INTERNATIONAL STANDARD

**ISO/IEC 14496-20**

First edition
2006-06-15

# Information technology — Coding of audio-visual objects —

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

*Technologies de l'information — Codage des objets audiovisuels —*

*Partie 20: Représentation de scène d'application allégée (LASeR) et format d'agrégation simple (SAF)*



Reference number
ISO/IEC 14496-20:2006(E)

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

Published in Switzerland

 

# Contents

Page

 

# Foreword

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

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

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

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 reference software for coding of audio-visual objects* [Technical Report]

— *Part 8: Carriage of ISO/IEC 14496 contents over IP networks*

— *Part 9: Reference hardware description* [Technical Report]

— *Part 10: Advanced Video Coding*

— *Part 11: Scene description and application engine*

— *Part 12: ISO base media file format*

— *Part 13: Intellectual Property Management and Protection (IPMP) extensions*

— *Part 14: MP4 file format*

— *Part 15: Advanced Video Coding (AVC) file format*

— *Part 16: Animation Framework eXtension (AFX)*

— *Part 17: Streaming text format*

— *Part 18: Font compression and streaming*

— *Part 19: Synthesized texture stream*

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

— *Part 21: MPEG-J GFX*

— *Part 22: Open Font Format*

# 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.

This International Standard 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 — Part 1: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*

RFC 2045, *Multipurpose Internet Mail Extensions (MIME) Part one: Format of Internet message bodies*, 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*, 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]*, 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* 12 July 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**
individually accessible portion of data within a media stream

NOTE An access unit is the smallest data entity to which timing information can be attributed.

**3.1.2**
**media time line**
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**
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**

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

## 3.2 Abbreviations

**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

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.

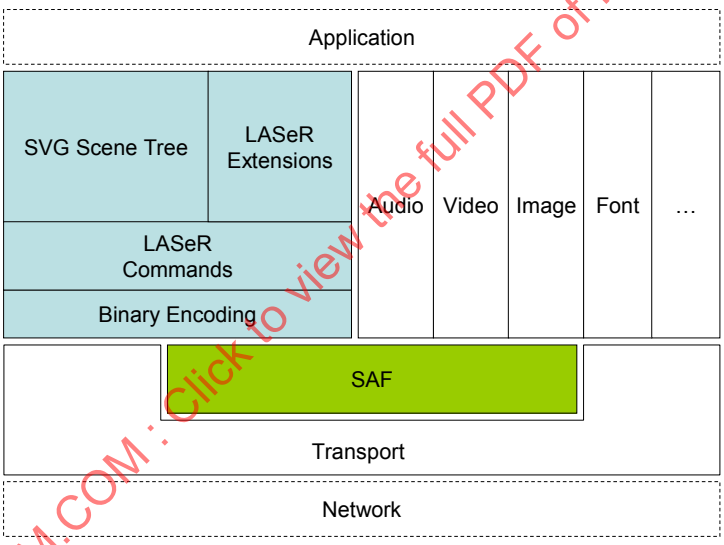


**Figure 1 — Architecture of LASeR and SAF**



## 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 6.8.37. 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] [2]. 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 6.6.2.3 a set of commands, called LASeR Commands, which can be applied to a LASeR scene.

### 6.2.2 Fonts

LASeR supports the encoding of fonts. Fonts shall be encoded separately from the scene, e.g. using ISO/IEC 14496-18, and sent as a media stream together with the scene stream. SVG elements related to font description are not supported by LASeR.

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. ISO/IEC 14496-18 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 ISO/IEC 14496-18 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

There are Scene Times, Wallclock Times, SMPTE timecodes, Media Times, and Encoded Scene Times. Wallclock times and SMPTE timecodes are not affected by the following discussion.

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).

Scene times within that access unit ("begin" in this example) are encoded for transmission relative to the scenetime of the access unit. In this example, the "begin" time X is transmitted as the Encoded Scene Time X-ST(r). For LASeR commands inside a script element, ST(r) is the scene time when the script 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.

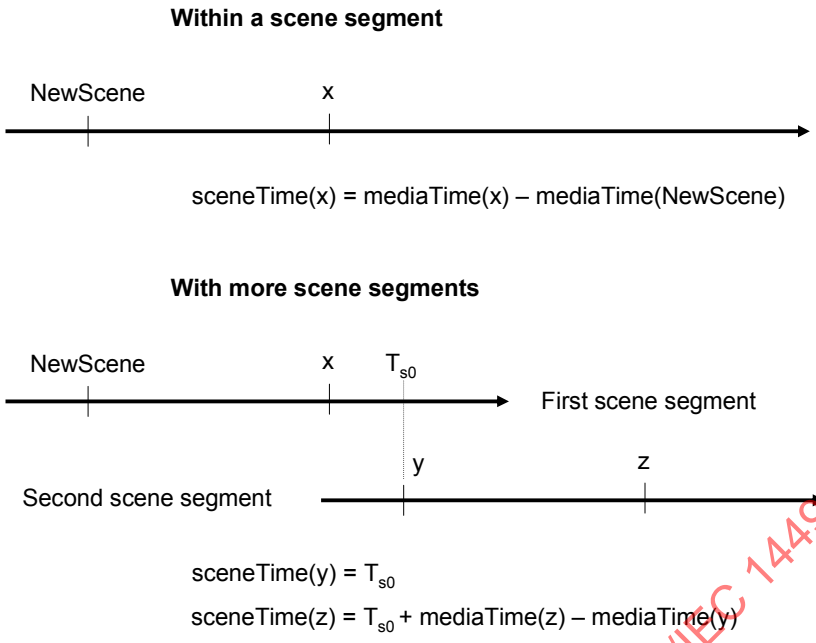


**Figure 2 — scene time and scene segments**

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 attribute of the LASeRHeader. Attributes “begin” and “end” are encoded as offset from the scene time of the current access unit. 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).

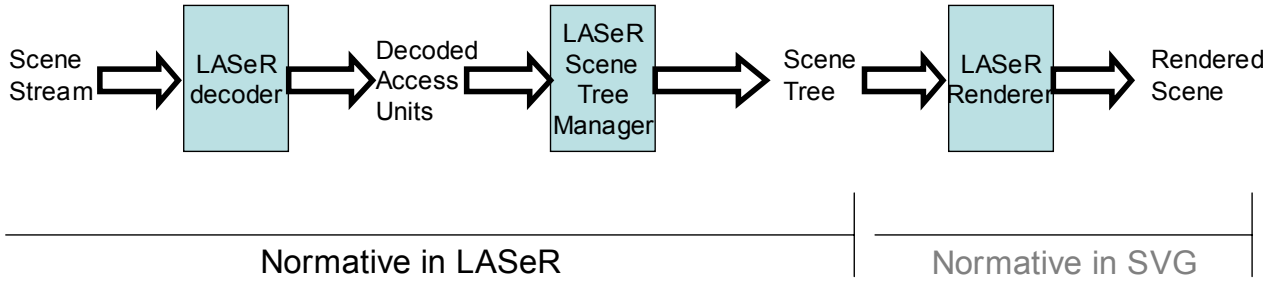


**Figure 3 — LASeR engine components and normative parts**

## 6.5 Supported Events

A LASeR engine supports the event model as specified in the DOM Level2 events specification [W3C DOM]0 with extensions compatible with the (informative) DOM Level 3 specification [3] and SVG Tiny 1.2 [2]. These extensions are: the definition of a namespace associated with each event; the naming of the events and of the animation events; and the notion of cancellability of an event.

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

**Table 1 — List of supported events**

| Event name | Namespace | Description | Bubble | Canc. |
|---|---|---|---|---|
| “focusin” (or deprecated “DOMFocusIn”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “focusout” (or deprecated “DOMFocusOut”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “activate” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “click” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “mousedown” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “mouseup” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “mouseover” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “mouseout” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | Yes |
| “mousemove” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “load” (or deprecated “SVGLoad”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | No | No |
| “resize” (or deprecated “SVGResize”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “scroll” (or deprecated “SVGScroll”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “zoom” (or deprecated “SVGZoom”) | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | No |
| “beginEvent” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | ??? |
| “endEvent” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | ??? |
| “repeatEvent” | http://www.w3.org/2001/xml-events | As defined in subclause 16.2 of [W3C SVG11]. | Yes | ??? |
| “keyup” | http://www.w3.org/2001/xml-events | reserved for future use | No | No |
| “keydown” | http://www.w3.org/2001/xml-events | reserved for future use | No | No |
| “textInput" | http://www.w3.org/2001/xml-events | reserved for future use | No | No |

| | | | | |
|---|---|---|---|---|
| “accessKey(keyCode)” | urn:mpeg:mpeg4:laser:2005 | The key keyCode has been pressed, as defined in subclause 6.4 of [SMIL2] | No | No |
| “longAccessKey(keyCode)” | urn:mpeg:mpeg4:laser:2005 | Similar to accessKey but for the fact that the event is only triggered if the key has been pressed for a longer time, the definition of “longer” being left to the appreciation of the browser implementation. | No | No |
| “pause” | urn:mpeg:mpeg4:laser:2005 | Freezes the clock of the timed object they are sent to, and have no effect on non timed objects. | No | No |
| “resume” | urn:mpeg:mpeg4:laser:2005 | Restarts the clock of the timed object they are sent to, and have no effect on non timed objects. | No | No |

The value of keyCode in Table 1 is defined in Table 2

**Table 2 — Defined key codes**

| Key Name | Key Code | Comment |
|---|---|---|
| KEY_UP | 0 | |
| KEY_DOWN | 1 | |
| KEY_LEFT | 2 | |
| KEY_RIGHT | 3 | |
| KEY_ENTER | 4 | also called FIRE sometimes |
| NO_KEY | 5 | special value disabling the wait for a key |
| ANY_KEY | 6 | matches any key |
| SOFT_KEY_1 | 7 | soft key n°1 (usually below the screen to the left) |
| SOFT_KEY_2 | 8 | soft key n°2 (usually below the screen to the right) |
| KEY_POUND | 35 | # |
| KEY_STAR | 42 | * |
| KEY_0 | 48 | |
| KEY_1 | 49 | |
| KEY_2 | 50 | |
| KEY_3 | 51 | |
| KEY_4 | 52 | |
| KEY_5 | 53 | |
| KEY_6 | 54 | |
| KEY_7 | 55 | |
| KEY_8 | 56 | |
| KEY_9 | 57 | |



## 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

#### 6.6.2.1 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, the other scene segments having the ***append*** bit set and 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 shall cache permanent streams and selected scene information on a best effort basis. The principles behind this caching closely follow the state caching mechanism in HTTP, commonly called cookies [RFC 2695]. 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 and permanent streams 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.

