
**Information technology - Multimedia
application format (MPEG-A) —**

**Part 13:
Augmented reality application format**

*Technologies de l'information - Format des applications
multimedias —*

Partie 13: Format pour les Applications de Réalité Augmentée





COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2017, Published in Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized otherwise in any form or by any means, electronic or mechanical, including photocopying, or posting on the internet or an intranet, without prior written permission. Permission can be requested from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

Contents

Page

Foreword	iv
Introduction	v
1 Scope	1
2 Normative references	1
3 Terms, definitions, and abbreviated terms	1
3.1 Terms and definitions	1
3.2 Abbreviated terms	3
4 ARAF principle and context	3
5 ARAF scene description	5
5.1 General	5
5.1.1 Elementary media	7
5.1.2 Programming information	34
5.1.3 User interactivity	35
5.1.4 Scene related information (spatial and temporal relationships)	43
5.1.5 Dynamic and animated scene	98
5.1.6 Communication and compression	102
5.1.7 Terminal	112
6 ARAF for sensors and actuators	113
6.1 General	113
6.1.1 Usage of InputSensor and script nodes	113
6.2 Access to local camera sensor	116
6.3 Usage of outputactuator and script nodes	117
6.3.1 General	117
7 ARAF compression	120
8 Reference software	121
8.1 General	121
8.2 Implementation details	121
8.3 Utility software	122
9 Conformance	122
Annex A (informative) Map related prototypes implementation	125
Annex B (informative) ARAF support for proprietary formats	143
Annex C (informative) ARAF interactive applications description	144
Bibliography	146

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.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular the different approval criteria needed for the different types of ISO documents should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation on the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT) see the following URL: www.iso.org/iso/foreword.html.

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

This second edition cancels and replaces the first edition (ISO/IEC 23000-13:2014), which has been technically revised.

It also incorporates the Amendment ISO/IEC 23000-13:2014/Amd. 1:2015.

A list of all parts in the ISO/IEC 23000 series can be found on the ISO website.

Introduction

Augmented Reality (AR) applications refer to a view of a real-world environment (RWE), whose elements are augmented by content, such as graphics or sound, in a computer driven process. Augmented Reality Application Format (ARAF) is a collection of a subset of the ISO/IEC 14496-11 Scene Description and Application Engine standard, combined with other relevant MPEG standards (e.g. ISO/IEC 23005, MPEG-V), designed to enable the consumption of 2D/3D multimedia content. Consequently, this document focuses not on client or server procedures, but on the data formats used to provide an augmented reality presentation.

Information technology - Multimedia application format (MPEG-A) —

Part 13: Augmented reality application format

1 Scope

This document specifies the following:

- scene description elements for representing AR content;
- mechanisms to connect to local and remote sensors and actuators;
- mechanisms to integrated compressed media (image, audio, video, graphics);
- mechanisms to connect to remote resources such as maps and compressed media.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements 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 10646-1:2012, *Information technology — Universal multiple-octet coded character set (UCS) — Part 1: Architecture and basic multilingual plane*

ISO/IEC 14496-1:2010 + Amd. 2:2014, *Information technology — Coding of audio-visual objects — Part 1: Systems*

ISO/IEC 14496-3:2009, *Information technology — Coding of audio-visual objects — Part 3: Audio*

ISO/IEC 14496-11:2015, *Information technology — Coding of audio-visual objects — Part 11: Scene description and application engine*

ISO/IEC 14496-16:2011, *Information technology — Coding of audio-visual objects — Part 16: Animation Framework eXtension (AFX)*

ISO/IEC 14772-1:1997, *Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding*

ISO/IEC 23005-5, *Information technology — Media context and control — Part 5: Data formats for interaction devices*

3 Terms, definitions, and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23000-1 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

- ISO Online browsing platform: available at <http://www.iso.org/obp>

— IEC Electropedia: available at <http://www.electropedia.org/>

3.1.1

ARAF browser

augmented reality application format compliant browser

3.1.2

MAR scene

textual result of the MAREC creation, played by an *ARAF browser* ([3.1.1](#))

Note 1 to entry: The result is a MAR experience.

3.1.3

MAR experience

act of playing the ARAF scene using an *ARAF browser* ([3.1.1](#))

Note 1 to entry: The ARAF browser interprets the ARAF scene and presents the result on the end-user's device.

3.1.4

content creator

creator of the media files that are being used within the *MAR experience* ([3.1.3](#))

Note 1 to entry: The media files can be 2D and/or 3D graphics, images, videos and/or sounds.

3.1.5

end-user device

smartphone or mobile device used by an end-user to play a *MAR scene* ([3.1.2](#))

Note 1 to entry: The device shall have an ARAF browser installed.

3.1.6

processing server

server that offers at least one required processing functionality for a *MAR experience* ([3.1.3](#)) and it is capable of communicating with an *ARAF browser* ([3.1.1](#))

3.1.7

target resource

target image or target image descriptor

Note 1 to entry: The target image represents the image that shall be detected and recognized by a recognition library. The target image descriptor is represented by the visual descriptors extracted from a target image. The target resources may be specified by the MAREC or they can be already stored in databases on remote servers.

3.1.8

prerecorded video

prerecorded 2D video whose location is specified by MAREC

Note 1 to entry: The video file can be stored locally (on the device where the MAR experience is played) or remotely (anywhere else on the web). The recognition process shall be performed on the frames (still images) composing the video.

3.1.9

live video camera (stream)

live 2D video camera feed

Note 1 to entry: The URL of the camera providing the real time capture is specified by the MAREC. The URL can point to one of the cameras of the device where the MAR experience is played or to any other camera that can provide a live video stream and the ARAF browser can connect to.

3.1.10**image recognition library**

library that is able to recognize *target resources* ([3.1.7](#)) in a video

Note 1 to entry: The library can run locally (implemented in the ARAF browser) or remotely (on a processing server). The result of an image recognition library is an array of indexes of the recognized target resources.

3.1.11**image recognition and tracking library**

library that is able to recognize and track *target resources* ([3.1.7](#)) in a video

Note 1 to entry: The library can run locally (implemented in the ARAF browser) or remotely (on a processing server). The result of a recognition and tracking library is an array of indexes of the recognized target resources and their pose matrixes. Each recognized target resource shall have a pose matrix associated or a default value if the corresponding pose matrix could not be computed.

3.1.12**augmentation resource**

media objects that are used in the augmentation of the *MAR experience* ([3.1.3](#))

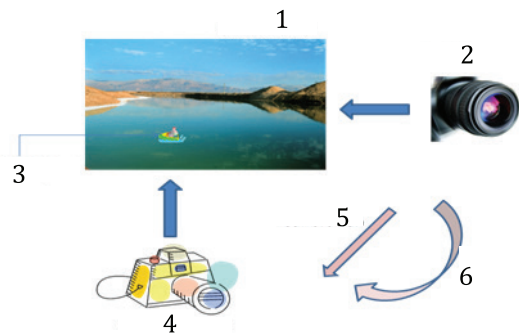
Note 1 to entry: A valid augmentation resource can be a 2D/3D graphic element, an image, a video, a sound or a BIFS scene. The augmentation resources can be stored locally in the MAR Scene or remotely anywhere on the Web, as long as the ARAF browser is capable of accesing their locations. In this case, a URL pointing to the augmentation resource is stored in the MAR scene.

3.2 Abbreviated terms

AR	Augmented Reality
ARAF	Augmented Reality Application Format
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
MAR	Mixed and Augmented Reality
MARE	Mixed and Augmented Reality Experience
MAREC	Mixed and Augmented Reality Experience Creator
PROTO	A PROTOtype is a mechanism used to group together scene graph elements in order to implement one or several specific functionalities.
RTR	Recognized Target Resource

4 ARAF principle and context

Augmented Reality (AR) applications refer to a view of a real-world environment whose elements are augmented by content, such as graphics or sound, in a computer driven process. [Figure 1](#) illustrates two real and virtual cameras and the composition of a real image and graphics objects. [Annex C](#) describes several application scenarios for augmented reality.



- Key**
- 1 real picture
 - 2 real camera
 - 3 graphic object
 - 4 virtual camera
 - 5 calibration
 - 6 position and orientation

Figure 1 — Simplified illustration of the AR principle

The Augmented Reality Application Format (ARAF) is an extension of a subset of the MPEG-4 part 11 Scene Description and Application Engine standard, combined with other relevant MPEG standards (MPEG-4, MPEG-V), designed to enable the consumption of 2D/3D multimedia content as depicted in [Figure 2](#).

An ARAF, available as a file or stream, is interpreted by a device, called **ARAF device**. The nodes of the ARAF scene point to different sources of multimedia content such as 2D/3D image, 2D/3D audio, 2D/3D video, 2D/3D graphics and sensor/sensory information sources/sinks that are either remote or/and local.

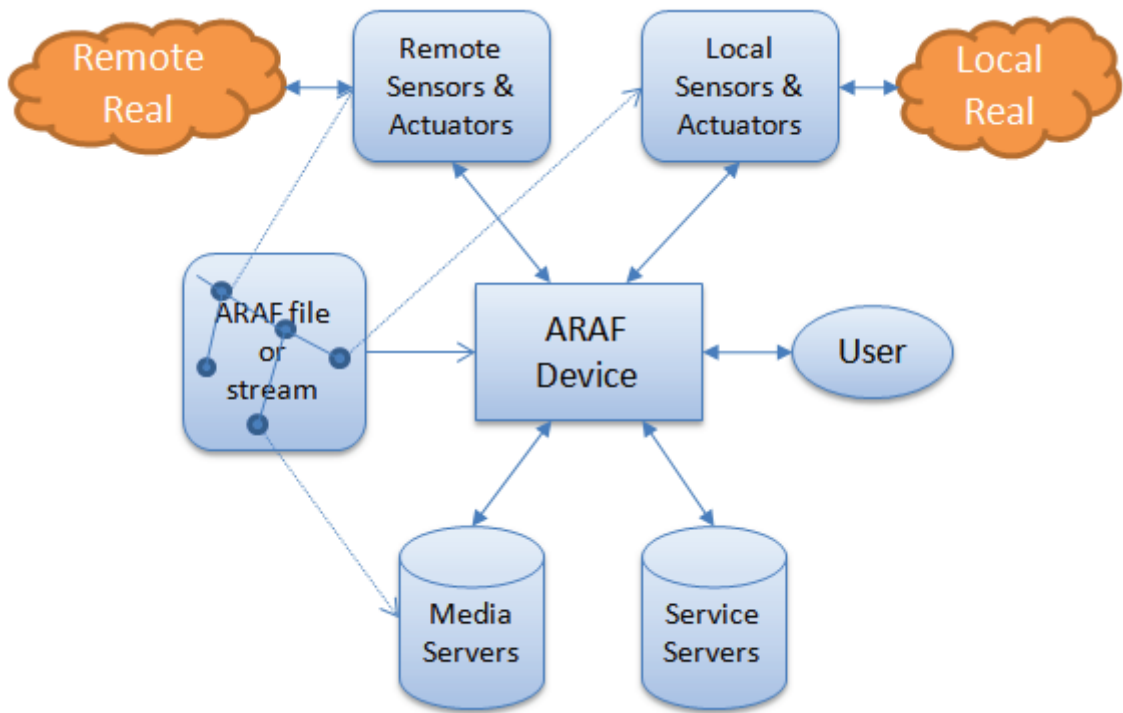


Figure 2 — The ARAF context

5 ARAF scene description

5.1 General

To describe the multimedia scene, ARAF is based on ISO/IEC 14496-11 which at its turn is based on ISO/IEC 14772-1 (VRML97). About two hundred nodes are standardized in MPEG-4 BIFS and VRML, allowing various kinds of scenes to be constructed. ARAF is referring to a subset of MPEG-4 BIFS nodes and external prototypes as defined in ISO/IEC 14496-11:2015, Annex E for scene description as presented in [Table 1](#).

Table 1 — ARAF nodes and prototypes

Category	Sub-category	Node, prototypes/elements name in MPEG-4 BIFS/XMT	Type
Elementary media	Audio	AudioSource	Node
		Sound	Node
		Sound2D	Node
	Image and video	ImageTexture	Node
		MovieTexture	Node
	Textual information	FontStyle	Node
		Text	Node
	Graphics	Appearance	Node
		Color	Node
		LineProperties	Node
		LinearGradient	Node
		Material	Node
		Material2D	Node
		Rectangle	Node
		Shape	Node
		SBVCAAnimationV2	Node
		SBBone	Node
		SBSegment	Node
		SBSite	Node
		SBSkinnedModel	Node
		MorphShape	Node
		Coordinate	Node
		TextureCoordinate	Node
		Normal	Node
		IndexedFaceSet	Node
		IndexedLineSet	Node
Programming information		Script	Node

Table 1 (continued)

Category	Sub-category	Node, prototypes/elements name in MPEG-4 BIFS/XMT	Type
User interactivity		InputSensor	Node
		OutputActuator	Node
		SphereSensor	Node
		TimeSensor	Node
		TouchSensor	Node
		MediaSensor	Node
		PlaneSensor	Node
		Background	Node
Scene related) information (spatial and temporal relationships		Background2D	Node
		CameraCalibration	PROTO
		Group	Node
		Inline	Node
		Layer2D	Node
		Layer3D	Node
		Layout	Node
		NavigationInfo	Node
		OrderedGroup	Node
		LocImg	PROTO
		RemImgProxy	PROTO
		RemImgServer	PROTO
		RemImgComp	PROTO
		LocAud	PROTO
		RemAud	PROTO
		Switch	Node
		Transform	Node
		Transform2D	Node
		Viewpoint	Node
		Viewport	Node
		Form	Node
Dynamic and animated scene		OrientationInterpolator	Node
		ScalarInterpolator	Node
		CoordinateInterpolator	Node
		ColorInterpolator	Node
		PositionInterpolator	Node
		Valuator	Node
Communication and compression	Maps	BitWrapper	Node
		MediaControl	Node
		Map	PROTO
		MapOverlay	PROTO
		MapMarker	PROTO
		MapPlayer	PROTO
Terminal		TermCap	Node

All the above listed elements are specified in MPEG-4 Part 11. However, to facilitate the implementation of ARAF content, the current document contains their XML syntax as well as the semantics and functionality.

