shall be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the LASeR namespaces and in other namespaces.

Because LASeR is extensible, conformant "application/laser+saf" processors must expect that content received is well-formed LASeR and SAF, but it cannot be guaranteed that the processor will recognize all of the elements and attributes in the document.

LASeR has a published Test Suite as part of ISO/IEC 14496-4:2001/AMD17.

Published specification:

This media type registration is extracted from Appendix B of ISO/IEC 14496-20.

Additional information:
Person & email address to contact for further information:

Leonardo Chiariglione, Olivier Avaro, Philippe de Cuetos, Jean-Claude Dufourd
(member-laser-media-types@googlegroups.com)

Intended usage:

COMMON

Author/Change controller:

The LASeR specification is a work product of the ISO/IEC/JTC1/SC29/WG11 (MPEG). ISO/MPEG has change control over these specifications.

B.4 Registration of Media Type application/laser

This media type is to be used for scene descriptions in LASeR binary format, e.g. for transport with RTP.

MIME media type name:

application

MIME subtype name:

laser

Required parameters:

None.

Optional parameters:

None

Encoding considerations:

None

Restrictions on usage:

None

Security considerations:

Several LASeR elements may cause arbitrary URIs to be referenced. In this case, the security issues of [RFC3986], section 7, should be considered.

In common with HTML, LASeR documents may reference external media such as images, audio, video, style sheets, and scripting languages. Scripting languages are executable content. In this case, the security considerations in the Media Type registrations for those formats shall apply.

In addition, because of the extensibility features for LASeR and of XML in general, it is possible that "application/laser " may describe content that has security implications beyond those described here. However, if the processor follows only the normative semantics of this specification, this content will be outside the LASeR namespaces and shall be ignored. Only in the case where the processor recognizes and processes the additional content, or where further processing of that content is dispatched to other processors, would security issues potentially arise. And in that case, they would fall outside the domain of this registration document.

Interoperability considerations:

This specification describes processing semantics that dictate behavior that must be followed when dealing with, among other things, unrecognized elements and attributes, both in the LASeR namespaces and in other namespaces.

Because LASeR is extensible, conformant "application/laser" processors must expect that content received is well-formed LASeR, but it cannot be guaranteed that the processor will recognize all of the elements and attributes in the document.

LASeR has a published Test Suite as part of ISO/IEC 14496-4:2001/AMD17.

Published specification:

This media type registration is extracted from Appendix B of ISO/IEC 14496-20.

Additional information:
Person & email address to contact for further information:

Leonardo Chiariglione, Olivier Avaro, Philippe de Cuetos, Jean-Claude Dufourd
(member-laser-media-types@googlegroups.com)

Intended usage:

COMMON

Author/Change controller:

The LASeR specification is a work product of the ISO/IEC/JTC1/SC29/WG11 (MPEG). ISO/MPEG has change control over these specifications.

B.5 Registration of Media Type application/saf

This media type is to be used for content packaged in SAF, not containing any LASeR stream.

MIME media type name:

application

MIME subtype name:

saf

Required parameters:

None

Optional parameters:

None

Encoding considerations:

Same as for application/xml. See [RFC3023] , section 3.2.

Restrictions on usage:

None

Security considerations:

This media type has no security implications but those of its packaged content.

Interoperability considerations:

SAF has a published Test Suite as part of ISO/IEC 14496-4:2001/AMD17.

Published specification:

This media type registration is extracted from Appendix B of ISO/IEC 14496-20.

Additional information:
Person & email address to contact for further information:

Leonardo Chiariglione, Olivier Avaro, Philippe de Cuetos, Jean-Claude Dufourd
(member-laser-media-types@googlegroups.com)

Intended usage:

COMMON

Author/Change controller:

The SAF specification is a work product of the ISO/IEC/JTC1/SC29/WG11 (MPEG). ISO/MPEG has change control over these specifications.

Bibliography

[1] Feiner, Foley, Hughes, van Dam, Computer Graphics: Principles and Practice, second edition in C, 1996 Addison-Wesley Publishing Company, Inc., New York

[2] W3C, Scalable Vector Graphics (SVG) Tiny 1.2 Specification [Last Call], http://www.w3.org/TR/2005/WD-SVGMobile12-20050413/

[3] Document Object Model (DOM) Level 3 Events Specification, Version 1.0, W3C Working Group Note 07 November 2003

[4] Digital Television Closed Captioning, Standard EIA-708-B, EIA

[5] The Java™ Language Specification, Third Edition, Addison-Wesley Professional, June 14, 2005, ISBN: 0321246780

[6] W3C XML Events, http://www.w3.org/TR/xml-events/