#### 6.6.2.2 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.
- ***append***: this Boolean attribute defines whether the scene segment starting with this LASeRHeader is an addition to the scene already present in the LASeR engine, or if it defines a new scene altogether. The combination of append and the NewScene command is defined in Table 3.

**Table 3 — Behavior of combinations of append and NewScene**

| Append value | type of the first LASeR Command following the LASeRHeader | Behavior |
|---|---|---|
| true | NewScene | the new scene segment defines a new scene, the append value is ignored |
| false | | |
| true | other command | the previous scene is kept |
| false | other command | the behavior is undefined |

- ***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”.
- **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.
- ***hasStringIds***: this Boolean attribute defines whether ids are carried as strings, allowing XML canonical reconstruction.

#### 6.6.2.3 Attributes of LASeRUnitHeader

- ***resetEncodingContext:*** when this Boolean attribute is true, the encoding context defined in subclause 6.6.2.4 shall be reset upon reception of this LASeRUnitHeader.

#### 6.6.2.4 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.

## 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.

The following restrictions apply to all commands:

- Commands shall refer to existing elements and attributes.
- The following attributes cannot be updated: 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.

### 6.7.2 Add

#### 6.7.2.1 Semantics

The ***Add*** command is similar to ***Attribute Replacement*** defined in subclause 6.7.8.1, 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>
```

#### 6.7.2.2 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.

#### 6.7.2.3 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

#### 6.7.3.1 Semantics

The ***Clean*** command erases the storing area identified by the attribute ***groupID*** with the most specific matching defined in 6.6.2.1. The element information stored in the corresponding memory area is not available anymore.

#### 6.7.3.2 Attributes

- ***groupID***: this string attribute defines the group ID as defined in 6.6.2.1.

### 6.7.4 Delete

#### 6.7.4.1 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.

#### 6.7.4.2 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***: this attribute defines the name of the attribute in which the deletion happens, by default "children". This attribute is only allowed in conjunction with index.

#### 6.7.4.3 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

#### 6.7.5.1 Semantics

The ***Insert*** command inserts an element in a parent list.

Examples (fragments):

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

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

    <Insert href="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 href="Shape4" attribute="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***.

#### 6.7.5.2 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.
- ***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.

#### 6.7.5.3 Children

Any element.

#### 6.7.5.4 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

#### 6.7.6.1 Semantics

The ***NewScene*** command inserts a new scene in the browser. 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.

#### 6.7.6.2 Attributes

No attributes

#### 6.7.6.3 Children

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

#### 6.7.6.4 DOM Formulation

This subclause is informative.

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

### 6.7.7 RefreshScene

#### 6.7.7.1 Semantics

The **RefreshScene** command provides a functionally identical copy of the current scene, to a browser 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 browsers in other states (i.e. those that have a scene for which they have received all information) this command shall be ignored. For browsers 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 browser 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 browsers 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.

#### 6.7.7.2 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.

#### 6.7.7.3 Children

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

#### 6.7.7.4 DOM Formulation

This subclause is informative.

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

### 6.7.8 Replace

#### 6.7.8.1 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 ***attribute*** 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.

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.

#### 6.7.8.2 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.

#### 6.7.8.3 Children

Any element or text content

#### 6.7.8.4 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

#### 6.7.9.1 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.

#### 6.7.9.2 Attributes

- ***groupID***: this string attribute defines the group ID as defined in 6.6.2.1.

### 6.7.10 Save

#### 6.7.10.1 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.

#### 6.7.10.2 Attributes

- ***groupID***: this string attribute defines the group as defined in 6.6.2.1.
- ***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.

### 6.7.11 SendEvent

#### 6.7.11.1 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.

#### 6.7.11.2 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

## 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 6.8.37. The list of possible children is given in the summary table in subclause 6.8.37. 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].

### 6.8.4 SVG animate

#### 6.8.4.1 Semantics

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

#### 6.8.4.2 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.

### 6.8.5 SVG animateColor

#### 6.8.5.1 Semantics

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

#### 6.8.5.2 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.

### 6.8.6 SVG animateMotion

#### 6.8.6.1 Semantics

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

#### 6.8.6.2 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.

### 6.8.7 SVG animateTransform

#### 6.8.7.1 Semantics

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

#### 6.8.7.2 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.

### 6.8.8 SMIL audio

#### 6.8.8.1 Semantics

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

#### 6.8.8.2 Attributes

- clipBegin: 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.

- 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.

- 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 SVG cursor

The ***SVG cursor*** element is defined in subclause 16.8.3 of [W3C SVG11].

The cursor element may point to any LASeR element to confer it the semantics of virtual pointer. This specification extends the behavior described in [W3C SVG11] in the following manner.

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:

```
<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>

<cursor id="cursor1" xlink:href="#vp">
  <!-- pointer movement -->
  <ev:listener event="accessKey(UP)" handler="#s1"/>
  <script id="s1">
    <lsr:Add ref="g" attribute="translation" pointvalue="0 -5"/>
  </script>
  <ev:listener event="accessKey(DOWN)" handler="#s2"/>
  <script id="s2">
    <lsr:Add ref="g" attribute="translation" pointvalue="0 5"/>
  </script>
  <ev:listener event="accessKey(LEFT)" handler="#s3"/>
  <script id="s3">
    <lsr:Add ref="g" attribute="translation" pointvalue="-5 0"/>
  </script>
  <ev:listener event="accessKey(RIGHT)" handler="#s4"/>
  <script id="s4">
    <lsr:Add ref="g" attribute="translation" pointvalue="5 0"/>
  </script>
</cursor>

<!-- activate event to be translated to a click -->
<ev:listener event="accessKey(FIRE)" handler="#cursor1"/>