MPEG-4 Part 11 describes a scene with a hierarchical structure that can be represented as a graph. Nodes of the graph build up various types of objects, such as audio video, image, graphic, text, etc. Furthermore, to ensure the flexibility, a new, user-defined type of node derived from a parent one can also be defined on demand by using the *Proto* method.

In general, nodes expose a set of parameters, through which aspects of their appearance and behavior can be controlled. By setting these values, scene designers have a tool to force a scene-reconstruction at clients' terminals to adhere to their intention in a predefined manner. In more complicated scenario, the structure of BIFS nodes is not necessarily static; nodes can be added or removed from the scene graph arbitrarily.

Certain types of nodes called *sensors*, such as TimeSensor, TouchSensor, can interact with users and generate appropriate triggers, which are transmitted to others nodes by routing mechanism, causing changes in state of these receiving nodes. They are bases for the dynamic behavior of a multimedia content supported by MPEG-4.

The maximum flexibility in the programmable feature of MPEG-4 scene is carried out with the *Script* node. By routing mechanism to Event In *valueIn* attribute of Script node, the associated function (defined in its URL attribute) with the same name Event In *valueIn* () will be triggered. The behavior of this function is user-defined, i.e. scene-designer can freely process some computations, and then sets the values for every Event Out *valueOut* attribute, which consecutively affect the states of other nodes linked to them.

Direct manipulation of nodes' states is also possible in MPEG-4 Part 11: the Field *field* attribute can refer to any node in the scene; through this link, all attributes of the contacted node will be exposed to direct setting and modifying operators within the *Script* node. The syntax of the language used to implement the function of Script node is ECMAScript (see ISO/IEC 16262).

ARAF supports the definition and reusability of complex objects by using the MPEG-4 PROTO mechanism. The PROTO statement creates its own nodes by defining a configurable object prototype; it can integrate any other node from the scene graph.

ARAF makes extensive use of EXTERNPROTO mechanism which are nodes whose syntax is identified by URNs as given in ISO/IEC 14496-11:2015, Annex E, while their names are only informative and for convenience can be changed by the content creator in the declaration step.

[Table 1](#) indicates the MPEG-4 Part 11 nodes that are included in ARAF. For each node, it is specified in the version of this document when it was published. Further, the XML syntax as well as the semantics and functionality of these elements are described.

5.1.1 Elementary media

5.1.1.1 Audio

The following audio related nodes are used in ARAF: AudioSource, Sound, Sound2D.

5.1.1.1.1 AudioSource

5.1.1.1.1.1 XSD description

```
<complexType name="AudioSourceType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="children" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFAudioNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
</complexType>
```

```

    </element>
  </all>
  <attribute name="url" type="xmta:MfUrl" use="optional"/>
  <attribute name="pitch" type="xmta:SFFloat" use="optional" default="1"/>
  <attribute name="speed" type="xmta:SFFloat" use="optional" default="1"/>
  <attribute name="startTime" type="xmta:SFTIME" use="optional" default="0"/>
  <attribute name="stopTime" type="xmta:SFTIME" use="optional" default="0"/>
  <attribute name="numChan" type="xmta:SFIInt32" use="optional" default="1"/>
  <attribute name="phaseGroup" type="xmta:MFIInt32" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="AudioSource" type="xmta:AudioSourceType"/>

```

5.1.1.1.1.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.15.

This node is used to add sound to a BIFS scene. See ISO/IEC 14496-3 for information on the various audio tools available for coding sound.

The addChildren eventIn specifies a list of nodes that shall be added to the children field. The removeChildren eventIn specifies a list of nodes that shall be removed from the children field.

The children field allows buffered AudioBuffer or AdvancedAudioBuffer data to be used as sound samples within a structured audio decoding process. Only AudioBuffer and AdvancedAudioBuffer nodes shall be children to an AudioSource node, and only in the case where url indicates a structured audio bitstream. The pitch field controls the playback pitch for the structured audio, the parametric speech (HVXC) and the parametric audio (HILN) decoder. It is specified as a ratio, where 1 indicates the original bitstream pitch, values other than 1 indicate pitch-shifting by the given ratio. This field is available through the getttune() core opcode in the structured audio decoder (see ISO/IEC 14496-3:2009, Clause 5). To adjust the pitch of other decoder types, use the AudioFX node with an appropriate effects orchestra.

The speed field controls the playback speed for the structured audio decoder (see ISO/IEC 14496-3:2009, Clause 5), the parametric speech (HVXC) and the parametric audio (HILN) decoder. It is specified as a ratio, where 1 indicates the original speed; values other than 1 indicate multiplicative time-scaling by the given ratio (i.e. 0,5 specifies twice as fast). The value of this field shall be made available to the structured audio decoder indicated by the url field. ISO/IEC 14496-3:2009, 5.7.3.3.6, list item 8 describes the use of this field to control the structured audio decoder. To adjust the speed of other decoder types, use the AudioFX node with an appropriate effects orchestra (see ISO/IEC 14496-3:2009, 5.9.14.4).

The startTime and stopTime exposedFields and their effects on the AudioSource node are described in ISO/IEC 14496-11:2015, 7.1.1.1.6.2. The numChan field describes how many channels of audio are in the decoded bitstream.

5.1.1.1.2 Sound

5.1.1.1.2.1 XSD description

```

<complexType name="SoundType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="source" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFAudioNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="direction" type="xmta:SFFloat" use="optional" default="0 0 1"/>
  <attribute name="intensity" type="xmta:SFFloat" use="optional" default="1"/>
  <attribute name="location" type="xmta:SFFloat" use="optional" default="0 0 0"/>
  <attribute name="maxBack" type="xmta:SFFloat" use="optional" default="10"/>
  <attribute name="maxFront" type="xmta:SFFloat" use="optional" default="10"/>
  <attribute name="minBack" type="xmta:SFFloat" use="optional" default="1"/>

```

```

<attribute name="minFront" type="xmta:SFFloat" use="optional" default="1"/>
<attribute name="priority" type="xmta:SFFloat" use="optional" default="0"/>
<attribute name="spatialize" type="xmta:SFBool" use="optional" default="true"/>
<attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Sound" type="xmta:SoundType"/>

```

5.1.1.1.2.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.116.

The Sound node is used to attach sound to a scene, thereby giving it spatial qualities and relating it to the visual content of the scene. The Sound node relates an audio BIFS sub-graph to the rest of an audio-visual scene. By using this node, sound may be attached to a group, and spatialized or moved around as appropriate for the spatial transforms above the node. By using the functionality of the audio BIFS nodes, sounds in an audio scene described using ISO/IEC 14496-11 may be filtered and mixed before being spatially composited into the scene. The semantics of this node are as defined in ISO/IEC 14472-1:1997, 6.42, with the following exceptions and additions.

The source field allows the connection of an audio sub-graph containing the sound. The spatialize field determines whether the Sound shall be spatialized. If this flag is set, the sound shall be presented spatially according to the local coordinate system and current listeningPoint, so that it apparently comes from a source located at the location point, facing in the direction given by direction. The exact manner of spatialization is implementation-dependant, but implementators are encouraged to provide the maximum sophistication possible depending on terminal resources. If there are multiple channels of sound output from the child sound, they may or may not be spatialized, according to the phaseGroup properties of the child, as follows. Any individual channels, that is, channels not phase-related to other channels, are summed linearly and then spatialized. Any phase-grouped channels are not spatialized, but passed through this node unchanged. The sound presented in the scene is thus a single spatialized sound, represented by the sum of the individual channels, plus an “ambient” sound represented by mapping all the remaining channels into the presentation system as described in ISO/IEC 14496-11:2015, 7.1.1.2.13.2.2. If the spatialize field is not set, the audio channels from the child are passed through unchanged, and the sound presented in the scene due to this node is an “ambient” sound represented by mapping all the audio channels output by the child into the presentation system as described in ISO/IEC 14496-11:2015, 7.1.1.2.13.2.2.

As with the visual objects in the scene, the Sound node may be included as a child or descendant of any of the grouping or transform nodes. For each of these nodes, the sound semantics are as follows. Affine transformations presented in the grouping and transform nodes affect the apparent spatialization position of spatialized sound. They have no effect on “ambient” sounds. If a particular grouping or transform node has multiple Sound nodes as descendants, then they are combined for presentation as follows. Each of the Sound nodes may be producing a spatialized sound, a multichannel ambient sound, or both. For all of the spatialized sounds in descendant nodes, the sounds are linearly combined through simple summation from presentation. For multichannel ambient sounds, the sounds are linearly combined channel-by-channel for presentation.

5.1.1.1.3 Sound2D

5.1.1.1.3.1 XSD description

```

<complexType name="Sound2DType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="source" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFAudioNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="intensity" type="xmta:SFFloat" use="optional" default="1"/>
  <attribute name="location" type="xmta:SFFVec2f" use="optional" default="0 0"/>
  <attribute name="spatialize" type="xmta:SFBool" use="optional" default="true"/>
  <attributeGroup ref="xmta:DefUseGroup"/>

```

```
</complexType>
<element name="Sound2D" type="xmta:Sound2DType"/>
```

5.1.1.1.3.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.117.

The Sound2D node relates an audio BIFS sub-graph to the other parts of a 2D audio-visual scene. It shall not be used in 3D contexts. By using this node, sound may be attached to a group of visual nodes. By using the functionality of the audio BIFS nodes, sounds in an audio scene may be filtered and mixed before being spatially composed into the scene.

The intensity field adjusts the loudness of the sound. Its value ranges from 0.0 to 1.0, and this value specifies a factor that is used during the playback of the sound. The location field specifies the location of the sound in the 2D scene. The source field connects the audio source to the Sound2D node. The spatialize field specifies whether the sound shall be spatialized on the 2D screen. If this flag is set, the sound shall be spatialized with the maximum sophistication possible. The 2D sound is spatialized assuming a distance of 1 m between the user and a 2D scene of size 2 m × 1,5 m, giving the minimum and maximum azimuth angles of -45° and +45°, and the minimum and maximum elevation angles of -37° and +37°. The same rules for multichannel audio spatialization apply to the Sound2D node as to the Sound (3D) node. Using the phaseGroup flag in the AudioSource node, it is possible to determine whether the channels of the source sound contain important phase relations, and that spatialization at the terminal should not be performed.

As with the visual objects in the scene (and for the Sound node), the Sound2D node may be included as a child or descendant of any of the grouping or transform nodes. For each of these nodes, the sound semantics are as follows. Affine transformations presented in the grouping and transform nodes affect the apparent spatialization position of spatialized sound.

If a transform node has multiple Sound2D nodes as descendants, then they are combined for presentation. If Sound and Sound2D nodes are both used in a scene, all shall be treated the same way according to these semantics.

5.1.1.2 Image and video

The following image and video related nodes are used in ARAF: ImageTexture, MovieTexture.

5.1.1.2.1 ImageTexture

5.1.1.2.1.1 XSD description

```
<complexType name="ImageTextureType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="url" type="xmta:MfUrl" use="optional"/>
  <attribute name="repeatS" type="xmta:SfBool" use="optional" default="true"/>
  <attribute name="repeatT" type="xmta:SfBool" use="optional" default="true"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="ImageTexture" type="xmta:ImageTextureType"/>
```

5.1.1.2.1.2 Functionality and semantics

As defined in ISO/IEC 14772-1:1997, 6.22.

The ImageTexture node defines a texture map by specifying an image file and general parameters for mapping to geometry. Texture maps are defined in a 2D coordinate system (s, t) that ranges from [0.0, 1.0] in both directions. The bottom edge of the image corresponds to the S-axis of the texture map, and left edge of the image corresponds to the T-axis of the texture map. The lower-left pixel of the image corresponds to s=0, t=0, and the top-right pixel of the image corresponds to s=1, t=1.

The texture is read from the URL specified by the url field. When the url field contains no values ([]), texturing is disabled. Browsers shall support the JPEG and PNG image file formats. In addition, browsers may support other image formats (e.g. CGM) which can be rendered into a 2D image. Support for the GIF format is also recommended (including transparency).

The repeatS and repeatT fields specify how the texture wraps in the S and T directions. If repeatS is TRUE (the default), the texture map is repeated outside the [0.0, 1.0] texture coordinate range in the S direction so that it fills the shape. If repeatS is FALSE, the texture coordinates are clamped in the S direction to lie within the [0.0, 1.0] range. The repeatT field is analogous to the repeatS field.

5.1.1.2.2 MovieTexture

5.1.1.2.2.1 XSD description

```
<complexType name="MovieTextureType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="loop" type="xmta:SFFloat" use="optional" default="false"/>
  <attribute name="speed" type="xmta:SFFloat" use="optional" default="1"/>
  <attribute name="startTime" type="xmta:SFFloat" use="optional" default="0"/>
  <attribute name="stopTime" type="xmta:SFFloat" use="optional" default="0"/>
  <attribute name="url" type="xmta:MFUrl" use="optional"/>
  <attribute name="repeatS" type="xmta:SFFloat" use="optional" default="true"/>
  <attribute name="repeatT" type="xmta:SFFloat" use="optional" default="true"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="MovieTexture" type="xmta:MovieTextureType"/>
```

5.1.1.2.2.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.86.