<!-- mouse sensitive shape -->
<polygon …>
   <ev:listener event="click" handler="someScript"/>
</polygon>
```

### 6.8.11 SVG defs

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

### 6.8.12 SVG desc

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

### 6.8.13 SVG ellipse

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

### 6.8.14 SVG foreignObject

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

### 6.8.15 SVG g

#### 6.8.15.1 Semantics

The semantics of the ***SVG g*** element are based on those of subclause 5.2 of [W3C SVG11]. Extra semantics are added:

- it may act as a selection element, rendering zero or one of its children,
- it may act as a clipping element, limiting the rendering of its children to a rectangle (whose borders are parallel to the screen borders),
- and it may act as a simple layout tool, by spacing its children by a specified amount, thus creating rows or columns of children.

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

- ***choice*** >= 0 & ***choice*** < N: only the child of index choice is displayed
- ***choice*** == none | ***choice*** >= N: nothing is displayed
- ***choice*** == delta & ***size*** != null: 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. This creates a row or column of objects.
- ***choice*** == clip & ***size*** != null: all children are clipped by an 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.
- in all other cases: all the children are displayed at (0,0) of the local coordinate system without clipping.

#### 6.8.15.2 Attributes

- **size:** a pair of coordinates which, depending on the value of choice, can be used as a column or row spacing, or as clipping rectangle size. This attribute is animatable but not inheritable.
- **choice:** the rendering mode selector. This attribute is animatable but not inheritable.

### 6.8.16 SVG image

#### 6.8.16.1 Semantics

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

#### 6.8.16.2 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 6.8.36. This attribute is not animatable and not inheritable.

### 6.8.17 SVG line

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

### 6.8.18 SVG linearGradient

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

### 6.8.19 XML Events listener

#### 6.8.19.1 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 [2] and also adds compatible extensions.

#### 6.8.19.2 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 the supported handlers and the default action to be performed:

| Event | Element | Action |
|---|---|---|
| activate | any element with timing attributes | Depends on the value of timeAttribute. |
| activate | a | follow the hyperlink |
| activate | script | executes the script content |
| activate | text with editable="true" | starts the editing process for timeAttribute='begin' |
| activate | text with editable="true" | aborts the editing process for timeAttribute='end' |
| pause | any element with timing attributes | freezes the clock of this element |
| resume | any element with timing attributes | releases the clock of this element |

- **enabled:** this Boolean attribute specifies whether the listener is currently active or not. This attribute is not animatable and not inheritable.

- **timeAttribute**: the possible values are 'begin' and 'end'. This attribute defines the action to be performed when an 'activate' event happens and the target element is an element with timing attributes or an editable text. When the timeAttribute is specified and the handler attribute of the listener element does not point to an element with begin and end attributes or to an editable text element, then the timeAttribute value is ignored. The default value is 'begin'. This attribute is not animatable and not inheritable.

- **delay:** the optional delay before activation in seconds. Note: consecutive events are delayed independently from the value of delay. This attribute is not animatable and not inheritable.

#### 6.8.19.3 Children

None.

### 6.8.20 SVG metadata

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

### 6.8.21 SVG mpath

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

### 6.8.22 SVG path

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

### 6.8.23 SVG polygon

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

### 6.8.24 SVG polyline

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

### 6.8.25 SVG radialGradient

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

### 6.8.26 SVG rect

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

### 6.8.27 SVG script

#### 6.8.27.1 Semantics

The ***SVG script*** element is defined in subclause 18.2 of [W3C SVG11]. In the context of this specification, it allows sets of scene updates to be inserted in the scene, for later execution upon activation by time or through the ***XML Events listener*** element.

#### 6.8.27.2 Attributes

- begin: this attribute specifies the time at which the script element is triggered. This attribute is not animatable and not inheritable.
- enabled: this Boolean attribute specifies whether the element is activatable or not. This attribute is not animatable and not inheritable.

#### 6.8.27.3 Children

If the ***SVG type*** attribute is “application/laserscript”, the possible children are LASeR Commands to apply to the scene upon activation of the ***SVG script***.

If the ***SVG type*** attribute is “application/ecmascript”, the usage of this element is the same as in SVG.

### 6.8.28 SVG set

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

### 6.8.29 SVG stop

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

### 6.8.30 SVG svg

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

#### 6.8.30.1 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].

### 6.8.31 SVG switch

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

### 6.8.32 SVG text

#### 6.8.32.1 Semantics

The ***SVG text*** element is defined in subclause 10.4 of [W3C SVG11], with the following extension: through its extended values, the attribute ***font-style*** combines the features of ***SVG font-style***, ***SVG font-weight*** and ***SVG text-decoration***.

#### 6.8.32.2 Attributes

- additional values for font-style: when this attribute has SVG values (“normal”, “italic”, “oblique” and “inherit”), the SVG behaviour applies. The following additional keywords are taken from [4]: “PLAIN”, “ITALIC”, “BOLD”, “BOLDITALIC”, “UNDERLINE”, “OUTLINE”, “EMBOSS”, “ENGRAVE”, “LEFTDROPSHADOW”, “RIGHTDROPSHADOW”.
- 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 browser. If set to "true", the browser 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 [2], i.e. it governs alignment in the direction orthogonal to the main direction of the text.

### 6.8.33 SVG title

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

### 6.8.34 SVG tspan

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

### 6.8.35 SVG use

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

### 6.8.36 SMIL video

#### 6.8.36.1 Semantics

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

#### 6.8.36.2 Attributes

- clipBegin: 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.
- 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.
- overlay: the following value is added to the SVG list of possible values for overlay: “fullscreen”. The semantic of this value is that the video is rendered alone in the rendering area, possibly filling the whole rendering area. 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.
- 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. |



### 6.8.37 Summary of Possible Children and 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 |
| 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 |
| 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 |
| cursor | class id x xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space 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 |

| 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 | audio-level choice 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 size 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 |
| 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 |
| 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 delay event handler observer phase timeAttribute propagate defaultAction target |
| metadata | class id xml:base xml:lang xml:space |
| mpath | class id xlink:actuate xlink:arcrole xlink:href xlink:role xlink:show xlink:title xlink:type xml:base xml:lang xml:space |
| 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 |

| Element name | Attributes |
|---|---|
| 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 |
| 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 |
| 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 |
| 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 |
| 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 |

| Element name | Attributes |
|---|---|
| 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 |
| 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 |
| video | audio-level begin display dur end 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 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 |



# 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 4

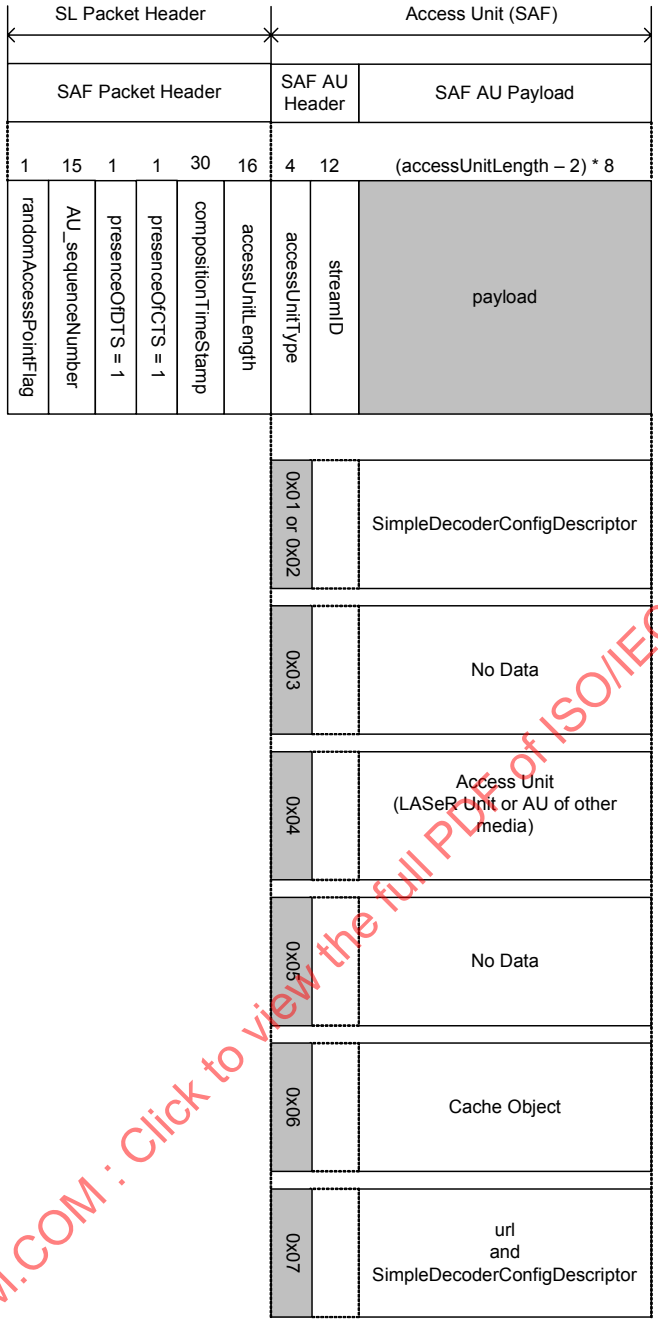


**Figure 4 — SAF Packet architecture**



### 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) AU_sequenceNumber;
    const bit(1) presenceOfDTS = 0;
    const bit(1) presenceOfCTS = 1;
    bit(30) compositionTimeStamp;
    uint(16) accessUnitLength;
}
```

### 7.4.2 Semantics

`AU_sequenceNumber` – if present, successive access units shall either have the same sequence number or the value be continuously incremented as a modulo counter.

`randomAccessPointFlag` – when set to one indicates that random access to the content of this elementary stream is possible here.

`compositionTimeStamp` – is a composition time stamp. The composition time tc of the first composition unit resulting from this access unit is reconstructed from this composition time stamp according to the formula:

$$tc = (compositionTimeStamp/timeStampResolution + k * 2^{32}/timeStampResolution)$$

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. When the packet conveys a cacheUnit, this field specifies the validity period in second.

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

`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. Values 0 and 1 are reserved for future ISO use.

## 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 | StreamHeader | A SimpleDecoderConfigDescriptor |
| 0x02 | StreamHeader (permanent[a]) | 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 ~ 0x0F | Reserved | - |

[a] "permanent" indicates that the payloads of the SAF access units of this stream shall be stored beyond the life of the current scene for a duration stored in the `compositionTimeStamp` of this SAF Packet.

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

`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 StreamHeader, the `payload` shall convey a SimpleDecoderConfigDescriptor whose syntax and semantics is specified in 7.6.

For values of `safAU.accessUnitType` that refer to LASeR scene unit or cache unit, the `payload` shall convey a scene 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 SAF 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. The payload data shall replace the stored presentation referenced by the url of this cache unit when the `safAU.accessUnitType` is `0x06`. Otherwise, the payload data shall be appended to the existing presentation.

### 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.

## 8 Profiles

### 8.1 Overview

The LASeR mini and full profiles are defined in the SceneGraph dimension of MPEG-4 Systems profiles.

### 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.

#### 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.

### 8.2.3 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 lsr:rotation lsr:scale lsr:translation transform xlink:href |
| animate | id enabled begin end dur repeatCount repeatDur restart **attributeName fill=”freeze”** to xlink:href from by values calcMode keyTimes keySplines **additive=”replace”** accumulate |
| animateColor | id enabled begin end dur repeatCount repeatDur restart **attributeName fill=”freeze”**to xlink:href from by values calcMode keyTimes keySplines **additive=”replace”** accumulate |
| animateMotion | id path keyPoints rotate enabled begin end dur repeatCount repeatDur restart **fill=”freeze”** to xlink:href from by values calcMode keyTimes keySplines **additive=”replace”** accumulate |
| animateTransform | id type enabled begin end dur repeatCount repeatDur restart attributeName **fill=”freeze”**to xlink:href 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 lsr:rotation lsr:scale lsr:translation transform |
| 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 lsr:rotation lsr:scale lsr:translation transform |
| foreignObject | id externalResourcesRequired width height x y viewport-fill viewport-fill-opacity display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats |
| g | id externalResourcesRequired choice size display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation transform |
| image | id externalResourcesRequired width height x y **transformBehavior(no geometric)** opacity display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation transform xlink:href |
| line | id x1 y1 x2 y2 display stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation transform |
| linearGradient | id gradient-units x1 x2 y1 y2 display visibility |

| Element name | Attributes |
|---|---|
| ev:listener | id 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 lsr:rotation lsr:scale lsr:translation transform |
| polygon | id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation transform |
| polyline | id points display fill fill-opacity fill-rule stroke stroke-opacity stroke-width visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation 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 rotation scale translation transform |
| script | externalResourcesRequired id begin type enabled |
| set | id enabled begin end dur repeatCount repeatDur restart **attributeName fill="freeze"** to xlink:href |
| 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 lsr:rotation lsr:scale lsr:translation transform |
| 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 |
| use | id externalResourcesRequired overflow x y display visibility requiredFeatures requiredExtensions systemLanguage requiredFormats lsr:rotation lsr:scale lsr:translation 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 \| fullscreen" transformBehavior(no geometric)** lsr:rotation lsr:scale lsr:translation 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.