The loop, startTime, and stopTime exposedFields and the isActive eventOut, and their effects on the MovieTexture node, are described in ISO/IEC 14496-11:2015, 7.1.1.1.6.2. The speed exposedField controls playback speed. It does not affect the delivery of the stream attached to the MovieTexture node. For streaming media, value of speed other than 1 shall be ignored.

A MovieTexture shall display frame or VOP 0 if speed is 0. For positive values of speed, the frame or VOP that an active MovieTexture will display at time now corresponds to the frame or VOP at movie time (i.e. in the movie's local time base with frame or VOP 0 at time 0, at speed = 1): $\text{fmod}(\text{now} - \text{startTime}, \text{duration}/\text{speed})$. If speed is negative, then the frame or VOP to display is the frame or VOP at movie time: $\text{duration} + \text{fmod}(\text{now} - \text{startTime}, \text{duration}/\text{speed})$. A MovieTexture node is inactive before startTime is reached. If speed is non-negative, then the first VOP shall be used as texture, if it is already available. If speed is negative, then the last VOP shall be used as texture, if it is already available.

When a MovieTexture becomes inactive, the VOP corresponding to the time at which the MovieTexture became inactive shall persist as the texture. The speed exposedField indicates how fast the movie shall be played. A speed of 2 indicates the movie plays twice as fast. Note that the duration_changed eventOut is not affected by the speed exposedField. set_speed events shall be ignored while the movie is playing.

5.1.1.3 Textual information

The following text related nodes are used in ARAF: FontStyle, Text.

5.1.1.3.1 FontStyle

5.1.1.3.1.1 XSD description

```
<complexType name="FontStyleType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
</complexType>
```

```

    </all>
    <attribute name="family" type="xmta:MFString" use="optional"
default=""SERIF""/>
    <attribute name="horizontal" type="xmta:SFBBool" use="optional" default="true"/>
    <attribute name="justify" type="xmta:MFString" use="optional"
default=""BEGIN""/>
    <attribute name="language" type="xmta:SFBString" use="optional"
default=""""/>
    <attribute name="leftToRight" type="xmta:SFBBool" use="optional" default="true"/>
    <attribute name="size" type="xmta:SFFloat" use="optional" default="1"/>
    <attribute name="spacing" type="xmta:SFFloat" use="optional" default="1"/>
    <attribute name="style" type="xmta:SFBString" use="optional"
default=""PLAIN""/>
    <attribute name="topToBottom" type="xmta:SFBBool" use="optional" default="true"/>
    <attributeGroup ref="xmta:DefUseGroup"/>
  </complexType>
<element name="FontStyle" type="xmta:FontStyleType"/>

```

5.1.1.3.1.2 Functionality and semantics

As defined in ISO/IEC 14772-1:1997, 6.20.

The FontStyle node defines the size, family, and style used for Text nodes, as well as the direction of the text strings and any language-specific rendering techniques used for non-English text.

The size field specifies the nominal height, in the local coordinate system of the Text node, of glyphs rendered and determines the spacing of adjacent lines of text. Values of the size field shall be greater than zero.

The spacing field determines the line spacing between adjacent lines of text. The distance between the baseline of each line of text is (spacing × size) in the appropriate direction (depending on other fields described below). Values of the spacing field shall be non-negative.

Font attributes are defined with the family and style fields. The browser shall map the specified font attributes to an appropriate available font as described below.

The family field contains a case-sensitive MFString value that specifies a sequence of font family names in preference order. The browser shall search the MFString value for the first font family name matching a supported font family. If none of the string values matches a supported font family, the default font family "SERIF" shall be used. All browsers shall support at least "SERIF" (the default) for a serif font such as Times Roman; "SANS" for a sans-serif font such as Helvetica; and "TYPEWRITER" for a fixed-pitch font such as Courier. An empty family value is identical to ["SERIF"].

The style field specifies a case-sensitive SFBString value that may be "PLAIN" (the default) for default plain type; "BOLD" for boldface type; "ITALIC" for italic type; or "BOLDITALIC" for bold and italic type. An empty style value ("") is identical to "PLAIN".

The horizontal, leftToRight, and topToBottom fields indicate the direction of the text. The horizontal field indicates whether the text advances horizontally in its major direction (horizontal = TRUE, the default) or vertically in its major direction (horizontal = FALSE). The leftToRight and topToBottom fields indicate direction of text advance in the major (characters within a single string) and minor (successive strings) axes of layout. Which field is used for the major direction and which is used for the minor direction is determined by the horizontal field.

For horizontal text (horizontal = TRUE), characters on each line of text advance in the positive X direction if leftToRight is TRUE or in the negative X direction if leftToRight is FALSE. Characters are advanced according to their natural advance width. Each line of characters is advanced in the negative Y direction if topToBottom is TRUE or in the positive Y direction if topToBottom is FALSE. Lines are advanced by the amount of size × spacing.

For vertical text (horizontal = FALSE), characters on each line of text advance in the negativeY direction if topToBottom is TRUE or in the positive Y direction if topToBottom is FALSE. Characters are advanced according to their natural advance height. Each line of characters is advanced in the positive X direction

if leftToRight is TRUE or in the negative X direction if leftToRight is FALSE. Lines are advanced by the amount of size × spacing.

The justify field determines alignment of the above text layout relative to the origin of the object coordinate system. The justify field is an MFString which can contain 2 values. The first value specifies alignment along the major axis and the second value specifies alignment along the minor axis, as determined by the horizontal field. An empty justify value ("") is equivalent to the default value. If the second string, minor alignment, is not specified, minor alignment defaults to the value "FIRST". Thus, justify values of "", "BEGIN", and ["BEGIN" "FIRST"] are equivalent.

The major alignment is along the X-axis when horizontal is TRUE and along the Y-axis when horizontal is FALSE. The minor alignment is along the Y-axis when horizontal is TRUE and along the X-axis when horizontal is FALSE. The possible values for each enumerant of the justify field are "FIRST", "BEGIN", "MIDDLE", and "END". For major alignment, each line of text is positioned individually according to the major alignment enumerant. For minor alignment, the block of text representing all lines together is positioned according to the minor alignment enumerant.

The language field specifies the context of the language for the text string. Due to the multilingual nature of the ISO/IEC 10646-1, the language field is needed to provide a proper language attribute of the text string. The format is based on RFC 1766: language[_territory].

The value for the language tag is based on ISO 639 (e.g. "zh" for Chinese, "jp" for Japanese, and "sc" for Swedish.) The territory tag is based on ISO 3166 country codes (e.g. "TW" for Taiwan and "CN" for China for the "zh" Chinese language tag). If the language field is empty (""), local language bindings are used.

The semantics of the FontStyle node are the ones specified above (ISO/IEC 14772-1:1997, 6.20), with the exception that the field types are exposedField and the semantics of the size and spacing fields are as follows.

The size field defines the size of the EM box of a font (The EM is a relative measure of the height of the glyphs in a font defined in a device- and resolution-independent font design units). This value corresponds to the distance between two adjacent baselines of unadjusted text, set in a particular font. The value of the size field is conveyed using the same metric units that are used for a scene description. If a scene uses pixel-based metrics, the value of the size field is specified in pixels, otherwise, it specifies the size in meters.

The spacing field defines the distance between two adjacent lines of text as the product of size and spacing. Special fonts provided in a font data stream can be accessed using the following syntax:

"OD:<odid>;FSID:<fsid>", where:

- <odid> is the numeric value of the objectDescriptorID of the associated font data stream,
- <fsid> is the numeric value of the requested font subset as conveyed by fontSubsetID within the associated font data stream.

5.1.1.3.2 Text

5.1.1.3.2.1 XSD description

```
<complexType name="TextType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="fontStyle" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFFontStyleNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="string" type="xmta:MFString" use="optional"/>
  <attribute name="length" type="xmta:MFFloat" use="optional"/>
  <attribute name="maxExtent" type="xmta:SFFloat" use="optional" default="0"/>
```

```
<attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Text" type="xmta:TextType"/>
```

5.1.1.3.2.2 Functionality and semantics

As defined in ISO/IEC 14772-1:1997, 6.47.

The Text node specifies a two-sided, flat text string object positioned in the Z=0 plane of the local coordinate system based on values defined in the fontStyle field (see ISO/IEC 14772-1:1997, 6.20, FontStyle). Text nodes may contain multiple text strings specified using the UTF-8 encoding as specified by ISO 10646-1 (see 2.[UTF8]). The text strings are stored in the order in which the text mode characters are to be produced as defined by the parameters in the FontStyle node.

The text strings are contained in the string field. The fontStyle field contains one FontStyle node that specifies the font size, font family and style, direction of the text strings, and any specific language rendering techniques used for the text.

The maxExtent field limits and compresses all of the text strings if the length of the maximum string is longer than the maximum extent, as measured in the local coordinate system. If the text string with the maximum length is shorter than the maxExtent, then there is no compressing. The maximum extent is measured horizontally for horizontal text (FontStyle node: horizontal=TRUE) and vertically for vertical text (FontStyle node: horizontal=FALSE). The maxExtent field shall be greater than or equal to zero.

The length field contains an MFFloat value that specifies the length of each text string in the local coordinate system. If the string is too short, it is stretched (either by scaling the text or by adding space between the characters). If the string is too long, it is compressed (either by scaling the text or by subtracting space between the characters). If a length value is missing (for example, if there are four strings but only three length values), the missing values are considered to be 0. The length field shall be greater than or equal to zero.

Specifying a value of 0 for both the maxExtent and length fields indicates that the string may be any length.

— ISO 10646-1 Character encodings

Characters in ISO 10646 (see 2.[UTF8]) are encoded in multiple octets. Code space is divided into four units, as follows:

```
+-----+-----+-----+-----+
| Group-octet | Plane-octet | Row-octet | Cell-octet |
+-----+-----+-----+-----+
```

ISO 10646-1 allows two basic forms for characters:

- a) UCS-2 (Universal Coded Character Set-2). This form is also known as the Basic Multilingual Plane (BMP). Characters are encoded in the lower two octets (row and cell).
- b) UCS-4 (Universal Coded Character Set-4). Characters are encoded in the full four octets.

In addition, three transformation formats (UCS Transformation Format or UTF) are accepted: UTF-7, UTF-8, and UTF-16. Each represents the nature of the transformation: 7-bit, 8-bit, or 16-bit. UTF-7 and UTF-16 are referenced in 2.[UTF8].

UTF-8 maintains transparency for all ASCII code values (0...127). It allows ASCII text (0x0..0x7F) to appear without any changes and encodes all characters from 0x80..0x7FFFFFFF into a series of six or fewer bytes.

If the most significant bit of the first character is 0, the remaining seven bits are interpreted as an ASCII character. Otherwise, the number of leading 1 bits indicates the number of bytes following. There is always a zero bit between the count bits and any data.

The first byte is one of the following. The X indicates bits available to encode the character:

0XXXXXXX only one byte	0..0x7F (ASCII)
110XXXXX two bytes	Maximum character value is 0x7FF

1110XXXX	three bytes	Maximum character value is 0xFFFF
11110XXX	four bytes	Maximum character value is 0x1FFFFFF
111110XX	five bytes	Maximum character value is 0x3FFFFFFF
1111110X	six bytes	Maximum character value is 0x7FFFFFFF

All following bytes have the format 10XXXXXX.

As a two byte example, the symbol for a register trade mark is ® or 174 in ISO Latin-1 (see ISO/IEC 8859-1). It is encoded as 0x00AE in UCS-2 of ISO 10646. In UTF-8, it has the following two byte encoding: 0xC2, 0xAE.

— Appearance

Textures are applied to text as follows. The texture origin is at the origin of the first string, as determined by the justification. The texture is scaled equally in both S and T dimensions, with the font height representing 1 unit. S increases to the right, and T increases up.

ISO/IEC 14772-1:1997, 4.14 has details on VRML lighting equations and how Appearance, Material and textures interact with lighting.

The Text node does not participate in collision detection.

5.1.1.4 Graphics

The following graphics related nodes are used in ARAF: Appearance, Color, LineProperties, LinearGradient, Material, Material2D, Rectangle, Shape, SBVCAAnimationV2, SBBone, SBSegment, SBSite, SBSkinnedModel, MorphShape, Coordinate, TextureCoordinate, Normal, IndexedFaceSet, IndexedLineSet.

5.1.1.4.1 Appearance

5.1.1.4.1.1 XSD Description

```
<complexType name="AppearanceType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="material" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFMaterialNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="texture" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFTextureNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="textureTransform" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFTextureTransformNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Appearance" type="xmta:AppearanceType"/>
```

5.1.1.4.1.2 Functionality and semantics

As defined in ISO/IEC 14772-1:1997, 6.3.

The Appearance node specifies the visual properties of geometry. The value for each of the fields in this node may be NULL. However, if the field is non-NULL, it shall contain one node of the appropriate type.

The material field, if specified, shall contain a Material node. If the material field is NULL or unspecified, lighting is off (all lights are ignored during rendering of the object that references this Appearance) and the unlit object colour is (1, 1, 1). Details of the VRML lighting model are in ISO/IEC 14772-1:1997, 4.14.

The texture field, if specified, shall contain one of the various types of texture nodes (ImageTexture, MovieTexture, or PixelTexture). If the texture node is NULL or the texture field is unspecified, the object that references this Appearance is not textured.

The textureTransform field, if specified, shall contain a TextureTransform node. If the textureTransform is NULL or unspecified, the textureTransform field has no effect.

Additional specification: ISO/IEC 14496-11:2015, 7.2.2.6.2.