# 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 9.

**Table 9 — 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 `LaserHeader`.

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 5. Since RFC 3640 allows aggregation of SL packets, CTS-delta may be present if several SAF_packets are carried by one RTP packet.

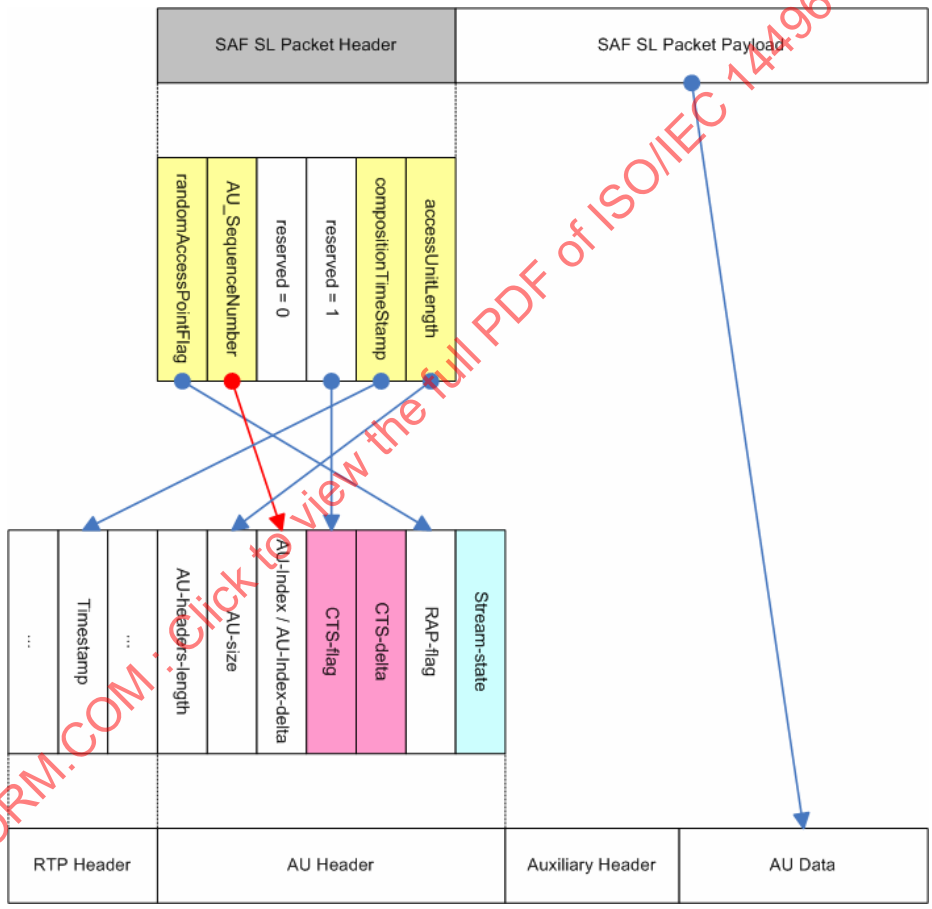




**Figure 5 — Mapping of SAF_SL_packet to RTP payload format in RFC 3640**

## 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

# 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 6.

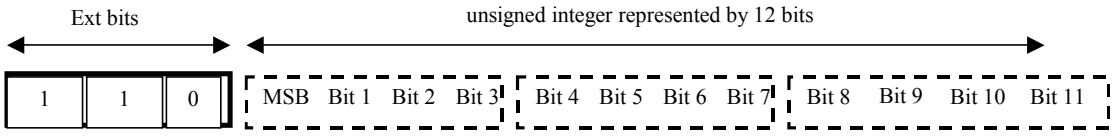


**Figure 6 — Example for the vluimsbf5 data type**

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:

| | | |
|---|---|---|
| xx | xy | xz |
| yx | yy | yx |
| 0 | 0 | 1 |

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 10 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 10 — Code points for private data encoding**

| codePoint | privateDataContainer | extConfiguration |
|---|---|---|
| | *defines encoding* | *defines configuration for* |
| 0 | anyXML as defined in … | not used |
| 1 | opaque binary data | opaque binary data |
| 2 | reserved for future use with ISO/IEC 23001-1 | reserved for future use with ISO/IEC 23001-1 |
| 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.

### 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 11. The correspondance of attribute to encoded type is defined in Table 12.

**Table 11 — 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 |
| ANIMTYPE_ints | 7 |
| ANIMTYPE_floats | 8 |
| ANIMTYPE_point | 9 |
| ANIMTYPE_id | 10 |
| ANIMTYPE_font | 11 |
| ANIMTYPE_uri | 12 |

**Table 12 — Attribute to encoded type correspondance**

| Attribute name | Encoded type for animation | Attribute name | Encoded type for animation |
|---|---|---|---|
| audio-level | ANIMTYPE_fraction | r | ANIMTYPE_float |
| choice | ANIMTYPE_enum | rotate | ANIMTYPE_float |
| color | ANIMTYPE_color | rx | ANIMTYPE_float |
| color-rendering | ANIMTYPE_enum | ry | ANIMTYPE_float |
| cx | ANIMTYPE_float | shape-rendering | ANIMTYPE_enum |
| cy | ANIMTYPE_float | size | ANIMTYPE_point |
| d | ANIMTYPE_path | stop-color | ANIMTYPE_color |
| display | ANIMTYPE_enum | stop-opacity | ANIMTYPE_fraction |
| display-align | ANIMTYPE_enum | stroke | ANIMTYPE_color |
| editable | ANIMTYPE_enum | stroke-dasharray | ANIMTYPE_floats |
| fill | ANIMTYPE_color | stroke-dashoffset | ANIMTYPE_float |
| fill-opacity | ANIMTYPE_fraction | stroke-linecap | ANIMTYPE_enum |
| fill-rule | ANIMTYPE_enum | stroke-linejoin | ANIMTYPE_enum |
| nav-right | ANIMTYPE_id | stroke-miterlimit | ANIMTYPE_float |
| nav-up | ANIMTYPE_id | stroke-opacity | ANIMTYPE_fraction |
| nav-up-right | ANIMTYPE_id | stroke-width | ANIMTYPE_float |
| nav-up-left | ANIMTYPE_id | target | ANIMTYPE_string |
| nav-down | ANIMTYPE_id | text-anchor | ANIMTYPE_enum |
| nav-down-right | ANIMTYPE_id | transform | ANIMTYPE_floats |
| nav-down-left | ANIMTYPE_id | type | ANIMTYPE_enum |
| nav-left | ANIMTYPE_id | vector-effect | ANIMTYPE_enum |
| focusable | ANIMTYPE_id | viewBox | ANIMTYPE_floats |
| font-family | ANIMTYPE_font | viewport-fill | ANIMTYPE_color |
| font-size | ANIMTYPE_float | viewport-fill-opacity | ANIMTYPE_fraction |
| font-style | ANIMTYPE_enum | visibility | ANIMTYPE_enum |
| font-weight | ANIMTYPE_enum | width | ANIMTYPE_float |
| gradientUnits | ANIMTYPE_enum | x | ANIMTYPE_float |
| height | ANIMTYPE_float | x1 | ANIMTYPE_float |
| image-rendering | ANIMTYPE_enum | x2 | ANIMTYPE_float |
| opacity | ANIMTYPE_fraction | xlink:href | ANIMTYPE_uri |
| pathLength | ANIMTYPE_float | y | ANIMTYPE_float |
| pointer-events | ANIMTYPE_ints | y1 | ANIMTYPE_float |
| points | ANIMTYPE_pointSeq | y2 | ANIMTYPE_float |
| preserveAspectRatio | ANIMTYPE_enum | | |



### 12.1.5 Encoding of enumerations

Code points associated with the following SDL structures:

- the bit field ‘AttributeName’ in class ‘attr_AttributeName’
- the variable ‘attributeIndex’ in class ‘attr_custom_updateValue’

The third column is informative and gives the originating specification.

| Code point | Attribute Name | Specification |
|---|---|---|
| 0 | accumulate | SVG 1.1 |
| 1 | additive | SVG 1.1 |
| 2 | append | LASeR |
| 3 | attributeName | SVG 1.1 |
| 4 | audio-level | SVG Tiny 1.2 |
| 5 | bandwidth | SMIL 2.0 |
| 6 | begin | SVG 1.1 |
| 7 | by | SVG 1.1 |
| 8 | calcMode | SVG 1.1 |
| 9 | children | LASeR |
| 10 | choice | LASeR |
| 11 | color | SVG 1.1 |
| 12 | color-rendering | SVG 1.1 |
| 13 | cx | SVG 1.1 |
| 14 | cy | SVG 1.1 |
| 15 | d | SVG 1.1 |
| 16 | delay | LASeR |
| 17 | display | SVG 1.1 |
| 18 | display-align | SVG Tiny 1.2 |
| 19 | dur | SVG 1.1 |
| 20 | editable | SVG Tiny 1.2 |
| 21 | enabled | LASeR |
| 22 | end | SVG 1.1 |
| 23 | event | XML Events |
| 24 | externalResourcesRequired | SVG 1.1 |
| 25 | fill | SVG 1.1 |
| 26 | fill-opacity | SVG Tiny 1.2 |
| 27 | fill-rule | SVG 1.1 |
| 28 | focusable | SVG Tiny 1.2 |
| 29 | font-family | SVG Tiny 1.2 |
| 30 | font-size | SVG Tiny 1.2 |
| 31 | font-style | SVG Tiny 1.2 |
| 32 | font-variant | SVG Tiny 1.2 |
| 33 | font-weight | SVG Tiny 1.2 |
| 34 | from | SVG Tiny 1.2 |
| 35 | gradientUnits | SVG Tiny 1.2 |
| 36 | handler | SVG Tiny 1.2 |
| 37 | height | SVG Tiny 1.2 |
| 38 | image-rendering | SVG Tiny 1.2 |
| 39 | keyCodes | SVG 1.1 |
| 40 | keyPoints | SVG 1.1 |
| 41 | keySplines | SVG 1.1 |
| 42 | keyTimes | SVG 1.1 |
| 43 | line-increment | SVG 1.1 |
| 44 | mediaCharacterEncoding | SVG 1.1 |
| 45 | mediaContentEncodings | SVG 1.1 |
| 46 | mediaSize | XML Events |

| Code point | Attribute Name | Specification |
|---|---|---|
| 47 | mediaTime | SVG 1.1 |
| 48 | nav-down | SVG 1.1 |
| 49 | nav-down-left | SVG 1.1 |
| 50 | nav-down-right | SVG 1.1 |
| 51 | nav-left | SVG 1.1 |
| 52 | nav-next | SVG 1.1 |
| 53 | nav-prev | SVG Tiny 1.2 |
| 54 | nav-right | SVG Tiny 1.2 |
| 55 | nav-up | SVG Tiny 1.2 |
| 56 | nav-up-left | SMIL 2.0 |
| 57 | nav-up-right | SMIL 2.0 |
| 58 | observer | XML Events |
| 59 | offset | SVG 1.1 |
| 60 | opacity | SVG 1.1 |
| 61 | overflow | SVG 1.1 |
| 62 | overlay | SVG Tiny 1.2 |
| 63 | path | SVG 1.1 |
| 64 | pathLength | SVG 1.1 |
| 65 | pointer-events | SVG 1.1 |
| 66 | points | SVG 1.1 |
| 67 | preserveAspectRatio | SVG 1.1 |
| 68 | r | SVG 1.1 |
| 69 | repeatCount | SVG 1.1 |
| 70 | repeatDur | SVG 1.1 |
| 71 | requiredExtensions | SVG 1.1 |
| 72 | requiredFeatures | SVG 1.1 |
| 73 | requiredFormats | SVG 1.1 |
| 74 | restart | SVG 1.1 |
| 75 | rotate | SVG 1.1 |
| 76 | rotation | LASeR |
| 77 | rx | SVG 1.1 |
| 78 | ry | SVG 1.1 |
| 79 | scale | LASeR |
| 80 | shape-rendering | SVG 1.1 |
| 81 | size | LASeR |
| 82 | solid-color | SVG Tiny 1.2 |
| 83 | solid-opacity | SVG Tiny 1.2 |
| 84 | stop-color | SVG 1.1 |
| 85 | stop-opacity | SVG 1.1 |
| 86 | stroke | SVG 1.1 |
| 87 | stroke-dasharray | SVG 1.1 |
| 88 | stroke-dashoffset | SVG 1.1 |
| 89 | stroke-linecap | SVG 1.1 |
| 90 | stroke-linejoin | SVG 1.1 |
| 91 | stroke-miterlimit | SVG 1.1 |
| 92 | stroke-opacity | SVG 1.1 |
| 93 | stroke-width | SVG 1.1 |

| Code point | Attribute Name | Specification |
|---|---|---|
| 94 | width (of svg element) | SVG 1.1 |
| 95 | height (of svg element) | SVG 1.1 |
| 96 | syncBehavior | SVG Tiny 1.2 |
| 97 | syncBehaviorDefault | SVG Tiny 1.2 |
| 98 | syncReference | LASeR |
| 99 | syncTolerance | SVG Tiny 1.2 |
| 100 | syncToleranceDefault | SVG Tiny 1.2 |
| 101 | systemLanguage | SVG 1.1 |
| 102 | text-anchor | SVG 1.1 |
| 103 | text-rendering | SVG 1.1 |
| 104 | textContent | LASeR |
| 105 | timeAttribute | LASeR |
| 106 | to | SVG 1.1 |
| 107 | transform | SVG 1.1 |
| 108 | transformBehavior | SVG Tiny 1.2 |
| 109 | translation | LASeR |
| 110 | type | SVG 1.1 |
| 111 | values | SVG 1.1 |
| 112 | vector-effect | SVG Tiny 1.2 |
| 113 | viewBox | SVG 1.1 |
| 114 | viewport-fill | SVG Tiny 1.2 |
| 115 | viewport-fill-opacity | SVG Tiny 1.2 |

| Code point | Attribute Name | Specification |
|---|---|---|
| 116 | visibility | SVG 1.1 |
| 117 | width | SVG 1.1 |
| 118 | x | SVG 1.1 |
| 119 | x1 | SVG 1.1 |
| 120 | x2 | SVG 1.1 |
| 121 | xlink:actuate | SVG 1.1 |
| 122 | xlink:arcrole | SVG 1.1 |
| 123 | xlink:href | SVG 1.1 |
| 124 | xlink:role | SVG 1.1 |
| 125 | xlink:show | SVG 1.1 |
| 126 | xlink:title | SVG 1.1 |
| 127 | xlink:type | SVG 1.1 |
| 128 | xml:base | SVG 1.1 |
| 129 | xml:lang | SVG 1.1 |
| 130 | xml:space | SVG 1.1 |
| 131 | y | SVG 1.1 |
| 132 | y1 | SVG 1.1 |
| 133 | y2 | SVG 1.1 |
| 134 | zoomAndPan | SVG 1.1 |
| 135 – 255 | ISO reserved | |

Code points associated with ‘accumulate’ bit in class ‘attr_accumulate’:

| Code point | Attribute value |
|---|---|
| 0 | none |
| 1 | sum |

Code points associated with ‘additive’ bit in class ‘attr_additive’:

| Code point | Attribute value |
|---|---|
| 0 | replace |
| 1 | sum |

Code points associated with ‘calcMode’ bit field in class ‘attr_calcMode’:

| Code point | Attribute value |
|---|---|
| 0 | discrete |
| 1 | linear |
| 2 | paced |
| 3 | spline |

Code points associated with ‘animFill’ bit in class ‘attr_animFill’:

| Code point | Attribute value |
|---|---|
| 0 | freeze |
| 1 | remove |

Code points associated with ‘restart’ bit field in class ‘attr_restart’:

| Code point | Attribute value |
|---|---|
| 0 | always |
| 1 | never |
| 2 | whenNotActive |
| 3 | ISO reserved |

Code points associated with ‘rotate’ bit in class ‘attr_rotate’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | auto-reverse |

Code points associated with ‘rotscatra’ bit field in class ‘attr_rotscatra’:

| Code point | Attribute value |
|---|---|
| 0 | rotate |
| 1 | scale |
| 2 | skewX |
| 3 | skewY |
| 4 | translate |
| 5 – 7 | ISO reserved |

Code points associated with bit field ‘syncBehavior’ in class ‘attr_syncBehavior’:

| Code point | Attribute value |
|---|---|
| 0 | canSlip |
| 1 | default |
| 2 | independent |
| 3 | locked |

Code points associated with 2-bits ‘choice’ in class ‘attr_choice’:

| Code point | Attribute value |
|---|---|
| 0 | all |
| 1 | clip |
| 2 | delta |
| 3 | none |

Code points associated with ‘transformBehavior’ bit field in class ‘attr_transformBehavior’:

| Code point | Attribute value |
|---|---|
| 0 | geometric |
| 1 | pinned |
| 2 | pinned_180 |
| 3 | pinned_270 |
| 4 | pinned_90 |
| 5 – 7 | ISO reserved |

Code points associated with bit ‘gradientUnits’ class ‘attr_gradientUnits:

| Code point | Attribute value |
|---|---|
| 0 | objectBoundingBox |
| 1 | userSpaceOnUse |

Code points associated with ‘script’ bit field in class ‘attr_element_script’:

| Code point | Attribute value |
|---|---|
| 0 | application/ecmascript |
| 1 | application/jar-archive |
| 2 | application/laserscript |
| 3 | ISO reserved |

Code points associated with bit ‘playbackOrder’ in class ‘attr_playbackOrder:

| Code point | Attribute value |
|---|---|
| 0 | all |
| 1 | forwardOnly |

Code points associated with bit field ‘preserveAspectRatio1’ in class ‘attr_preserveAspectRatio’:

| Code point | Attribute value |
|---|---|
| 0 | none |
| 1 | xMaxYMax |
| 2 | xMaxYMid |
| 3 | xMaxYMin |
| 4 | xMidYMax |
| 5 | xMidYMid |
| 6 | xMidYMin |
| 7 | xMinYMax |
| 8 | xMinYMid |
| 9 | xMinYMin |
| 10-15 | ISO Reserved |