The material field, if non-NULL, shall contain either a Material node or a Material2D node depending on the type of geometry node used in the geometry field of the Shape node that contains the Appearance node. The list below shows the geometry nodes that require a Material node, those that require a Material2D node and those where either may apply:

- Material2D only: Circle, Curve2D, IndexedFaceSet2D, IndexedLineSet2D, PointSet2D, Rectangle;
- Material only: Box, Cone, Cylinder, ElevationGrid, Extrusion, IndexedFaceSet, IndexedLineSet, PointSet, Sphere;
- Material2D or Material: Bitmap, Text.

Inside a Shape node in a 2D context, if no Appearance and therefore no Material2D is defined, the default values and behavior of the Material2D node shall be used. In a 3D context, the default behavior is specified in ISO/IEC 14772-1 (the object is unlit and has color 1 1 1).

5.1.1.4.2 Color

5.1.1.4.2.1 XSD description

```
<complexType name="ColorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="color" type="xmta:MFCColor" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Color" type="xmta:ColorType"/>
```

5.1.1.4.2.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.9.

This node defines a set of RGB colours to be used in the fields of another node.

Color nodes are only used to specify multiple colours for a single geometric shape, such as colours for the faces or vertices of an IndexedFaceSet. A Material node is used to specify the overall material parameters of lit geometry. If both a Material node and a Color node are specified for a geometric shape, the colours shall replace the diffuse component of the material.

RGB or RGBA textures take precedence over colours; specifying both an RGB or RGBA texture and a Color node for geometric shape will result in the Color node being ignored.

5.1.1.4.3 LineProperties

5.1.1.4.3.1 XSD description

```
<complexType name="LinePropertiesType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="lineColor" type="xmta:SFCColor" use="optional" default="0 0 0"/>
  <attribute name="lineStyle" type="xmta:SFloat32" use="optional" default="0"/>
  <attribute name="width" type="xmta:SFloat" use="optional" default="1"/>
</complexType>
```

```

    <attributeGroup ref="xmta:DefUseGroup"/>
  </complexType>
  <element name="LineProperties" type="xmta:LinePropertiesType"/>

```

5.1.1.4.3.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.75.2.

The LineProperties node specifies line parameters used in 2D and 3D rendering.

The lineColor field specifies the color with which to draw the lines and outlines of 2D geometries.

The lineStyle field shall contain the line style type to apply to lines. The allowed values are described in [Table 2](#).

Table 2 — LineProperties: line styles

lineStyle	Description
0	solid
1	dash
2	dot
3	dash-dot
4	dash-dash-dot
5	dash-dot-dot

The terminal shall draw each line style in a manner that is distinguishable from each other line style. The width field determines the width, in the local coordinate system, of rendered lines. The width is not subject to the local transformation. The cap and join style to be used are as follows. The wide lines should end with a square form flush with the end of the lines.

5.1.1.4.4 LinearGradient

5.1.1.4.4.1 XSD description

```

<complexType name="LinearGradientType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="transform" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="endPoint" type="xmta:SFFVec2f" use="optional" default="1 0"/>
  <attribute name="key" type="xmta:MFFloat" use="optional"/>
  <attribute name="keyValue" type="xmta:MFCOLOR" use="optional"/>
  <attribute name="opacity" type="xmta:MFFloat" use="optional" default="1"/>
  <attribute name="spreadMethod" type="xmta:SFFInt32" use="optional" default="0"/>
  <attribute name="startPoint" type="xmta:SFFVec2f" use="optional" default="0 0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="LinearGradient" type="xmta:LinearGradientType"/>

```

5.1.1.4.4.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.76.2.

The LinearGradient node is a texture node that generates a texture procedurally. startPoint and endPoint define the gradient vector, in percentage of the bounds of the object.

Key represents a location along the gradient vector, expressed in percentage of its length. At each location, an RGB color is specified in keyValue. Opacity for each color value can be specified. By default,

colors are 100 % opaque. One value of opacity can be specified meaning all color values have the same opacity, else an opacity shall be specified for each color value.

Transform is an optional parameter that defines how the coordinate system of the gradient can be transformed from the gradient coordinate system onto the target coordinate system. By default, the gradient coordinate system is the same as the object it is applied to. This allows effects such as skewing the gradient.

Only a 2D Transformation node (e.g. Transform2D, TransformMatrix2D) can be present here.

spreadMethod can be pad (0), reflect (1), or repeat (2). It indicates what happens if the gradient starts or ends inside the bounds of the object. Pad means that the last color is used, reflect says to reflect the gradient pattern start-to-end, end-to-start, ... repeatedly until the target object is filled, and repeat says to repeat the gradient pattern start-to-end, start-to-end, ... until the target object is filled.

Opacity for each color value can be specified. By default, colors are 100 % opaque. One value of opacity can be specified, meaning all color values have the same opacity, else an opacity shall be specified for each color value.

5.1.1.4.5 Material

5.1.1.4.5.1 XSD description

```
<complexType name="MaterialType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="ambientIntensity" type="xmta:SFFloat" use="optional"
default="0.2"/>
  <attribute name="diffuseColor" type="xmta:SFCOLOR" use="optional" default="0.8
0.8 0.8"/>
  <attribute name="emissiveColor" type="xmta:SFCOLOR" use="optional" default="0 0
0"/>
  <attribute name="shininess" type="xmta:SFFloat" use="optional" default="0.2"/>
  <attribute name="specularColor" type="xmta:SFCOLOR" use="optional" default="0 0
0"/>
  <attribute name="transparency" type="xmta:SFFloat" use="optional" default="0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Material" type="xmta:MaterialType"/>
```

5.1.1.4.5.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.27.

The Material node specifies surface material properties for associated geometry nodes and is used by the VRML lighting equations during rendering. All of the fields in the Material node range from 0.0 to 1.0.

The fields in the Material node determine how light reflects off an object to create colour.

- The *ambientIntensity* field specifies how much ambient light from light sources this surface shall reflect. Ambient light is omnidirectional and depends only on the number of light sources, not their positions with respect to the surface. Ambient colour is calculated as *ambientIntensity* × *diffuseColor*.
- The *diffuseColor* field reflects all VRML light sources depending on the angle of the surface with respect to the light source. The more directly the surface faces the light, the more diffuse light reflects.
- The *emissiveColor* field models "glowing" objects. This can be useful for displaying pre-lit models (where the light energy of the room is computed explicitly), or for displaying scientific data.
- The *specularColor* and *shininess* fields determine the specular highlights (e.g. the shiny spots on an apple). When the angle from the light to the surface is close to the angle from the surface to the

viewer, the *specularColor* is added to the diffuse and ambient colour calculations. Lower shininess values produce soft glows, while higher values result in sharper, smaller highlights.

- e) The *transparency* field specifies how "clear" an object is, with 1.0 being completely transparent, and 0.0 completely opaque.

5.1.1.4.6 Material2D

5.1.1.4.6.1 XSD description

```
<complexType name="Material2DType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="lineProps" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFLinePropertiesNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="emissiveColor" type="xmta:SFCOLOR" use="optional" default="0.8
0.8 0.8"/>
  <attribute name="filled" type="xmta:SFBool" use="optional" default="false"/>
  <attribute name="transparency" type="xmta:SFFloat" use="optional" default="0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Material2D" type="xmta:Material2DType"/>
```

5.1.1.4.6.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.80.2.

The **Material2D** node specifies the characteristics of a rendered 2D **Shape**. Material2D shall be used as the material field of an Appearance node in certain circumstances (see ISO/IEC 14496-11:2015, 7.2.2.6.2). The **emissiveColor** field specifies the color of the 2D **Shape**. If the shape is not filled, the interior is not drawn. The **filled** field specifies whether rendered nodes are filled or drawn using lines. This field affects **IndexedFaceSet2D**, **Circle** and **Rectangle** nodes. If the rendered node is not filled, the line shall be drawn centered on the rendered node outline. That means that half the line will fall inside the rendered node and the other half outside.

The **lineProps** field contains information about line rendering in the form of a **LineProperties** node. When **filled** is true, if **lineProps** is null, no outline is drawn; if **lineProps** is non-null, an outline is drawn using **lineProps** information. When **filled** is false and **lineProps** is null, an outline is drawn with default width (1), default style (solid) and as line color the emissive color of the Material2D. When **filled** is false and **lineProps** is defined, line color, width and style are taken from the **lineProps** node. See ISO/IEC 14496-11:2015, 7.2.2.75 for more information on **LineProperties**.

The **transparency** field specifies the transparency of the 2D **Shape** and applies both to the filled interior as well as to the outline when drawn. The part of the line which lies outside of the geometry shall not be sensitive to pointer activity. When mapping texture onto a geometry and an outline is to be drawn, the texture shall first mapped onto the geometry, where the geometry dimensions are those without an outline. Then after the geometry is textured the outline shall be drawn.

5.1.1.4.7 Rectangle

5.1.1.4.7.1 XSD description

```
<complexType name="RectangleType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="size" type="xmta:SFFVec2f" use="optional" default="2 2"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
```

```

</complexType>
<element name="Rectangle" type="xmta:RectangleType"/>

```

5.1.1.4.7.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.110.2.

This node specifies a rectangle centered at (0.0) in the local coordinate system. The **size** field specifies the horizontal and vertical size of the rendered rectangle.

5.1.1.4.8 Shape

5.1.1.4.8.1 XSD description

```

<complexType name="ShapeType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="appearance" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFAppearanceNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="geometry" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFGGeometryNodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Shape" type="xmta:ShapeType"/>

```

5.1.1.4.8.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.41.

The Shape node has two fields, *appearance* and *geometry*, which are used to create rendered objects in the world. The *appearance* field contains an Appearance node that specifies the visual attributes (e.g. material and texture) to be applied to the geometry. The *geometry* field contains a geometry node. The specified geometry node is rendered with the specified appearance nodes applied.

ISO/IEC 14772-1:1997, 4.14 contains details of the VRML lighting model and the interaction between Appearance nodes and geometry nodes.

If the *geometry* field is NULL, the object is not drawn.

5.1.1.4.9 SBVCAAnimationV2

5.1.1.4.9.1 XSD description

```

<complexType name="SBVCAAnimationV2Type">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="virtualCharacters" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
  <attribute name="activeUrlIndex" type="xmta:MInt32" use="optional"/>
  <attribute name="loop" type="xmta:SBool" use="optional" default="false"/>
  <attribute name="speed" type="xmta:SFloat" use="optional" default="1"/>
  <attribute name="startTime" type="xmta:SFloat" use="optional" default="0"/>
  <attribute name="stopTime" type="xmta:SFloat" use="optional" default="0"/>
  <attribute name="transitionTime" type="xmta:SFloat" use="optional" default="0"/>
  <attribute name="url" type="xmta:MUrl" use="optional"/>

```

```

    <attributeGroup ref="xmta:DefUseGroup"/>
  </complexType>
  <element name="SBVCAAnimationV2" type="xmta:SBVCAAnimationV2Type"/>

```

5.1.1.4.9.2 Functionality and semantics

As specified in ISO/IEC 14496-16:2011, 4.7.1.7.

This node is an extension of the SBVCAAnimation node and the added functionality consists in streaming control and animation data collection. The BBA stream can be controlled as a elementary media stream, and can be used in connection with the MediaControl node.

The **virtualCharacters** field specifies a list of SBSkinnedModel nodes. The length of the list can be 1 or greater.

The **url** field refers to the BBA stream which contains encoded animation data related to the SBSkinnedModel nodes from the virtualCharacters list and is used for outband bitstreams. The animation will be extracted from the first element of the animation URL list and if the case when it is not available the following element will be used.

The **loop**, **startTime**, and **stopTime** exposedFields and the **isActive** eventOut, and their effects on the SBVCAAnimationV2 node, are similar with the ones described by VRML specifications (ISO/IEC 14772-1) for AudioClip, MovieTexture, and TimeSensor nodes and are described as follows.

The values of the exposedFields are used to determine when the node becomes active or inactive.

The SBVCAAnimationV2 node can execute for 0 or more cycles. A cycle is defined by field data within the node. If, at the end of a cycle, the value of **loop** is FALSE, execution is terminated. Conversely, if **loop** is TRUE at the end of a cycle, a time-dependent node continues execution into the next cycle. A time-dependent node with **loop** TRUE at the end of every cycle continues cycling forever if **startTime** >= **stopTime**, or until **stopTime** if **startTime** < **stopTime**.

The SBVCAAnimationV2 node generates an **isActive** TRUE event when it becomes active and generates an **isActive** FALSE event when it becomes inactive. These are the only times at which an **isActive** event is generated. In particular, **isActive** events are not sent at each tick of a simulation.

The SBVCAAnimationV2 node is inactive until its **startTime** is reached. When time *now* becomes greater than or equal to **startTime**, an **isActive** TRUE event is generated and the SBVCAAnimationV2 node becomes active (*now* refers to the time at which the player is simulating and displaying the virtual world). When a SBVCAAnimationV2 node is read from a mp4 file and the ROUTEs specified within the mp4 file have been established, the node should determine if it is active and, if so, generate an **isActive** TRUE event and begin generating any other necessary events. However, if a SBVCAAnimationV2 node would have become inactive at any time before the reading of the mp4 file, no events are generated upon the completion of the read.

An active SBVCAAnimationV2 node will become inactive when **stopTime** is reached if **stopTime** > **startTime**. The value of **stopTime** is ignored if **stopTime** <= **startTime**. Also, an active SBVCAAnimationV2 node will become inactive at the end of the current cycle if **loop** is FALSE. If an active SBVCAAnimationV2 node receives a *set_loop* FALSE event, execution continues until the end of the current cycle or until **stopTime** (if **stopTime** > **startTime**), whichever occurs first. The termination at the end of cycle can be overridden by a subsequent *set_loop* TRUE event.

Any *set_startTime* events to an active SBVCAAnimationV2 node are ignored. Any *set_stopTime* event where **stopTime** <= **startTime** sent to an active SBVCAAnimationV2 node is also ignored. A *set_stopTime* event where **startTime** < **stopTime** <= *now* sent to an active SBVCAAnimationV2 node results in events being generated as if **stopTime** has just been reached. That is, final events, including an **isActive** FALSE, are generated and the node becomes inactive. The **stopTime_changed** event will have the *set_stopTime* value.

A SBVCAAnimationV2 node may be restarted while it is active by sending a *set_stopTime* event equal to the current time (which will cause the node to become inactive) and a *set_startTime* event, setting it to the current time or any time in the future. These events will have the same time stamp and should be processed as *set_stopTime*, then *set_startTime* to produce the correct behaviour.

The **speed** exposedField controls playback speed. It does not affect the delivery of the stream attached to the **SBVCAnimationV2** node. For streaming media, value of **speed** other than 1 shall be ignored.

A **SBVCAnimationV2** shall display first frame if **speed** is 0. For positive values of **speed**, the frame that an active **SBVCAnimationV2** will display at time *now* corresponds to the frame at animation time (i.e. in the animation's local time base with frame 0 at time 0, at speed = 1):

$$\text{fmod}(\text{now} - \text{startTime}, \text{duration}/\text{speed})$$

If **speed** is negative, then the frame to display is the frame at animation time:

$$\text{duration} + \text{fmod}(\text{now} - \text{startTime}, \text{duration}/\text{speed}).$$

When a **SBVCAnimationV2** becomes inactive, the frame corresponding to the time at which the **SBVCAnimationV2** became inactive shall persist. The **speed** exposedField indicates how fast the movie shall be played. A speed of 2 indicates the animation plays twice as fast. Note that the **duration_changed** eventOut is not affected by the **speed** exposedField. **set_speed** events shall be ignored while the animation is playing.

An event shall be generated via the **duration_changed** field whenever a change is made to the **startTime** or **stopTime** fields. An event shall also be triggered if these fields are changed simultaneously, even if the duration does not actually change.

activeUrlIndex allows to select or to combine specific animation resource referred in the **url[]** field. When this field is instantiated, the behavior of the **url[]** field changes from the alternative selection into a combined selection. In the case of alternative mode, if the first resource in the **url[]** field is not available, the second one will be used, and so on. In the combined mode, the following cases can occur.

- a) **activeUrlIndex** has one field: the resource from **url[]** that has this index is used for animation. When the **activeUrlIndex** is updated a transition between to the old animation (frame) and the new one is performed. The transition use linear interpolation for translation, center and scale and SLERP for spherical data as rotation and scaleOrientation. The time of transition is specified by using the **transitionTime** field.
- b) **activeUrlIndex** has several fields: a composition between the two resources is performed by the terminal: for the bones that are common in two or more resources a mean procedure has to be applied. The mean is computed by using linear interpolation for translation, center and scale and SLERP for spherical data as rotation and scaleOrientation.

In all the cases, when a transition between two animation resources is needed, when the **transitionTime** is not zero, a interpolation shall be performed by the player. The **transitionTime** is specified in milliseconds.

5.1.1.4.10 SBBone

5.1.1.4.10.1 XSD Description

```
<complexType name="SBBoneType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="children" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
  <attribute name="boneID" type="xmta:SFInt32" use="optional" default="0"/>
  <attribute name="center" type="xmta:SFFVec3f" use="optional" default="0 0 0"/>
  <attribute name="endpoint" type="xmta:SFFVec3f" use="optional" default="0 0 1"/>
  <attribute name="falloff" type="xmta:SFInt32" use="optional" default="1"/>
  <attribute name="ikChainPosition" type="xmta:SFInt32" use="optional" default="0"/>
  <attribute name="ikPitchLimit" type="xmta:MFFloat" use="optional"/>
  <attribute name="ikRollLimit" type="xmta:MFFloat" use="optional"/>
</complexType>
```

```

    <attribute name="ikTxLimit" type="xmta:MFFloat" use="optional"/>
    <attribute name="ikTyLimit" type="xmta:MFFloat" use="optional"/>
    <attribute name="ikTzLimit" type="xmta:MFFloat" use="optional"/>
    <attribute name="ikYawLimit" type="xmta:MFFloat" use="optional"/>
    <attribute name="rotation" type="xmta:SFRotation" use="optional" default="0 0 1
0"/>
    <attribute name="rotationOrder" type="xmta:SFINt32" use="optional" default="0"/>
    <attribute name="scale" type="xmta:SFFVec3f" use="optional" default="1 1 1"/>
    <attribute name="scaleOrientation" type="xmta:SFRotation" use="optional"
default="0 0 1 0"/>
    <attribute name="sectionInner" type="xmta:MFFloat" use="optional"/>
    <attribute name="sectionOuter" type="xmta:MFFloat" use="optional"/>
    <attribute name="sectionPosition" type="xmta:MFFloat" use="optional"/>
    <attribute name="skinCoordIndex" type="xmta:MFINt32" use="optional"/>
    <attribute name="skinCoordWeight" type="xmta:MFFloat" use="optional"/>
    <attribute name="translation" type="xmta:SFFVec3f" use="optional" default="0 0 0"/>
    <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="SBBone" type="xmta:SBBoneType"/>

```

5.1.1.4.10.2 Functionality and semantics

As specified in ISO/IEC 14496-1:2002, 6.8.1.1.2.

SBBone node specifies data related to one bone from the skeleton.

The **boneID** field is a unique identifier which allows that the bone to be addressed at animation run-time.

The **center** field specifies a translation offset from the origin of the local coordinate system.

The **translation** field specifies a translation to the bone coordinate system.

The **rotation** field specifies a rotation of the bone coordinate system.

The **scale** field specifies a non-uniform scale of the bone coordinate system. **scale** values shall be greater than zero.

The **scaleOrientation** specifies a rotation of the bone coordinate system before the scale (to specify scales in arbitrary orientations). The **scaleOrientation** applies only to the scale operation.

The possible geometric 3D transformation consists of (in order): 1) (possibly) non-uniform scale about an arbitrary point, 2) a rotation about an arbitrary point and axis and 3) a translation.

The **rotationOrder** field specifies the rotation order when deals with the decomposition of the rotation in respect with system coordinate axes.

Two ways of specifying the influence region of the bone are allowed:

The **skinCoordIndex** field contains a list of indices of all skin vertices affected by the current bone. Mostly, the skin influence region of bone will contain vertices from the 3D neighborhood of the bone, but special cases of influence are also accepted.

The **skinCoordWeight** field contains a list of weights (one per vertex listed in **skinCoordIndex**) that measures the contribution of the current bone to the vertex in question. The length **skinCoordIndex** is equal with the length of **skinCoordWeight**. The sum of all **skinCoordWeight** related to the same vertex shall be 1.

The **endpoint** field specifies the bone 3D end point and is used to compute the bone length.

The **sectionInner** field is a list of inner influence region radii for different sections.

The **sectionOuter** field is a list of outer influence region radii for different sections.

The **sectionPosition** field is a list of positions of all the sections defined by the designer.

The **falloff** field specifies the function between the amplitude affectedness and distance: -1 for x^3 , 0 for x^2 , 1 for x , 2 for $\sin x$, 3 for $x^{1/2}$ and 4 for $x^{1/3}$.

The two schemes can be used independently or in combination, in which case the individual vertex weights take precedence.

The **ikChainPosition** field specifies the position of the bone in the kinematics chain. If the bone is the root of the IK chain then **ikChainPosition**=1. In this case, when applying IK scheme, only the orientation of the bone is changed. If the bone is last in the kinematics chain **ikChainPosition**=2. In this case, the animation stream has to include the desired position of the bone (X, Y and Z world coordinates). If **ikChainPosition**=3 the bone belongs to the IK chain but is not the first or the last one in the chain. In this case, position and orientation of the bone are computed by the IK procedure. Finally, if the bone does not belong to any IK chain (**ikChainPosition**=0), it is necessary to transmit the bone local transformation in order to animate the bone. If an animation stream contains motion information about a bone which has **ikChainPosition** 1, this information will be ignored. If an animation stream contains motion information about a bone which has **ikChainPosition** 3, this means that the animation producer wants to ensure the orientation of the bone and the IK solver will use this value as a constrain.

The **ikYawLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the X axis.

The **ikPitchLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the Y axis.

The **ikRollLimit** field consists in a pair of min/max values which limit the bone rotation with respect to the Z axis.

The **ikTxLimit** field consists in a pair of min/max values which limit the bone translation in the X direction.

The **ikTyLimit** field consists in a pair of min/max values which limit the bone translation in the Y direction.

The **ikTzLimit** field consists in a pair of min/max values which limit the bone translation in the Z direction.

The **SBBone** node is used as a building block to describe the hierarchy of the articulated model by attaching one or more child objects. The **children** field has the same semantic as used in BIFS; the absolute geometric transformation of any child of a bone is obtained through a composition with the bone-parent transformation.

5.1.1.4.11 SBSegment

5.1.1.4.11.1 XSD description

```
<complexType name="SBSegmentType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="children" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
  <attribute name="centerOfMass" type="xmta:SFFVec3f" use="optional" default="0 0
0"/>
  <attribute name="mass" type="xmta:SFFloat" use="optional" default="0"/>
  <attribute name="momentsOfInertia" type="xmta:MFFVec3f" use="optional" default="0
0 0 0 0 0 0"/>
  <attribute name="name" type="xmta:SFFString" use="optional"
default="&quot;&quot;"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
```

```
</complexType>
<element name="SBSegment" type="xmta:SBSegmentType"/>
```

5.1.1.4.11.2 Functionality and semantics

As specified in ISO/IEC 14496-1:2002, 6.8.1.2.2.

The **name** field shall be present, so that the **SBSegment** can be identified at runtime. Each **SBSegment** should have a DEF name that matches the **name** field for that Segment, but with a distinguishing prefix in front of it.

The **mass** field is the total mass of the segment.

The **centerOfMass** field is the location within the segment of its center of mass. Note that a zero value was chosen for the mass in order to give a clear indication that no mass value is available.

The **momentsOfInertia** field contains the moment of inertia matrix. The first three elements are the first row of the 3x3 matrix, the next three elements are the second row, and the final three elements are the third row.

The **children** field can be any object attached at this level of the skeleton, including a **SBSkinnedModel**.

An **SBSegment** node is a grouping node especially introduced to address two issues.

- a) The first one is to the requirement to separate different parts from the skinned model into deformation-independent parts. Between two deformation-independent parts the geometrical transformation of one of them do not imply skin deformations on the other. This is essential for run-time animation optimization. The **SBSegment** node may contain as a child an **SBSkinnedModel** node (see the **SBSkinnedModel** node description below). Portions of the model which are not part of the seamless mesh can be attached to the skeleton hierarchy by using an **SBSegment** node.
- b) The second deals with the requirement to attach standalone 3D objects at different parts of the skeleton hierarchy. For example, a ring can be attached to a finger; the ring geometry and attributes are defined outside of skinned model but the ring will have the same local geometrical transformation as the attached bone.

5.1.1.4.12 SBSite

5.1.1.4.12.1 XSD description

```
<complexType name="SBSiteType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="children" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
  <attribute name="center" type="xmta:SFFVec3f" use="optional" default="0 0 0"/>
  <attribute name="name" type="xmta:SFFString" use="optional"
default="&quot;&quot;"/>
  <attribute name="rotation" type="xmta:SFFRotation" use="optional" default="0 0 1
0"/>
  <attribute name="scale" type="xmta:SFFVec3f" use="optional" default="1 1 1"/>
  <attribute name="scaleOrientation" type="xmta:SFFRotation" use="optional"
default="0 0 1 0"/>
  <attribute name="translation" type="xmta:SFFVec3f" use="optional" default="0 0
0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="SBSite" type="xmta:SBSiteType"/>
```

5.1.1.4.12.2 Functionality and semantics

As specified in ISO/IEC 14496-1:2002, 6.8.1.3.2.

The **center** field specifies a translation offset and can be used to compute a bone length. The **rotation** field specifies a rotation of the coordinate system of the **SBSite** node.

The **scale** field specifies a non-uniform scale of the **SBSite** node coordinate system and the **scale** values shall be greater than zero.

The **scaleOrientation** specifies a rotation of the coordinate system of the **SBSite** node before the scale, thus allowing a scale at an arbitrary orientation. The **scaleOrientation** applies only to the scale operation.

The **translation** field specifies a translation of the coordinate system of the **SBSite** node.

The **children** field is used to store any object that can be attached to the **SBSegment** node.

The **SBSite** node can be used for three purposes. The first is to define an "end effector", i.e. a location which can be used by an inverse kinematics system. The second is to define an attachment point for accessories such as clothing. The third is to define a location for a virtual camera in the reference frame of a **SBSegment** node.

SBSite nodes are stored within the **children** field of an **SBSegment** node. The **SBSite** node is a specialized grouping node that defines a coordinate system for nodes in its **children** field that is relative to the coordinate systems of its parent node. The reason a **SBSite** node is considered a specialized grouping node is that it can only be defined as a child of a **SBSegment** node.