Code points associated with bit field ‘preserveAspectRatio2’ in class ‘attr_preserveAspectRatio’:

| Code point | Attribute value |
|---|---|
| 0 | ISO Reserved |
| 1 | defer xMaxYMax |
| 2 | defer xMaxYMid |
| 3 | defer xMaxYMin |
| 4 | defer xMidYMax |
| 5 | defer xMidYMid |
| 6 | defer xMidYMin |
| 7 | defer xMinYMax |
| 8 | defer xMinYMid |
| 9 | defer xMinYMin |
| 10-15 | ISO Reserved |

Code points associated with bit field ‘preserveAspectRatio3’ in class ‘attr_preserveAspectRatio’:

| Code point | Attribute value |
|---|---|
| 0 – 31 | ISO Reserved |

Code points associated with bit field ‘syncBehaviorDefault’ in class ‘attr_syncBehaviorDefault’:

| Code point | Attribute value |
|---|---|
| 0 | canSlip |
| 1 | independent |
| 2 | inherit |
| 3 | locked |

Code points associated with bit field ‘timeLineBegin’ in class ‘attr_timeLineBegin’:

| Code point | Attribute value |
|---|---|
| 0 | onLoad |
| 1 | onStart |

Code points associated with bit field ‘zoomAndPan’ in class ‘attr_zoomAndPan:

| Code point | Attribute value |
|---|---|
| 0 | disable |
| 1 | magnify |

Code points associated with bit field ‘overflow’ in class ‘attr_overflow’:

| Code point | Attribute value |
|---|---|
| 0 | visible |
| 1 – 4 | ISO reserved |

Code points associated with bit field ‘overlay’ in class ‘attr_overlay’:

| Code point | Attribute value |
|---|---|
| 0 | fullscreen |
| 1 | none |
| 2 | top |
| 3 | ISO reserved |

Code points associated with bit ‘defaultAction’ in class ‘attr_defaultAction’:

| Code point | Attribute value |
|---|---|
| 0 | cancel |
| 1 | perform |

Code points associated with bit ‘phase’ in class ‘attr_phase’:

| Code point | Attribute value |
|---|---|
| 0 | default |
| 1 | ISO reserved |

Code points associated with bit ‘propagate’ in class ‘attr_propagate’:

| Code point | Attribute value |
|---|---|
| 0 | continue |
| 1 | stop |

Code points associated with bit ‘beginEnd’ in class ‘attr_beginEnd’:

| Code point | Attribute value |
|---|---|
| 0 | begin |
| 1 | end |

Code points associated with bit field ‘event’ in class ‘attr_custom_event’:

| Code point | Attribute value |
|---|---|
| 0 | abort |
| 1 | accessKey |
| 2 | activate |
| 3 | begin |
| 4 | click |
| 5 | end |
| 6 | error |
| 7 | focusin |
| 8 | focusout |
| 9 | keydown |
| 10 | keyup |
| 11 | load |
| 12 | longAccessKey |
| 13 | mousedown |
| 14 | mouseout |
| 15 | mouseover |
| 16 | mouseup |
| 17 | pause |
| 18 | repeat |
| 19 | resize |
| 20 | resume |
| 21 | scroll |
| 22 | textinput |
| 23 | unload |
| 24 | zoom |
| 25 – 63 | ISO reserved |

Code points associated with bit field ‘color’ in class ‘attr_custom_paint’:

| Code point | Attribute value |
|---|---|
| 0 | inherit |
| 1 | currentColor |
| 2 | none |
| 3 | ISO Reserved |

Code points associated with bit field ‘units’ in class ‘attr_custom_valueWithUnits’:

| Code point | Attribute value |
|---|---|
| 0 | no unit |
| 1 | ‘in’ |
| 2 | ‘cm’ |
| 3 | ‘mm’ |
| 4 | ‘pt’ |
| 5 | ‘pc’ |
| 6 | ‘%’ |
| 7 | ISO reserved |

Code points associated with bit field ‘enum’ in class ‘focus’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | self |

Code points associated with bit field ‘enum’ in class ‘attr_custom_time’:

| Code point | Attribute value |
|---|---|
| 0 | indefinite |
| 1 | media |

Code points associated with bit field ‘color-rendering’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | inherit |
| 2 | optimizeQuality |
| 3 | optimizeSpeed |

Code points associated with bit field ‘display’ in class ‘attr_custom_rare’:

| Code point | Attribute Value |
|---|---|
| 0 | block |
| 1 | compact |
| 2 | inherit |
| 3 | inline |
| 4 | inline-table |
| 5 | list-item |
| 6 | marker |
| 7 | none |
| 8 | run-in |
| 9 | table |
| 10 | table-caption |
| 11 | table-cell |
| 12 | table-column |
| 13 | table-column-group |
| 14 | table-footer-group |
| 15 | table-header-group |
| 16 | table-row |
| 17 | table-row-group |
| 18 – 31 | ISO reserved |

Code points associated with bit field ‘display-align’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | after |
| 1 | before |
| 2 | center |
| 3 | ISO reserved |

Code points associated with bit field ‘fill-rule’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | evenodd |
| 1 | inherit |
| 2 | nonzero |
| 3 | ISO reserved |

Code points associated with bit field ‘image-rendering’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | inherit |
| 2 | optimizeQuality |
| 3 | optimizeSpeed |

Code points associated with bit field ‘line-increment’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | inherit |

Code points associated with bit field ‘pointer-events’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | all |
| 1 | fill |
| 2 | inherit |
| 3 | none |
| 4 | painted |
| 5 | stroke |
| 6 | visible |
| 7 | visibleFill |
| 8 | visiblePainted |
| 9 | visibleStroke |
| 10 – 15 | ISO reserved |

Code points associated with bit field ‘shape-rendering’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | crispEdges |
| 2 | geometricPrecision |
| 3 | inherit |
| 4 | optimizeSpeed |
| 5 – 7 | ISO reserved |

Code points associated with bit field ‘stroke-linecap’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | butt |
| 1 | inherit |
| 2 | round |
| 3 | square |

Code points associated with bit field ‘stroke-linejoin’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | bevel |
| 1 | inherit |
| 2 | miter |
| 3 | round |

Code points associated with bit field ‘textAnchor’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | end |
| 1 | inherit |
| 2 | middle |
| 3 | start |

Code points associated with bit field ‘text-rendering’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | geometricPrecision |
| 2 | inherit |
| 3 | optimizeLegibility |
| 4 | optimizeSpeed |
| 5 – 7 | ISO reserved |

Code points associated with bit field ‘vector-effect’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | default |
| 1 | inherit |
| 2 | non-scaling-stroke |
| 3 – 15 | ISO reserved |

Code points associated with bit field ‘visibility’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | hidden |
| 1 | inherit |
| 2 | visible |
| 3 | ISO Reserved |

Code points associated with bit field ‘feature’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | http://www.w3.org/TR/SVGTiny12/feature#Animation |
| 1 | http://www.w3.org/TR/SVGTiny12/feature#Audio |
| 2 | http://www.w3.org/TR/SVGTiny12/feature#ComposedVideo |
| 3 | http://www.w3.org/TR/SVGTiny12/feature#ConditionalProcessing |
| 4 | http://www.w3.org/TR/SVGTiny12/feature#ConditionalProcessingAttribute |
| 5 | http://www.w3.org/TR/SVGTiny12/feature#CoreAttribute |
| 6 | http://www.w3.org/TR/SVGTiny12/feature#Extensibility |
| 7 | http://www.w3.org/TR/SVGTiny12/feature#ExternalResourcesRequired |
| 8 | http://www.w3.org/TR/SVGTiny12/feature#Font |
| 9 | http://www.w3.org/TR/SVGTiny12/feature#Gradient |
| 10 | http://www.w3.org/TR/SVGTiny12/feature#GraphicsAttribute |
| 11 | http://www.w3.org/TR/SVGTiny12/feature#Handler |
| 12 | http://www.w3.org/TR/SVGTiny12/feature#Hyperlinking |
| 13 | http://www.w3.org/TR/SVGTiny12/feature#Image |
| 14 | http://www.w3.org/TR/SVGTiny12/feature#OpacityAttribute |
| 15 | http://www.w3.org/TR/SVGTiny12/feature#PaintAttribute |
| 16 | http://www.w3.org/TR/SVGTiny12/feature#Prefetch |
| 17 | http://www.w3.org/TR/SVGTiny12/feature#SVG |
| 18 | http://www.w3.org/TR/SVGTiny12/feature#SVG-animation |
| 19 | http://www.w3.org/TR/SVGTiny12/feature#SVG-dynamic |
| 20 | http://www.w3.org/TR/SVGTiny12/feature#SVG-static |
| 21 | http://www.w3.org/TR/SVGTiny12/feature#SVGDOM |

| Code point | Attribute value |
|---|---|
| 22 | http://www.w3.org/TR/SVGTiny12/feature#SVGDOM-animation |
| 23 | http://www.w3.org/TR/SVGTiny12/feature#SVGDOM-dynamic |
| 24 | http://www.w3.org/TR/SVGTiny12/feature#SVGDOM-static |
| 25 | http://www.w3.org/TR/SVGTiny12/feature#Script |
| 26 | http://www.w3.org/TR/SVGTiny12/feature#Shape |
| 27 | http://www.w3.org/TR/SVGTiny12/feature#SolidColor |
| 28 | http://www.w3.org/TR/SVGTiny12/feature#Structure |
| 29 | http://www.w3.org/TR/SVGTiny12/feature#Text |
| 30 | http://www.w3.org/TR/SVGTiny12/feature#TimedAnimation |
| 31 | http://www.w3.org/TR/SVGTiny12/feature#TransformedVideo |
| 32 | http://www.w3.org/TR/SVGTiny12/feature#Video |
| 33 | http://www.w3.org/TR/SVGTiny12/feature#XlinkAttribute |
| 34 – 61 | ISO Reserved |