5.1.1.4.13 SBSkinnedModel

5.1.1.4.13.1 XSD description

```
<complexType name="SBSkinnedModelType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="bones" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFSBBoneNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
    <element name="muscles" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFSBMuscleNodeType" minOccurs="0"
maxOccurs="unbounded"/>
      </complexType>
    </element>
    <element name="segments" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFSBSegmentNodeType" minOccurs="0"
maxOccurs="unbounded"/>
      </complexType>
    </element>
    <element name="sites" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFSBSiteNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
    <element name="skeleton" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
    <element name="skin" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
</complexType>
```

```

</element>
<element name="skinCoord" form="qualified" minOccurs="0">
  <complexType>
    <group ref="xmta:SFCoordinateNodeType" minOccurs="0"/>
  </complexType>
</element>
<element name="skinNormal" form="qualified" minOccurs="0">
  <complexType>
    <group ref="xmta:SFCNormalNodeType" minOccurs="0"/>
  </complexType>
</element>
<element name="weighsComputationSkinCoord" form="qualified" minOccurs="0">
  <complexType>
    <group ref="xmta:SF3DNodeType" minOccurs="0"/>
  </complexType>
</element>
</all>
<attribute name="center" type="xmta:SFCVec3f" use="optional" default="0 0 0"/>
<attribute name="name" type="xmta:SFCString" use="optional"
default="&quot;&quot;"/>
<attribute name="rotation" type="xmta:SFCRotation" use="optional" default="0 0 1
0"/>
<attribute name="scale" type="xmta:SFCVec3f" use="optional" default="1 1 1"/>
<attribute name="scaleOrientation" type="xmta:SFCRotation" use="optional"
default="0 0 1 0"/>
<attribute name="translation" type="xmta:SFCVec3f" use="optional" default="0 0
0"/>
<attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="SBSSkinnedModel" type="xmta:SBSSkinnedModelType"/>

```

5.1.1.4.13.2 Functionality and semantics

As specified in ISO/IEC 14496-16:2011, 4.5.2.1.2.

The **SBSSkinnedModel** node is the top of the hierarchy of Skin&Bones related nodes and contains the definition parameters for the entire seamless model or of a seamless part of the model.

The **name** field specify the name of the skinned model allowing easily identification at the animation run-time.

The **center** field specifies a translation offset from the origin of the local coordinate system.

The **translation** field specifies a translation of the coordinate system.

The **rotation** field specifies a rotation of the coordinate system

The **scale** field specifies a non-uniform scale of the coordinate system. **scale** values shall be greater than zero.

The **scaleOrientation** specifies a rotation of the coordinate system before the scale (to specify scales in arbitrary orientations). The **scaleOrientation** applies only to the scale operation.

The **skinCoord** contains the 3d coordinates of all vertices of the seamless model.

The **skin** consists of a collection of shapes that share the same **skinCoord**. This mechanism allows considering the model as a continuous mesh and, in the same time, to attach different attributes (like color, texture) to different parts of the model.

The **skeleton** field specifies the root of the bones hierarchy.

The **bones** fields consist in the lists of all bones previously defined as **SBBone** node.

The **segments** fields consist in the lists of all bones previously defined as **SBSegment** node.

The **sites** fields consist in the lists of all bones previously defined as **SBSites** node. The **muscles** fields consist in the lists of all bones previously defined as **SBMuscle** node.

The **weighsComputationSkinCoord** field describes a specific static position of the skinned model. In many cases the static position of the articulated model defined by **skinCoord** and **skin** fields is not appropriate to compute the influence region of a bone. In this case the **weighsComputationSkinCoord** field allows specifying the skinned model vertices in a more appropriate static posture. This posture will be used just during the initialization stage and ignored during the animation. All the skeleton transformations are related to the posture defined by **skinCoord** field.

5.1.1.4.14 MorphShape

5.1.1.4.14.1 XSD description

```
<complexType name="MorphShapeType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="baseShape" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="targetShapes" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SF3DNodeType" minOccurs="0" maxOccurs="unbounded"/>
      </complexType>
    </element>
  </all>
  <attribute name="morphID" type="xmta:SFInt32" use="optional" default="0"/>
  <attribute name="weights" type="xmta:MFFloat" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="MorphShape" type="xmta:MorphShapeType"/>
```

5.1.1.4.14.2 Functionality and semantics

As specified in ISO/IEC 14496-16:2011, 4.3.6.

morphID: a unique identifier between 0 and 1023 which allows that the morph to be addressed at animation run-time.

baseShape: a Shape node that represent the base mesh. The geometry field of the baseShape can be any geometry supported by ISOIEC 14496 (e.g. IndexedFaceSet, IndexedLineSet, SolidRep).

targetShapes: a vector of Shapes nodes representing the shape of the target meshes. The tool used for defining an appearance and a geometry of a target shape shall be the same as the tool used for defining the appearance and the geometry of the base shape (e.g. if the baseShape is defined by using IndexedFaceSet, all the target shapes shall be defined by using IndexedFaceSet).

weights: a vector of integers of the same size as the **targetShapes**. The morphed shape is obtained according to [Formula \(1\)](#):

$$M = B + \sum_{i=1}^n (T_i - B) * w_i \quad (1)$$

where

M is the morphed shape;

B is the base shape;

T_i is the target shape i ;

W_i is the weight of the T_i .

The morphing is performed for all the components of the Shape (Appearance and Geometry) that have different values in the base shape and the target shapes (e.g. if the base shape and the target shapes are defined by using IndexedFaceSet and the *coord* field contains different values in the base shape and in the target geometries, the *coord* component of the morph shape is obtained by using previous equation applied to the *coord* field. Note that the size of the *coord* field shall be the same for the base shapes and the target shapes).

If the shapes (base and targets) are defined by using IndexedFaceSet, a typical decoder should support morphing of the following geometry components: *coord*, *normals*, *color*, *texCoord*.

5.1.1.4.15 Coordinate

5.1.1.4.15.1 XSD description

```
<complexType name="CoordinateType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="point" type="xmta:MVec3f" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Coordinate" type="xmta:CoordinateType"/>
```

5.1.1.4.15.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.12.

This node defines a set of 3D coordinates to be used in the *coord* field of vertex-based geometry nodes including IndexedFaceSet, IndexedLineSet, and PointSet.

5.1.1.4.16 TextureCoordinate

5.1.1.4.16.1 XSD description

```
<complexType name="TextureCoordinateType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="point" type="xmta:MVec2f" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="TextureCoordinate" type="xmta:TextureCoordinateType"/>
```

5.1.1.4.16.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.48.

The TextureCoordinate node specifies a set of 2D texture coordinates used by vertex-based geometry nodes (e.g. IndexedFaceSet and ElevationGrid) to map textures to vertices. Textures are two dimensional colour functions that, given an (s, t) coordinate, return a colour value $colour(s, t)$. Texture map values (ImageTexture, MovieTexture, and PixelTexture) range from $[0.0, 1.0]$ along the S-axis and T-axis. However, TextureCoordinate values, specified by the *point* field, may be in the range $(-\infty, \infty)$. Texture coordinates identify a location (and thus a colour value) in the texture map. The horizontal coordinate s is specified first, followed by the vertical coordinate t .

If the texture map is repeated in a given direction (S-axis or T-axis), a texture coordinate C (s or t) is mapped into a texture map that has N pixels in the given direction as follows:

Texture map location = $[C - \text{floor}(C)] \times N$

If the texture map is not repeated, the texture coordinates are clamped to the 0.0 to 1.0 range as follows:

```
Texture map location = N,      if C > 1,0,
                      = 0,0,   if C < 0,0,
                      = C × N, if 0,0 ≤ C ≤ 1,0.
```

5.1.1.4.17 Normal

5.1.1.4.17.1 XSD description

```
<complexType name="NormalType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="vector" type="xmta:MVec3f" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Normal" type="xmta:NormalType"/>
```

5.1.1.4.17.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.30.

This node defines a set of 3D surface normal vectors to be used in the *vector* field of some geometry nodes (e.g. IndexedFaceSet and ElevationGrid). This node contains one multiple-valued field that contains the normal vectors. Normals shall be of unit length.

5.1.1.4.18 IndexedFaceSet

5.1.1.4.18.1 XSD description

```
<complexType name="IndexedFaceSetType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element name="color" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFCOLORNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="coord" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFCOORDINATENodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="normal" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFNORMALNodeType" minOccurs="0"/>
      </complexType>
    </element>
    <element name="texCoord" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFTEXCOORDINATENodeType" minOccurs="0"/>
      </complexType>
    </element>
  </all>
  <attribute name="ccw" type="xmta:SFBool" use="optional" default="true"/>
  <attribute name="colorIndex" type="xmta:MInt32" use="optional"/>
  <attribute name="colorPerVertex" type="xmta:SFBool" use="optional"
default="true"/>
  <attribute name="convex" type="xmta:SFBool" use="optional" default="true"/>
  <attribute name="coordIndex" type="xmta:MInt32" use="optional"/>
  <attribute name="creaseAngle" type="xmta:SFFloat" use="optional" default="0"/>
  <attribute name="normalIndex" type="xmta:MInt32" use="optional"/>
  <attribute name="normalPerVertex" type="xmta:SFBool" use="optional"
default="true"/>
  <attribute name="solid" type="xmta:SFBool" use="optional" default="true"/>
  <attribute name="texCoordIndex" type="xmta:MInt32" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="IndexedFaceSet" type="xmta:IndexedFaceSetType"/>
```

5.1.1.4.18.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.23.

The IndexedFaceSet node represents a 3D shape formed by constructing faces (polygons) from vertices listed in the *coord* field. The *coord* field contains a Coordinate node that defines the 3D vertices referenced by the *coordIndex* field. IndexedFaceSet uses the indices in its *coordIndex* field to specify the polygonal faces by indexing into the coordinates in the Coordinate node. An index of "-1" indicates that the current face has ended and the next one begins. The last face may be (but does not have to be) followed by a "-1" index. If the greatest index in the *coordIndex* field is N, the Coordinate node shall contain N+1 coordinates (indexed as 0 to N). Each face of the IndexedFaceSet shall have:

- a) at least three non-coincident vertices;
- b) vertices that define a planar polygon;
- c) vertices that define a non-self-intersecting polygon.

Otherwise, the results are undefined.

The IndexedFaceSet node is specified in the local coordinate system and is affected by the transformations of its ancestors.

Descriptions of the *coord*, *normal*, and *texCoord* fields are provided in the Coordinate, Normal, and TextureCoordinate nodes, respectively.

Details on lighting equations and the interaction between *color* field, *normal* field, textures, materials, and geometries are provided in ISO/IEC 14772-1:1997, 4.14.

If the *color* field is not NULL, it shall contain a Color node whose colours are applied to the vertices or faces of the IndexedFaceSet as follows:

- If *colorPerVertex* is FALSE, colours are applied to each face, as follows.
 - If the *colorIndex* field is not empty, then one colour is used for each face of the IndexedFaceSet. There shall be at least as many indices in the *colorIndex* field as there are faces in the IndexedFaceSet. If the greatest index in the *colorIndex* field is N, then there shall be N+1 colours in the Color node. The *colorIndex* field shall not contain any negative entries.
 - If the *colorIndex* field is empty, then the colours in the Color node are applied to each face of the IndexedFaceSet in order. There shall be at least as many colours in the Color node as there are faces.
- If *colorPerVertex* is TRUE, colours are applied to each vertex, as follows.
 - If the *colorIndex* field is not empty, then colours are applied to each vertex of the IndexedFaceSet in exactly the same manner that the *coordIndex* field is used to choose coordinates for each vertex from the Coordinate node. The *colorIndex* field shall contain at least as many indices as the *coordIndex* field, and shall contain end-of-face markers (-1) in exactly the same places as the *coordIndex* field. If the greatest index in the *colorIndex* field is N, then there shall be N+1 colours in the Color node.
 - If the *colorIndex* field is empty, then the *coordIndex* field is used to choose colours from the Color node. If the greatest index in the *coordIndex* field is N, then there shall be N+1 colours in the Color node.

If the *color* field is NULL, the geometry shall be rendered normally using the Material and texture defined in the Appearance node (see ISO/IEC 14772-1:1997, 4.14 for details).

If the *normal* field is not NULL, it shall contain a Normal node whose normals are applied to the vertices or faces of the IndexedFaceSet in a manner exactly equivalent to that described above for applying colours to vertices/faces (where *normalPerVertex* corresponds to *colorPerVertex* and *normalIndex*

corresponds to *colorIndex*). If the *normal* field is NULL, the browser shall automatically generate normals, using *creaseAngle* to determine if and how normals are smoothed across shared vertices (see ISO/IEC 14772-1:1997, 4.6.3.5, Crease angle field).

If the *texCoord* field is not NULL, it shall contain a TextureCoordinate node. The texture coordinates in that node are applied to the vertices of the IndexedFaceSet as follows.

- If the *texCoordIndex* field is not empty, then it is used to choose texture coordinates for each vertex of the IndexedFaceSet in exactly the same manner that the *coordIndex* field is used to choose coordinates for each vertex from the Coordinate node. The *texCoordIndex* field shall contain at least as many indices as the *coordIndex* field, and shall contain end-of-face markers (–1) in exactly the same places as the *coordIndex* field. If the greatest index in the *texCoordIndex* field is N, then there shall be N+1 texture coordinates in the TextureCoordinate node.
- If the *texCoordIndex* field is empty, then the *coordIndex* array is used to choose texture coordinates from the TextureCoordinate node. If the greatest index in the *coordIndex* field is N, then there shall be N+1 texture coordinates in the TextureCoordinate node.

If the *texCoord* field is NULL, a default texture coordinate mapping is calculated using the local coordinate system bounding box of the shape. The longest dimension of the bounding box defines the S coordinates, and the next longest defines the T coordinates. If two or all three dimensions of the bounding box are equal, ties shall be broken by choosing the X, Y, or Z dimension in that order of preference. The value of the S coordinate ranges from 0 to 1, from one end of the bounding box to the other. The T coordinate ranges between 0 and the ratio of the second greatest dimension of the bounding box to the greatest dimension.

Some restrictions specified in ISO/IEC 14496-11:2015, 7.2.2.66.2.

The **IndexedFaceSet** node represents a 3D polygon mesh formed by constructing faces (polygons) from points specified in the **coord** field. If the **coordIndex** field is not NULL, **IndexedFaceSet** uses the indices in its **coordIndex** field to specify the polygonal faces by connecting together points from the **coord** field. An index of –1 shall indicate that the current face has ended and the next one begins. The last face may be followed by a –1. **IndexedFaceSet** shall be specified in the local coordinate system and shall be affected by parent transformations. The **coord** field specifies the vertices of the face set and is specified by **Coordinate** node. If the **coordIndex** field is not NULL, the indices of the **coordIndex** field shall be used to specify the faces by connecting together points from the **coord** field. An index of –1 shall indicate that the current face has ended and the next one begins.

The last face may be followed by a –1. If the **coordIndex** field is NULL, the vertices of the **coord** field are laid out in their respective order to specify one face. If the **color** field is NULL and there is a **Material** node defined for the **Appearance** affecting this **IndexedFaceSet**, then the **emissiveColor** of the **Material** node shall be used to draw the faces.

In order to use 3D Mesh Coding (3DMC) with the **IndexedFaceSet** node, the **use3DMeshCoding** flag in BIFSv2Config should be set to TRUE, as described in ISO/IEC 14496-11:2015, 8.5.3.2. This will require every IndexedFaceSet node in that elementary stream to be coded with 3DMC. Note that 3DMC does not support the use of DEF and USE within the fields of **IndexedFaceSet**. Also, an empty **IndexedFaceSet** should not be included in a stream where use3DmeshCoding flag is set to TRUE. A scene with both 3DMC coded and BIFS coded **IndexedFaceSet** nodes can be created by sending the compressed and uncompressed nodes in separate streams. This can be done with an **Inline** node or by sending separate elementary streams in the same object descriptor. The latter approach has the advantage of keeping the nodes in the same name space, see the example in ISO/IEC 14496-11:2015, 7.8.

5.1.1.4.19 IndexedLineSet

5.1.1.4.19.1 XSD description

```
<complexType name="IndexedLineSetType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
```

```

    <element name="color" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFCOLORNodeType" minOccurs="0" />
      </complexType>
    </element>
    <element name="coord" form="qualified" minOccurs="0">
      <complexType>
        <group ref="xmta:SFCOORDINATENodeType" minOccurs="0" />
      </complexType>
    </element>
  </all>
  <attribute name="colorIndex" type="xmta:MFIN32" use="optional"/>
  <attribute name="colorPerVertex" type="xmta:SFBOL" use="optional" default="true" />
/>

  <attribute name="coordIndex" type="xmta:MFIN32" use="optional"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="IndexedLineSet" type="xmta:IndexedLineSetType"/>

```

5.1.1.4.19.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.24.

The *IndexedLineSet* node represents a 3D geometry formed by constructing polylines from 3D vertices specified in the *coord* field. *IndexedLineSet* uses the indices in its *coordIndex* field to specify the polylines by connecting vertices from the *coord* field. An index of "-1" indicates that the current polyline has ended and the next one begins. The last polyline may be (but does not have to be) followed by a "-1". *IndexedLineSet* is specified in the local coordinate system and is affected by the transformations of its ancestors.

The *coord* field specifies the 3D vertices of the line set and contains a *Coordinate* node.

Lines are not lit, are not texture-mapped, and do not participate in collision detection. The width of lines is implementation dependent and each line segment is solid (i.e. not dashed).

If the *color* field is not NULL, it shall contain a *Color* node. The colours are applied to the line(s) as follows:

- a) If *colorPerVertex* is FALSE.
 - 1) If the *colorIndex* field is not empty, one colour is used for each polyline of the *IndexedLineSet*. There shall be at least as many indices in the *colorIndex* field as there are polylines in the *IndexedLineSet*. If the greatest index in the *colorIndex* field is N, there shall be N+1 colours in the *Color* node. The *colorIndex* field shall not contain any negative entries.
 - 2) If the *colorIndex* field is empty, the colours from the *Color* node are applied to each polyline of the *IndexedLineSet* in order. There shall be at least as many colours in the *Color* node as there are polylines.
- b) If *colorPerVertex* is TRUE.
 - If the *colorIndex* field is not empty, colours are applied to each vertex of the *IndexedLineSet* in exactly the same manner that the *coordIndex* field is used to supply coordinates for each vertex from the *Coordinate* node. The *colorIndex* field shall contain at least as many indices as the *coordIndex* field and shall contain end-of-polyline markers (-1) in exactly the same places as the *coordIndex* field. If the greatest index in the *colorIndex* field is N, there shall be N+1 colours in the *Color* node.
 - If the *colorIndex* field is empty, the *coordIndex* field is used to choose colours from the *Color* node. If the greatest index in the *coordIndex* field is N, there shall be N+1 colours in the *Color* node.

If the *color* field is NULL and there is a *Material* defined for the *Appearance* affecting this *IndexedLineSet*, the *emissiveColor* of the *Material* shall be used to draw the lines. Details on lighting equations as they affect *IndexedLineSet* nodes are described in ISO/IEC 14772-1:1997, 4.14.

5.1.2 Programming information

The following programming related node is used in ARAF: Script.

5.1.2.1 Script

5.1.2.1.1 XSD description

```
<complexType name="ScriptType">
  <sequence>
    <element ref="xmta:field" minOccurs="0" maxOccurs="unbounded"/>
    <element ref="xmta:IS" minOccurs="0" maxOccurs="unbounded"/>
  </sequence>
  <attribute name="url" type="xmta:MFScript" use="optional"/>
  <attribute name="directOutput" type="xmta:SFBool" use="optional" default="false"/>
  <attribute name="mustEvaluate" type="xmta:SFBool" use="optional" default="false"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="Script" type="xmta:ScriptType"/>
```

5.1.2.1.2 Functionality and semantics

As defined in ISO/IEC 14772-1:1997, 6.40.

The Script node is used to program behaviour in a scene. Script nodes typically

- signify a change or user action,
- receive events from other nodes,
- contain a program module that performs some computation, and
- effect change somewhere else in the scene by sending events.

Each Script node has associated programming language code, referenced by the *url* field, that is executed to carry out the Script node's function. That code is referred to as the "script" in the rest of this description. Details on the *url* field can be found in ISO/IEC 14772-1:1997, 4.5.

Browsers are not required to support any specific language. Detailed information on scripting languages is described in ISO/IEC 14772-1:1997, 4.12 supporting a scripting language for which a language binding is specified shall adhere to that language binding.

Sometime before a script receives the first event it shall be initialized (any language-dependent or user-defined *initialize()* is performed). The script is able to receive and process events that are sent to it. Each event that can be received shall be declared in the Script node using the same syntax as is used in a prototype definition:

eventIn type name

The *type* can be any of the standard VRML fields (as defined in ISO/IEC 14772-1:1997, Clause 5). *Name* shall be an identifier that is unique for this Script node.

The Script node is able to generate events in response to the incoming events. Each event that may be generated shall be declared in the Script node using the following syntax:

eventOut type name

With the exception of the *url* field, *exposedFields* are not allowed in Script nodes.

If the Script node's *mustEvaluate* field is FALSE, the browser may delay sending input events to the script until its outputs are needed by the browser. If the *mustEvaluate* field is TRUE, the browser shall send input events to the script as soon as possible, regardless of whether the outputs are needed. The *mustEvaluate* field shall be set to TRUE only if the Script node has effects that are not known to the browser (such as sending information across the network). Otherwise, poor performance may result.

Once the script has access to a VRML node (via an SFNode or MFNode value either in one of the Script node's fields or passed in as an eventIn), the script is able to read the contents of that node's exposed fields. If the Script node's *directOutput* field is TRUE, the script may also send events directly to any node to which it has access, and may dynamically establish or break routes. If *directOutput* is FALSE (the default), the script may only affect the rest of the world via events sent through its eventOuts. The results are undefined if *directOutput* is FALSE and the script sends events directly to a node to which it has access.

A script is able to communicate directly with the VRML browser to get information such as the current time and the current world URL. This is strictly defined by the API for the specific scripting language being used.

The location of the Script node in the scene graph has no affect on its operation. For example, if a parent of a Script node is a Switch node with whichChoice set to "-1" (i.e. ignore its children), the Script node continues to operate as specified (i.e. it receives and sends events).

5.1.3 User interactivity

5.1.3.1 General

The following user interactivity related nodes are used in ARAF: InputSensor, SphereSensor, TimeSensor, TouchSensor, MediaSensor, PlaneSensor.

5.1.3.2 InputSensor

5.1.3.2.1 XSD description

```
<complexType name="InputSensorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
    <element ref="xmta:buffer" minOccurs="0"/>
  </all>
  <attribute name="enabled" type="xmta:SFBBool" use="optional" default="true"/>
  <attribute name="url" type="xmta:MFUrl" use="optional" default="&quot;&quot;"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="InputSensor" type="xmta:InputSensorType"/>
```

5.1.3.2.2 Functionality and semantics

As defined in ISO/IEC 14496-11:2015, 7.2.2.71.2.

The **InputSensor** node is used to add entry points for user inputs into a BIFS scene. It allows user events to trigger updates of the value of a field or the value of an element of a multiple field of an existing node.

Input devices are modelled as devices that generate frames of user input data. A device data frame (DDF) consists in a *list of values of any of the allowed types for node fields*. Values from DDFs are used to update the scene. For example, the DDF definition for a simple mouse is:

MouseDataFrame [

SFVec2f cursorPosition

SFBool singleButtonDown

]

The encoding of the DDF is implementation-dependent. Devices may send only complete DDF or sometimes subsets of DDF as well. The **buffer** field is a buffered bit string which contains a list of BIFS-Commands in the form of a CommandFrame (see ISO/IEC 14496-11, 8.6.2). Allowed BIFS-Commands are the following: FieldReplacement (see ISO/IEC 14496-11, 8.6.21), IndexedValueReplacement (see ISO/IEC 14496-11, 8.6.22) and NodeDeletion with a NULL node argument (see ISO/IEC 14496-11,

8.7.3.2). The **buffer** shall contain a number of BIFSCCommands that matches the number of fields in the DDF definition for the attached device. The type of the field replaced by the n^{th} command in the **buffer** shall match the type of the n^{th} field in the DDF definition.

The **url** field specifies the data source to be used (see ISO/IEC 14496-11, 7.1.1.2.7.1). The **url** field shall point to a stream of type UserInteractionStream, which “access units” are DDFs. When the **enabled** is set to TRUE, upon reception of a DDF, each value (in the order of the DDF definition) is placed in the corresponding replace command according to the DDF definition, and then the replace command is executed. These updates are not time-stamped; they are executed at the time of the event, assuming a zero-decoding time. It is not required that all the replace commands be executed when the **buffer** is executed. Each replace command in the **buffer** can be independently triggered depending on the data present in the current DDF. Moreover, the presence in the **buffer** field of a NodeDeletion command at the n^{th} position indicates that the value of the DDF corresponding to the n^{th} field of the DDF definition shall be ignored.

The **eventTime** eventOut carrying the current time is generated after a DDF has been processed.

5.1.3.3 OutputActuator

5.1.3.3.1 XSD description

```
<ProtoDeclare name="OutputActuator" locations="org:mpeg:outputactuator">
  <field name="enabled" type="Boolean" vrml97Hint="exposedField" booleanValue="TRUE"/>
  <field name="url" type="Strings" vrml97Hint="exposedField" stringArrayValue= ""/>
  <!--Any number of eventIn fields!-->
</ProtoDeclare>
```

5.1.3.3.2 BIFS Textual description

```
EXTERNPROTO OutputActuator [
  eventIn      SFBool      activate
  exposedField SFBool      enabled      TRUE
  exposedField MFString    url          []
  Any number of the following may then follow:
  eventIn      eventType   DDFEventName
] ""org:mpeg:outputActuator"
```

5.1.3.3.3 Functionality and semantics

The **OutputActuator** proto is used to communicate between the scene and the MPEG-V actuator. How to map these commands to the physical device is out of the scope of this document. It should be noted that the device interprets the command and produces the effect immediately when the command is received. The proto definition of **OutputActuator** is described below.

The **OutputActuator** PROTO can receive variable number of events that in turn generate Device Data Frames (DDFs) that are sent to the actuator. Each eventIn corresponds to one field in the DDF and has the same type. When **activate** eventIn is received, the DDF is assembled and sent to the device.

When declaring an **OutputActuator** in a **BIFS** scene, the eventIn fields shall be placed in their order of appearance in the associated DDF, after all other fields are declared. The **activate** eventIn shall be declared first in the extern proto declaration.

The **url** field specifies the device to be controlled.

Only if the **enabled** field is TRUE then the DDFs are generated.

The **mandatory** input events of the OutputActuator interface are as described in [Table 3](#).