Code points associated with bit field ‘space’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | default |
| 1 | preserve |

Code points associated with bit field ‘focusable’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | auto |
| 1 | false |
| 2 | true |
| 3 | ISO reserved |

Code points associated with bit field ‘font-style’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | BOLD |
| 1 | BOLDITALIC |
| 2 | EMBOSS |
| 3 | ENGRAVE |
| 4 | ITALIC |
| 5 | LEFTDROPSHADOW |
| 6 | OUTLINE |
| 7 | PLAIN |
| 8 | RIGHTDROPSHADOW |
| 9 | UNDERLINE |
| 10 | inherit |
| 11 | italic |
| 12 | normal |
| 13 | oblique |
| 14 – 31 | ISO reserved |

Code points associated with bit field ‘font-weight’ in class ‘attr_custom_rare’:

| Code point | Attribute value |
|---|---|
| 0 | 100 |
| 1 | 200 |
| 2 | 300 |
| 3 | 400 |
| 4 | 500 |
| 5 | 600 |
| 6 | 700 |
| 7 | 800 |
| 8 | 900 |
| 9 | bold |
| 10 | bolder |
| 11 | inherit |
| 12 | lighter |
| 13 | normal |
| 14 – 15 | 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) ch4;
  switch(ch4){
    case 0:
      privateAnyXMLAttribute  anyXML;
      break;
    case 1:
      privateOpaqueAttribute opaque;
      break;
    case 2:
      attr_any reserved;
      break;
    default:
      custom_extension zzz1;
      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_cursor cursor;
      break;
    case 8:
      element_defs defs;
      break;
    case 9:
      element_desc_metaData_title desc;
      break;
    case 10:
      element_ellipse ellipse;
      break;
    case 11:
      element_foreignObject foreignObject;
      break;
    case 12:
      element_g g;
```

```
      break;
    case 13:
      element_image image;
      break;
    case 14:
      element_line line;
      break;
    case 15:
      element_linearGradient linearGradient;
      break;
    case 16:
      element_desc_metaData_title metadata;
      break;
    case 17:
      element_mpath mpath;
      break;
    case 18:
      element_path path;
      break;
    case 19:
      element_polygon polygon;
      break;
    case 20:
      element_polygon polyline;
      break;
    case 21:
      element_radialGradient radialGradient;
      break;
    case 22:
      element_rect rect;
      break;
    case 23:
      element_sameg sameg;
      break;
    case 24:
      element_sameline sameline;
      break;
    case 25:
      element_samepath samepath;
      break;
    case 26:
      element_samepathfill samepathfill;
      break;
    case 27:
      element_samepolygon samepolygon;
      break;
    case 28:
      element_samepolygonfill samepolygonfill;
      break;
    case 29:
      element_samepolygonstroke samepolygonstroke;
      break;
    case 30:
      element_samepolygon samepolyline;
      break;
    case 31:
      element_samepolygonfill samepolylinefill;
      break;
    case 32:
      element_samepolygonstroke samepolylinestroke;
```

```
      break;
    case 33:
      element_samerect samerect;
      break;
    case 34:
      element_samerectfill samerectfill;
      break;
    case 35:
      element_sametext sametext;
      break;
    case 36:
      element_sametextfill sametextfill;
      break;
    case 37:
      element_sameuse sameuse;
      break;
    case 38:
      element_script script;
      break;
    case 39:
      element_set set;
      break;
    case 40:
      element_stop stop;
      break;
    case 41:
      element_switch switch;
      break;
    case 42:
      element_text text;
      break;
    case 43:
      element_desc_metaData_title title;
      break;
    case 44:
      element_tspan tspan;
      break;
    case 45:
      element_use use;
      break;
    case 46:
      element_video video;
      break;
    case 47:
      element_listener listener;
      break;
    case 48:
      element_any ext;
      break;
    case 49:
      privateElementContainer privateElement;
      break;
    case 50:
      attr_custom_byteAlignedString textContent;
      break;
    default:
      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_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_begin;
if(has_begin) {
  attr_times begin;
}
bit(1) has_dur;
if(has_dur) {
  attr_custom_time dur;
}
bit(1) enabled;
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_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  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_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_begin;
  if(has_begin) {
    attr_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_custom_time dur;
  }
  bit(1) enabled;
  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) 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_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_begin;
if(has_begin) {
  attr_times begin;
}
bit(1) has_dur;
if(has_dur) {
  attr_custom_time dur;
}
bit(1) enabled;
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_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  attr_rotscatra 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_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_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_custom_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_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_custom_time clipBegin;
  }
  bit(1) has_clipEnd;
  if(has_clipEnd) {
    attr_custom_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_cursor {
  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 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_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) has_choice;
  if(has_choice) {
    attr_choice choice;
  }
  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_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_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_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) has_begin;
  if(has_begin) {
    attr_times begin;
  }
  bit(1) enabled;
  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;
  }
  custom_updateList child0;
  bit(1) opt_group;
  if(opt_group) {
    privateAttributeContainer privateAttributes;
  }
}
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:
      custom_extension ext;
      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_groupID 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_cursor cursor;
          break;
        case 8:
          element_defs defs;
          break;
        case 9:
          element_desc_metaData_title desc;
          break;
        case 10:
          element_ellipse ellipse;
          break;
```

```
        case 11:
          element_foreignObject foreignObject;
          break;
        case 12:
          element_g g;
          break;
        case 13:
          element_image image;
          break;
        case 14:
          element_line line;
          break;
        case 15:
          element_linearGradient linearGradient;
          break;
        case 16:
          element_desc_metaData_title metadata;
          break;
        case 17:
          element_mpath mpath;
          break;
        case 18:
          element_path path;
          break;
        case 19:
          element_polygon polygon;
          break;
        case 20:
          element_polygon polyline;
          break;
        case 21:
          element_radialGradient radialGradient;
          break;
        case 22:
          element_rect rect;
          break;
        case 23:
          element_script script;
          break;
        case 24:
          element_set set;
          break;
        case 25:
          element_stop stop;
          break;
        case 26:
          element_svg svg;
          break;
        case 27:
          element_switch switch;
          break;
        case 28:
          element_text text;
          break;
        case 29:
          element_desc_metaData_title title;
          break;
        case 30:
          element_tspan tspan;
          break;
```

```
        case 31:
          element_use use;
          break;
        case 32:
          element_video video;
          break;
        case 33:
          element_listener listener;
          break;
        default:
          break;
        }
      break;
    case 1:
      bit(1) ch6;
      switch(ch6){
        case 0:
          element_any ext;
          break;
        case 1:
          privateElementContainer privateElement;
          break;
        default:
          break;
        }
      break;
    default:
      break;
    }
}
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_begin;
  if(has_begin) {
    attr_times begin;
  }
  bit(1) has_dur;
  if(has_dur) {
    attr_custom_time dur;
  }
  bit(1) enabled;
  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_attributeName;
  if(has_attributeName) {
    attr_AttributeName attributeName;
  }
  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_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_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_custom_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_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;
  }
  bit(1) 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_times begin;
}
bit(1) has_dur;
if(has_dur) {
  attr_custom_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_repeatCount;
if(has_repeatCount) {
  attr_repeatCount repeatCount;
}
bit(1) has_repeatDur;
if(has_repeatDur) {
  attr_repeatDur repeatDur;
}
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_custom_time clipBegin;
  }
  bit(1) has_clipEnd;
  if(has_clipEnd) {
    attr_custom_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_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) enabled;
  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) has_delay;
  if(has_delay) {
```

```
    attr_custom_time delay;
  }
  bit(1) has_timeAttribute;
  if(has_timeAttribute) {
    attr_beginEnd timeAttribute;
  }
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  object_content child0;
}
class privateElementContainer {
  bit(4) ch4;
  switch(ch4){
    case 0:
      privateAnyXMLElement anyXML;
      break;
    case 1:
      privateOpaqueElement opaque;
      break;
    case 2:
      element_any reserved;
      break;
    default:
      custom_extension zzz1;
      break;
    }
}
class update_NewScene {
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  element_svg svg;
}
class update_RefreshScene {
  vluimsbf5 time;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
  element_svg svg;
}
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_groupID groupID;
  bit(1) has_attr_any;
  if(has_attr_any) {
    attr_any any;
  }
}
class update_Save {
  attr_custom_IDREFS elementAttributeList;
  attr_groupID 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;
  }
}
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) {
    custom_extension ext;
  }
}
class LASeRUnitHeader {
  bit(1) resetEncodingContext;
  bit(1) opt_group;
  if(opt_group) {
    custom_extension ext;
  }
}
class initialisations {
  colorInitialisation c;
  fontInitialisation f;
  privateDataIdentifierInitialisation p;
  anyXMLInitialisation a;
  extendedInitialisation e;
}


class focus {
    bit(1) isEnum;
    if (isEnum) {
        //enumeration auto{0} self{1};
        bit(1) enum;
    } else {
        attr_custom_IDREF id;
    }
}

class attr_coordinateList {
  vluimsbf5 len;
  for (int t = 0; t < len; t++) {
    attr_custom_coordinate coord[[t]];
  }
}


class fixed_16_8i {
  bit(1) isInherit;
  if (!isInherit) {
    int(24) float;
  }
}

class attr_point {
  for (int t=0;t<2;t++) {
    attr_custom_coordinate item[[t]];
  }
}