Table 3 — OutputActuator: input events

Actuator type	Input events, in the given order	Input event meaning
Light	SFFloat, SFCOLOR	intensity, color
Vibration	SFFloat	intensity
Tactile	MFFloat	intensity
Flash	SFFloat, SFCOLOR	intensity, color
Heating	SFFloat	intensity
Cooling	SFFloat	intensity
Wind	SFFloat	intensity
Sprayer	SFFloat, SFInt32	intensity, sprayingType
Scent	SFFloat, SFInt32	intensity, scent
Fog	SFFloat	intensity
Rigid Body Motion	MFVec3f, MFVec3f, MFVec3f, MFVec3f, MFVec3f, MFVec3f	direction, speed, acceleration, angle, angleSpeed, angleAcceleration
Kinesthetic	MFVec3f, MFVec3f, MFVec3f, MFVec3f	position, orientation, force, torque

In the following example, two actuators of different types (Light and Vibration) are presented. In this case, the EXTERNPROTO OutputActuator is declared twice as follows:

```
EXTERNPROTO LightActuator [
eventIn          SFFloat    activate
exposedField     SFFloat    enabled      TRUE
exposedField     MFString   url          [hw://lightDevice1]
eventIn          SFFloat    intensity
eventIn          SFCOLOR    color
]"org:mpeg:outputactuator"

EXTERNPROTO VibrationActuator [
eventIn          SFFloat    activate
exposedField     SFFloat    enabled      TRUE
exposedField     MFString   url          [hw://vibrationDevice1]
eventIn          SFFloat    intensity
]"org:mpeg:outputactuator"
```

This could be instantiated in the scene as follows:

```
DEF SI_LIGHT ScalarInterpolator {
  Key          [0 0.5 1]
  keyValues    [0.1 0.2 0.3]
}

DEF SI_VIBRATION ScalarInterpolator {
  Key          [0 0.5 1]
  keyValues    [0.1 0.2 0.3]
}

DEF LIGHT_1 LightActuator {}
DEF LIGHT_2 LightActuator {
  url          [hw://lightDevice2]
}

DEF VIBRATION_1 VibrationActuator {}
DEF VIBRATION_2 VibrationActuator {
  enabled      FALSE
  url          [hw://vibrationDevice2]
}

ROUTE SI_LIGHT.value_changed TO LIGHT_1.intensity
ROUTE SI_LIGHT.value_changed TO LIGHT_2.intensity

ROUTE SI_VIBRATION.value_changed TO VIBRATION_1.intensity
ROUTE SI_VIBRATION.value_changed TO VIBRATION_2.intensity
```

5.1.3.4 SphereSensor

5.1.3.4.1 XSD description

```
<complexType name="SphereSensorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="autoOffset" type="xmta:SFBool" use="optional" default="true"/>
  <attribute name="enabled" type="xmta:SFBool" use="optional" default="true"/>
  <attribute name="offset" type="xmta:SFRotation" use="optional" default="0 1 0 0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="SphereSensor" type="xmta:SphereSensorType"/>
```

5.1.3.4.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.44.

The SphereSensor node maps pointing device motion into spherical rotation about the origin of the local coordinate system. The SphereSensor node uses the descendent geometry of its parent node to determine whether it is liable to generate events.

The *enabled* exposed field enables and disables the SphereSensor node. If *enabled* is TRUE, the sensor reacts appropriately to user events. If *enabled* is FALSE, the sensor does not track user input or send events. If *enabled* receives a FALSE event and *isActive* is TRUE, the sensor becomes disabled and deactivated, and outputs an *isActive* FALSE event. If *enabled* receives a TRUE event, the sensor is enabled and ready for user activation.

The SphereSensor node generates events when the pointing device is activated while the pointer is indicating any descendent geometry nodes of the sensor's parent group. See ISO/IEC 14772-1:1997, 4.6.7.5, for details on using the pointing device to activate the SphereSensor.

Upon activation of the pointing device (e.g. mouse button down) over the sensor's geometry, an *isActive* TRUE event is sent. The vector defined by the initial point of intersection on the SphereSensor's geometry and the local origin determines the radius of the sphere that is used to map subsequent pointing device motion while dragging. The virtual sphere defined by this radius and the local origin at the time of activation is used to interpret subsequent pointing device motion and is not affected by any changes to the sensor's coordinate system while the sensor is active. For each position of the bearing, a *rotation_changed* event is sent which corresponds to the sum of the relative rotation from the original intersection point plus the *offset* value. *trackPoint_changed* events reflect the unclamped drag position on the surface of this sphere. When the pointing device is deactivated and *autoOffset* is TRUE, *offset* is set to the last *rotation_changed* value and an *offset_changed* event is generated. See ISO/IEC 14772-1:1997, 4.6.7.4 for more details.

When the sensor generates an *isActive* TRUE event, it grabs all further motion events from the pointing device until it is released and generates an *isActive* FALSE event (other pointing-device sensors shall not generate events during this time). Motion of the pointing device while *isActive* is TRUE is termed a "drag". If a 2D pointing device is in use, *isActive* events will typically reflect the state of the primary button associated with the device (i.e. *isActive* is TRUE when the primary button is pressed and FALSE when it is released). If a 3D pointing device (e.g. wand) is in use, *isActive* events will typically reflect whether the pointer is within (or in contact with) the sensor's geometry.

While the pointing device is activated, *trackPoint_changed* and *rotation_changed* events are output. *trackPoint_changed* events represent the unclamped intersection points on the surface of the invisible sphere. If the pointing device is dragged off the sphere while activated, browsers may interpret this in a variety of ways (e.g. clamp all values to the sphere or continue to rotate as the point is dragged away from the sphere). Each movement of the pointing device while *isActive* is TRUE generates *trackPoint_changed* and *rotation_changed* events.

5.1.3.5 TimeSensor

5.1.3.5.1 XSD description

```
<complexType name="TimeSensorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="cycleInterval" type="xmta:SFloat" use="optional" default="1"/>
  <attribute name="enabled" type="xmta:SBoolean" use="optional" default="true"/>
  <attribute name="loop" type="xmta:SBoolean" use="optional" default="false"/>
  <attribute name="startTime" type="xmta:SFloat" use="optional" default="0"/>
  <attribute name="stopTime" type="xmta:SFloat" use="optional" default="0"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="TimeSensor" type="xmta:TimeSensorType"/>
```

5.1.3.5.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.50.

TimeSensor nodes generate events as time passes. TimeSensor nodes can be used for many purposes including:

- driving continuous simulations and animations;
- controlling periodic activities (e.g. one per minute);
- initiating single occurrence events such as an alarm clock.

The TimeSensor node contains two discrete eventOuts: *isActive* and *cycleTime*. The *isActive* eventOut sends TRUE when the TimeSensor node begins running, and FALSE when it stops running. The *cycleTime* eventOut sends a time event at *startTime* and at the beginning of each new cycle (useful for synchronization with other time-based objects). The remaining eventOuts generate continuous events. The *fraction_changed* eventOut, an SFloat in the closed interval [0.1], sends the completed fraction of the current cycle. The *time* eventOut sends the absolute time for a given *simulation tick*.

If the *enabled* exposedField is TRUE, the TimeSensor node is enabled and may be running. If a *set_enabled* FALSE event is received while the TimeSensor node is running, the sensor performs the following actions:

- evaluates and sends all relevant outputs;
- sends a FALSE value for *isActive*;
- disables itself.

Events on the exposedFields of the TimeSensor node (e.g. *set_startTime*) are processed and their corresponding eventOuts (e.g. *startTime_changed*) are sent regardless of the state of the *enabled* field. The remaining discussion assumes *enabled* is TRUE.

The *loop*, *startTime*, and *stopTime* exposedFields and the *isActive* eventOut and their effects on the TimeSensor node are discussed in detail in ISO/IEC 14772:1997, 4.6.9. The "cycle" of a TimeSensor node lasts for *cycleInterval* seconds. The value of *cycleInterval* shall be greater than zero.

A *cycleTime* eventOut can be used for synchronization purposes such as sound with animation. The value of a *cycleTime* eventOut will be equal to the time at the beginning of the current cycle. A *cycleTime* eventOut is generated at the beginning of every cycle, including the cycle starting at *startTime*. The first *cycleTime* eventOut for a TimeSensor node can be used as an alarm (single pulse at a specified time).

When a TimeSensor node becomes active, it generates an *isActive* = TRUE event and begins generating *time*, *fraction_changed*, and *cycleTime* events which may be routed to other nodes to drive animation or simulated behaviours. The behaviour at read time is described below. The *time* event sends the absolute

time for a given tick of the TimeSensor node (time fields and events represent the number of seconds since midnight GMT January 1, 1970).

fraction_changed events output a floating point value in the closed interval [0, 1]. At *startTime* the value of *fraction_changed* is 0. After *startTime*, the value of *fraction_changed* in any cycle will progress through the range [0.0, 1.0]. At $startTime + N \times cycleInterval$, for $N = 1, 2, \dots$, that is, at the end of every cycle, the value of *fraction_changed* is 1.

Let *now* represent the time at the current simulation tick. Then the *time* and *fraction_changed* eventOuts can then be computed as:

```
time = now
temp = (now - startTime) / cycleInterval
f = fractionalPart(temp)
if (f == 0.0 && now > startTime) fraction_changed = 1.0
else fraction_changed = f
```

where fractionalPart(x) is a function that returns the fractional part, (that is, the digits to the right of the decimal point), of a nonnegative floating point number.

A TimeSensor node can be set up to be active at read time by specifying *loop* TRUE (not the default) and *stopTime* less than or equal to *startTime* (satisfied by the default values). The *time* events output absolute times for each tick of the TimeSensor node simulation. The *time* events shall start at the first simulation tick greater than or equal to *startTime*. *time* events end at *stopTime*, or at $startTime + N \times cycleInterval$ for some positive integer value of N , or loop forever depending on the values of the other fields. An active TimeSensor node shall stop at the first simulation tick when $now \geq stopTime > startTime$.

No guarantees are made with respect to how often a TimeSensor node generates time events, but a TimeSensor node shall generate events at least at every simulation tick. TimeSensor nodes are guaranteed to generate final *time* and *fraction_changed* events. If *loop* is FALSE at the end of the N th *cycleInterval* and was TRUE at $startTime + M \times cycleInterval$ for all $0 < M < N$, the final *time* event will be generated with a value of $(startTime + N \times cycleInterval)$ or *stopTime* (if $stopTime > startTime$), whichever value is less. If *loop* is TRUE at the completion of every cycle, the final event is generated as evaluated at *stopTime* (if $stopTime > startTime$) or never.

An active TimeSensor node ignores *set_cycleInterval* and *set_startTime* events. An active TimeSensor node also ignores *set_stopTime* events for *set_stopTime* less than or equal to *startTime*. For example, if a *set_startTime* event is received while a TimeSensor node is active, that *set_startTime* event is ignored (the *startTime* field is not changed, and a *startTime_changed* eventOut is not generated). If an active TimeSensor node receives a *set_stopTime* event that is less than the current time, and greater than *startTime*, it behaves as if the *stopTime* requested is the current time and sends the final events based on the current time (note that *stopTime* is set as specified in the eventIn).

A TimeSensor read from a VRML file shall generate *isActive* TRUE, *time* and *fraction_changed* events if the sensor is enabled and all conditions for a TimeSensor to be active are met.

5.1.3.6 TouchSensor

5.1.3.6.1 XSD description

```
<complexType name="TouchSensorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
  <attribute name="enabled" type="xmta:SFBBool" use="optional" default="true"/>
  <attributeGroup ref="xmta:DefUseGroup"/>
</complexType>
<element name="TouchSensor" type="xmta:TouchSensorType"/>
```

5.1.3.6.2 Functionality and semantics

As specified in ISO/IEC 14772-1:1997, 6.51.

A TouchSensor node tracks the location and state of the pointing device and detects when the user points at geometry contained by the TouchSensor node's parent group. A TouchSensor node can be enabled or disabled by sending it an *enabled* event with a value of TRUE or FALSE. If the TouchSensor node is disabled, it does not track user input or send events.

The TouchSensor generates events when the pointing device points toward any geometry nodes that are descendants of the TouchSensor's parent group. See ISO/IEC 14772-1:1997, 4.6.7.5, for more details on using the pointing device to activate the TouchSensor.

The *isOver* eventOut reflects the state of the pointing device with regard to whether it is pointing towards the TouchSensor node's geometry or not. When the pointing device changes state from a position such that its bearing does not intersect any of the TouchSensor node's geometry to one in which it does intersect geometry, an *isOver* TRUE event is generated. When the pointing device moves from a position such that its bearing intersects geometry to one in which it no longer intersects the geometry, or some other geometry is obstructing the TouchSensor node's geometry, an *isOver* FALSE event is generated. These events are generated only when the pointing device has moved and changed "over" state. Events are not generated if the geometry itself is animating and moving underneath the pointing device.

As the user moves the bearing over the TouchSensor node's geometry, the point of intersection (if any) between the bearing and the geometry is determined. Each movement of the pointing device, while *isOver* is TRUE, generates *hitPoint_changed*, *hitNormal_changed* and *hitTexCoord_changed* events. *hitPoint_changed* events contain the 3D point on the surface of the underlying geometry, given in the TouchSensor node's coordinate system. *hitNormal_changed* events contain the surface normal vector at the *hitPoint*. *hitTexCoord_changed* events contain the texture coordinates of that surface at the *hitPoint*. The values of *hitTexCoord_changed* and *hitNormal_changed* events are computed as appropriate for the associated shape.

If *isOver* is TRUE, the user may activate the pointing device to cause the TouchSensor node to generate *isActive* events (e.g. by pressing the primary mouse button). When the TouchSensor node generates an *isActive* TRUE event, it grabs all further motion events from the pointing device until it is released and generates an *isActive* FALSE event (other pointing-device sensors will not generate events during this time). Motion of the pointing device while *isActive* is TRUE is termed a "drag." If a 2D pointing device is in use, *isActive* events reflect the state of the primary button associated with the device (i.e. *isActive* is TRUE when the primary button is pressed and FALSE when it is released). If a 3D pointing device is in use, *isActive* events will typically reflect whether the pointing device is within (or in contact with) the TouchSensor node's geometry.

The eventOut field *touchTime* is generated when all three of the following conditions are true.

- The pointing device was pointing towards the geometry when it was initially activated (*isActive* is TRUE).
- The pointing device is currently pointing towards the geometry (*isOver* is TRUE).
- The pointing device is deactivated (*isActive* FALSE event is also generated).

In a 2D context, there are restrictions on the SFVec3f eventOuts:

- *hitNormal_changed* always returns [0.0, 0.0, 1.0];
- *hitPoint_changed* always has 0.0 as Z coordinate.

5.1.3.7 MediaSensor

5.1.3.7.1 XSD description

```
<complexType name="MediaSensorType">
  <all>
    <element ref="xmta:IS" minOccurs="0"/>
  </all>
```