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

//for elements
class element_any {
  uint(extensionIDBits) reserved;
  vluimsbf5 len; //length in bits
```

```
  bit[len] toSkip;
}

//for attributes
class attr_any {
  do {
    uint(extensionIDBits) reserved;
    vluimsbf5 len; //length in bits
    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) append;
  bit(1) hasStringIds;
  bit(1) hasPrivateData;
  bit(1) hasExtendedAttributes;
  // extensionIDBits defines the number of bits of extension tags
  bit(4) extensionIDBits;
  bit(1) hasExtensionConfiguration;
  if (hasExtensionConfiguration) {
    vluimsbf5 len;
    byte[len] extConfiguration; // may be used by a ISO/IEC 23001-1 decoder as specified in clause 14
  }
  bit(1) hasExtension;
  if (hasExtension) {
    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 {
  bit(1) hasExtension;
  if (hasExtension) {
    custom_extension c;
  }
}
```

### 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 occ2;
    for(int t=0;t<occ2;t++) {
      elements child0[[t]];
    }
  }
}

class attr_script {
    bit(1) choice;
    switch(choice) {
      case 0:
        attr_custom_byteAlignedString string;
        break;
      case 1:
          // Enumeration: application/ecmascript{0} application/jar-archive{1} application/laserscript{2}
          bit(2) script;
        break;
      default:
        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_times {
    bit(1) choice;
    switch(choice) {
      case 0:
        vluimsbf5 max;
        for (int t=0;t<max;t++) {
          signedInt item[[t]];
        }
        break;
      case 1:
          // Enumeration: indefinite{0}
        break;
      default:
        break;
      }
}
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 unsignedInt0;
        break;
      case 1:
          // Enumeration: indefinite{0}
        break;
      default:
        break;
      }
}
```

```
class attr_restart {
    // Enumeration: always{0} never{1} whenNotActive{2}
    bit(2) restart;
}
class attr_AttributeName {
    // Enumeration: accumulate{0} additive{1} append{2} attributeName{3} audio-level{4} bandwidth{5} begin{6}
by{7} calcMode{8} children{9} choice{10} color{11} color-rendering{12} cx{13} cy{14} d{15} delay{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} from{34} gradientUnits{35} handler{36} height{37} image-rendering{38} keyCodes{39} keyPoints{40}
keySplines{41} keyTimes{42} line-increment{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-
anchor{102} text-rendering{103} textContent{104} timeAttribute{105} to{106} transform{107} transformBehavior{108}
translation{109} type{110} values{111} vector-effect{112} viewBox{113} viewport-fill{114} viewport-fill-
opacity{115} visibility{116} width{117} x{118} x1{119} x2{120} xlink:actuate{121} xlink:arcrole{122}
xlink:href{123} xlink:role{124} xlink:show{125} xlink:title{126} xlink:type{127} xml:base{128} xml:lang{129}
xml:space{130} y{131} y1{132} y2{133} zoomAndPan{134}
    bit(8) AttributeName;
}
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 unsignedInt0;
        break;
```

```
      case 1:
          // Enumeration: default{0}
        break;
      default:
        break;
      }
}
class attr_choice {
    bit(1) choice;
    switch(choice) {
      case 0:
        bit(8) value;
        break;
      case 1:
          // Enumeration: all{0} clip{1} delta{2} none{3}
          bit(2) choice;
        break;
      default:
        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 objectSame_content {
  bit(1) opt_group;
  if(opt_group) {
    vluimsbf5 occ0;
    for(int t=0;t<occ0;t++) {
      elements child0[[t]];
    }
  }
}
class attr_groupID {
  bit(10) value;
}
class attr_index {
  vluimsbf5 value;
}
class attr_playbackOrder {
    // Enumeration: all{0} forwardOnly{1}
    bit(1) playbackOrder;
}
class attr_preserveAspectRatio {
    bit(1) choice;
    switch(choice) {
      case 0:
          bit(1) choice1;
          switch(choice1) {
            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) preserveAspectRatio1;
              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) preserveAspectRatio2;
            break;
          default:
            break;
          }
        break;
      case 1:
          // Enumeration: reserved{0}
          bit(5) preserveAspectRatio3;
        break;
      default:
        break;
      }
}
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 unsignedInt0;
        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_overflow {
    // Enumeration: visible{0}
    bit(2) overflow;
}
class attr_overlay {
    bit(1) choice;
    switch(choice) {
      case 0:
        custom_extension overlayExType0;
        break;
      case 1:
          // Enumeration: fullscreen{0} none{1} top{2}
          bit(2) 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_beginEnd {
    // Enumeration: begin{0} end{1}
    bit(1) beginEnd;
}
class signedInt {
  bit(1) sign; // 1 is negative
  vluimsbf5 value;
}
```

### 12.2.3 Specific Data Types

#### 12.2.3.1 ID

##### 12.2.3.1.1 Syntax

```
class attr_custom_ID{
    vluimsbf5 ID;
    if (hasStringIds) { // hasStringIds is a LASeRHeader attribute
        attr_custom_byteAlignedString stringId;
    }
    bit(1) reserved;
    if (reserved) {
        vluimsbf5 len;
        bit[len] reserved;
    }
}
```



##### 12.2.3.1.2 Semantics

This class allocates a number for an id. If during the decoding process, the string value of the id is required (`hasStringIds==true`), the string value of this id is encoded. The `hasStringIds` value shall be initialised using the initialisation parameters as defined in subclause 6.6.2.2.

#### 12.2.3.2 IDRef

##### 12.2.3.2.1 Syntax

```
class attr_custom_IDREF{
    vluimsbf5 href;
}
```

##### 12.2.3.2.2 Semantics

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

#### 12.2.3.3 AnyURI

##### 12.2.3.3.1 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;
}
```

##### 12.2.3.3.2 Semantics

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

#### 12.2.3.4 Color

##### 12.2.3.4.1 Syntax

```
class attr_custom_paint {
  bit(1) hasIndex;
  if (hasIndex) {
    uint(colorIndexBits) color0;
  } else {
      bit(1) isEnum;
      if (isEnum) {
        //enumeration inherit{0} currentColor{1} none{2}
        bit(2) color;
      } else {
          bit(1) isURI;
          if (isURI) {
              attr_custom_anyURI uri;
          } else {
            custom_extension colorExType0; // extensibility
        }
      }
  }
}
```

##### 12.2.3.4.2 Semantics

This class allows pointing to a color. The `colorIndexBits` value shall be initialised using the initialisation parameters as defined in subclause 6.6.2.2. The value for each color shall be initialised using the colorInitialisation class in the LASeRUnit, whose syntax is given below in subclause 12.2.

#### 12.2.3.5 Matrix

##### 12.2.3.5.1 Syntax

```
class 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_fixed16_8 valueX;
          attr_custom_fixed16_8 valueY;
        }
      } else {
        custom_extension ext;
      }
    } else {
      bit(1) xx_yy_present;
      if (xx_yy_present) {
          uint(coordBits+scaleBits) xx;
          uint(coordBits+scaleBits) yy;
      }
      bit(1) xy_yx_present;
      if (xy_yx_present) {
          uint(coordBits+scaleBits) xx;
          uint(coordBits+scaleBits) yy;
      }
      bit(1) xz_yz_present;
      if (xz_yz_present) {
          uint(coordBits+scaleBits) xz;
          uint(coordBits+scaleBits) yz;
      }
    }
  }
}
```

##### 12.2.3.5.2 Semantics

This class decodes a matrix value. The `coordBits` and `scaleBits` values shall be initialised using the initialisation parameters as defined in subclause 6.6.2.2.

#### 12.2.3.6 Fraction

##### 12.2.3.6.1 Syntax

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

##### 12.2.3.6.2 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]0.)

#### 12.2.3.7 Path

##### 12.2.3.7.1 Syntax

```
class attr_custom_path {
    attr_custom_pointSequence seq;
    vluimsbf5 nbOfTypes;
    for (int i = 0; i < nbOfTypes; i++) {
        uint(5) type[[i]];
    }
}
```

##### 12.2.3.7.2 Semantics

This class decodes a path value.

#### 12.2.3.8 PointSequence

##### 12.2.3.8.1 Syntax

```
class attr_custom_pointSequence {//see subclause 13.1 for detailed semantics
    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 {
                unit(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 {
        custom_extension ext;
    }
}

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

#### 12.2.3.8.2 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..

#### 12.2.3.8.3 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.

#### 12.2.3.8.4 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.

#### 12.2.3.9 ValueWithUnits

##### 12.2.3.9.1 Syntax

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

##### 12.2.3.9.2 Semantics

This class decodes a value with unit.

#### 12.2.3.10 AnimatedValues

##### 12.2.3.10.1 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_fixed16_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:
                custom_extension privateData;
                break;
        }
    }
  }
}
```

#### 12.2.3.10.2 Semantics

This class decodes a list of animated values.

### 12.2.3.11 AnimatedValue

#### 12.2.3.11.1 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_fixed16_8 value;
            break;
        case 12:
            attr_custom_anyURI value;
            break;
        case 2:
            attr_custom_path value;
```